Ф. А. Новиков

ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ

3-е издание

Допущено Министерством образования и науки Российской Федерации
в качестве учебного пособия для студентов высших учебных заведений,
обучающихся по направлению подготовки дипломированных
специалистов «Информатика и вычислительная техника»

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2009

ББК 22.174я7
УДК 519.1(075)
Н73

Рецензенты:

Кафедра прикладной математики
Санкт-Петербургского государственного технического университета

***С. С. Лавров**, доктор технических наук, профессор,*
член-корреспондент Российской академии наук

Новиков Ф. А.

Н73 Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб.: Питер, 2009. — 384 с.: ил. — (Серия «Учебник для вузов»).

ISBN 978-5-91180-759-7

В учебнике изложены основные разделы дискретной математики и описаны важнейшие алгоритмы на дискретных структурах данных. Основу книги составляет материал лекционного курса, который автор читает в Санкт-Петербургском государственном техническом университете последние полтора десятилетия. Третье издание имеет ту же структуру и последовательность изложения, что и второе. В книгу внесено несколько десятков не очень объемных, но существенных добавлений, уточнений и определений. Обновлены упражнения, библиография и комментарии к ней.

Для студентов вузов, практикующих программистов и всех желающих изучить дискретную математику.

Допущено Министерством образования и науки Российской Федерации в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки дипломированных специалистов «Информатика и вычислительная техника».

ББК 22.174я7
УДК 519.1(075)

ISBN978-5-91180-759-7

Краткое содержание

Содержание

Предисловие к третьему изданию

В 2000 году издательство «Питер» выпустило первое издание этого учебника. В то время выбор учебной литературы по дискретной математике был небогат. За последние годы ситуация резко изменилась. Переведено достаточное количество добротных американских учебников и несколько отличных книг написано российскими авторами. Тем не менее предлагаемый учебник по-прежнему имеет моральное право на существование, поскольку комбинация основных отличительных особенностей (компактность изложения, плюс широта охвата материала, плюс внимание к деталям программной реализации), по мнению автора, остаётся привлекательной для широкой категории читателей.

Третье издание книги имеет ту же структуру, последовательность изложения и систему обозначений, что и второе издание. В книгу внесено несколько десятков не очень объёмных, но существенных добавлений, уточнений и определений. Обновлены упражнения, библиография и комментарии к ней.

Кроме того, исправлено несколько сотен опечаток, упущений и ошибок. Общее количество изменений значительно больше количества страниц. Если пользоваться жаргоном программистов, которым адресована эта книга, то можно сказать, что проведён «рефакторинг» всего материала.

Третье издание прошло профессиональное научное редактирование, за что автор выражает глубокую благодарность научному редактору.

Ф. А. Новиков

Предисловие ко второму изданию

В 2000 году издательство «Питер» выпустило в свет учебник Ф. А. Новикова «Дискретная математика для программистов». Книга оказалась удачной, о чём свидетельствуют тиражи и письма читателей. Резонно предположить, что объём, стиль и содержание книги отвечают возрастающим потребностям практических программистов. В то же время за прошедшие три года читатели и коллеги указали на многочисленные огрехи в изложении материала.

Вниманию читателей предлагается второе издание, исправленное и дополненное, в котором предпринята попытка устранить недостатки, сохранив достоинства первого издания. Сохранена предельная лаконичность в изложении материала: по мнению автора, каждая страница, без которой можно обойтись, является заметным недостатком. Сохранена мелкая рубрикация: раздел нижнего уровня помещается на одной, редко на двух страницах и может быть прочтен за один присест, не поднимая головы. Сохранена ориентация на программистов: описание объектов дискретной математики в большинстве случаев доводится до кода алгоритмов работы с этими объектами. Устранены замеченные ошибки: за три года автор, рецензенты, студенты и читатели заметили более 200 фактических неточностей и опечаток в книге — все замеченные опечатки устранены. Изменена система обозначений с целью устранения излишних вольностей.

В первом издании книги было 12 глав, во втором их десять. Сокращение количества глав по сравнению с первым изданием не означает сокращения объёма рассматриваемого материала. Это продиктовано желанием автора сделать структуру книги более уравновешенной. Из книги действительно удалено несколько второстепенных вопросов, которые в первом издании рассматривались довольно поверхностно. Хочется надеяться, что за счёт этого рассмотрение оставшихся вопросов удалось сделать более подробным.

Обновлен и дополнен список литературы. Как и в первом издании, этот список не имеет цели отразить историю предмета или дать полное библиографическое описание. Это рекомендации для студентов по дополнительному чтению, отражающие вкусы автора.

Ф. А. Новиков

Вступительное слово к первому изданию

Автор этой книги Ф. А. Новиков имеет большой опыт практического программирования, чтения лекций по дискретной математике и написания книг, посвящённых различным вопросам вычислительной техники и её программного обеспечения. Все это позволило ему создать книгу, наполненную обширным и интересным материалом. Она предназначена для студентов младших курсов, специализирующихся в области программирования, но будет полезна не только им, но и всем тем, кто обучается или стремится повысить квалификацию в направлениях, тесно связанных с программированием, вплоть до аспирантов.

Ф. А. Новиков охватывает ряд направлений дискретной математики: теорию множеств и алгебраические структуры, логику и булевы функции (причём затронута даже нетрадиционная проблема автоматического доказательства теорем), комбинаторику и кодирование. Особое внимание уделено общей теории графов — одному из важнейших инструментов программиста, и главным её приложениям. Вся книга наполнена примерами конкретных алгоритмов от простых до достаточно сложных, особенно во второй половине книге. Это не только полезный учебный материал, но и багаж, который не окажется излишним в будущей практической деятельности учащихся. Книг подобной направленности и с подобным подбором материала в моём поле зрения почти не было.

Книга снабжена списком русскоязычной литературы, из которой читатель сможет извлечь дополнительные сведения по заинтересовавшим его вопросам. Каждый источник из этого списка кратко охарактеризован в конце главы, к которой он относится.

Содержание книги во всех её разделах продуманно и конкретно. Решение автора не включать в книгу такие темы, как теория алгорифмов, надо считать правильным — учебный курс не должен быть перегружен.

Книга написана хорошим языком и, можно надеяться, будет благосклонно принята читателем и окажется для него хорошим подспорьем, в частности, при построении математической модели возникшей перед человеком задачи и при выборе подходящего представления данных. Тому и другому автор уделяет неизменное внимание. Поучительно также краткое, но в большинстве случаев достаточно убедительное, обоснование правильности предлагаемых алгоритмов.

Профессор, д. т. н., чл.-корр. РАН *С. С. Лавров*

Введение

В этой книге рассматриваются некоторые элементарные понятия «дискретной», или «конечной», математики, то есть математики, прежде всего изучающей конечные множества и различные структуры на них. Это означает, что понятия бесконечности, предела или непрерывности не являются предметом изучения, хотя могут использоваться как вспомогательные средства. Дискретная математика имеет широкий спектр приложений, прежде всего в областях, связанных с информационными технологиями и компьютерами. В самом первоначальном, ныне редко используемом названии компьютера — «электронная цифровая вычислительная машина» — слово «цифровая» указывает на принципиально дискретный характер работы данного устройства. Современные компьютеры неотделимы от современных программистов, для которых и написана эта книга.

Назначение и особенности книги

Данная книга представляет собой учебник по дискретной математике, включающий в себя описания важнейших алгоритмов над объектами дискретной математики. Учебник ориентирован на студентов программистских специальностей и практикующих программистов, которым по роду их занятий приходится иметь дело с конструированием и анализом алгоритмов.

В настоящее время имеется масса доступной литературы, покрывающей обе эти темы. С одной стороны, существуют десятки прекрасных книг по дискретной математике, начиная с элементарных учебников для начинающих и заканчивая исчерпывающими справочниками для специалистов. С другой стороны, большое число монографий, многие из которых стали классическими, посвящены способам конструирования эффективных алгоритмов. Как правило, такие монографии содержат детальное описание и анализ важнейших и известнейших алгоритмов. Однако книги, которые бы рассматривали обе эти темы одновременно и во взаимосвязи, практически отсутствуют. В качестве редких исключений можно назвать книгу В. Липского «Комбинаторика для программистов» [8], перевод которой выдержал уже два издания в России, и всеобъемлющую энциклопедию Д. Кнута «Искусство программирования для ЭВМ» [14]–[16]. Первая из упомянутых книг очень невелика по объёму и в значительной степени пересекается по выбранному материалу с данным учебником. Вторая отличается

максимальной глубиной изложения и большим объёмом, но не затрагивает некоторых важных для программистов разделов дискретной математики. Данный учебник принадлежит именно к такому жанру математической литературы, в котором математическое изложение доводится до уровня практически исполнимых программ. Он отличается большей *широтой*, но, пожалуй, меньшей *глубиной* охвата материала.

Учебник основан на лекционном курсе, который автор в течение многих лет читает студентам кафедры «Прикладная математика» Санкт-Петербургского государственного политехнического университета, что наложило определённый отпечаток на выбор материала. Учебник охватывает почти все основные разделы дискретной математики: теорию множеств, математическую логику, общую алгебру, комбинаторику и теорию графов. Кроме того, представлены и некоторые более специальные разделы, необходимые программистам, такие как теория кодирования и булевы функции. Данный список можно было бы продолжить, включив в него, например, вычислительную геометрию, линейное программирование и т. д., но основная цель состояла в том, чтобы написать именно краткий учебник, и, ограничив себя в объёме, пришлось ограничить и набор рассматриваемых тем.

В целом учебник преследует три основные цели:

1. Познакомить читателя с максимально широким кругом понятий дискретной математики. Количество определяемых и упоминаемых понятий и специальных терминов намного превышает количество понятий, обсуждаемых более детально. Тем самым у студента формируется терминологический запас, необходимый для самостоятельного изучения специальной математической и теоретико-программистской литературы.
2. Сообщить читателю необходимые конкретные сведения из дискретной математики, предусматриваемые стандартной программой технических высших учебных заведений. Разбор доказательств приведенных утверждений и выполнение упражнений позволят студенту овладеть методами дискретной математики, наиболее употребительными при решении практических задач.
3. Пополнить запас примеров нетривиальных алгоритмов. Изучение алгоритмов решения типовых задач дискретной математики и способов представления математических объектов в программах абсолютно необходимо практикующему программисту, поскольку позволяет уменьшить трудозатраты на «изобретение велосипеда» и существенно обогащает навыки конструирования алгоритмов.

Структура книги

Книга содержит 10 глав, которые, фактически, делятся на три группы. Первая группа, несколько бóльшая по объёму (главы 1, 2 и 5), содержит самые общие сведения из основных разделов дискретной математики. Доля алгоритмов в главах первой группы несколько меньше, а доля определений несколько

больше, чем во второй и третьей группах. В второй группе (главы 3, 4 и 6) собраны различные специальные разделы, которые можно включать или не включать в учебный курс в зависимости от конкретных обстоятельств. Третья группа глав (главы 7–10) целиком посвящена теории графов, поскольку связанные с ней вопросы (в частности, алгоритмы над графами) являются разделами дискретной математики, наиболее широко применяющимися в практическом программировании.

Главы делятся на разделы, которые, в свою очередь, делятся на подразделы. Каждый раздел посвящён одному конкретному вопросу темы главы и по объёму соответствует экзаменационному вопросу. Подразделы нужны для внутренних ссылок и более детальной структуризации материала. Как правило, в подразделе рассматривается какое-нибудь одно понятие, теорема или алгоритм. Подразделы в пределах раздела упорядочены по сложности: сначала приводятся основные определения, а затем рассматриваются более сложные вопросы, которые, в принципе, при первом чтении можно пропустить без ущерба для понимания остального материала.

Главы книги более или менее независимы и могут изучаться в любом порядке, за исключением первой главы, которая содержит набор базисных определений для дальнейшего изложения, и главы 7, в которой вводится специфическая терминология теории графов.

В конце каждой главы имеются два специальных раздела: «Комментарии» и «Упражнения». В комментариях приводится очень краткий обзор литературы по теме главы и даются ссылки на источники, содержащие более детальные описания упомянутых алгоритмов. Эти разделы не претендуют на статус исчерпывающего библиографического обзора, скорее это рекомендации для студентов по чтению для самообразования. Выбор источников, как правило, отражает вкусы автора. Упражнения немногочисленны — ровно по одному на каждый раздел — и очень просты. Как правило, в упражнения выносится дополнительный материал, непосредственно связанный с темой раздела (например, опущенные доказательства утверждений, аналогичных доказанным в основном тексте).

Используемые обозначения

Данная книга предназначена для программистов, то есть предполагается, что читатель не испытывает затруднений в понимании текстов программ на языке высокого уровня. Именно поэтому в качестве нотации для записи алгоритмов используется *некоторый* неспецифицированный язык программирования (*псевдокод*), похожий по синтаксису на Паскаль (но, конечно, Паскалем не являющийся).

Ключевые слова псевдокода выделены жирным шрифтом, идентификаторы объектов записываются курсивом, как это принято для математических объектов, примечания заключаются в фигурные скобки { }.

В языке используются общеупотребительные структуры управления: ветвления в краткой (**if** ... **then** ... **end if**) и полной (**if** ... **then** ... **else** ... **end if**) формах, циклы со счётчиком (**for** ... **from** ... **to** ... **do** ... **end for**), с постусловием (**repeat** ... **until**), с предусловием (**while** ... **do** ... **end while**) и по множеству (**for** ... **do** ... **end for**), а также переходы (**go to** ...)[1].

Для обозначения присваивания используется традиционный знак :=, вызов процедуры или функции ключевыми словами не выделяется, выход из процедуры вместе с возвратом значения обозначается ключевым словом **return**.

Подразумевается строгая статическая типизация, даже приведения не используются, за исключением разыменования, которое подразумевается везде, где оно нужно по здравому программистскому смыслу. В то же время объявления локальных переменных почти всегда опускаются, если их тип ясен из контекста.

Из структур данных считаются доступными массивы (**array** [...] **of** ...), структуры (**record** ... **end record**) и указатели (↑).

В программах широко используются математические обозначения, которые являются самоочевидными в конкретном контексте. Например, конструкция

for $x \in M$ **do**
  $P(x)$
end for

означает применение процедуры P ко всем элементам множества M.

«Лишние» разделители систематически опускаются, функции могут возвращать несколько значений, и вообще, разрешаются любые вольности, которые позволяют сделать тексты программ более краткими и читабельными.

Особого упоминания заслуживают три «естественных» оператора, аналоги которых в явном виде редко встречаются в настоящих языках программирования. Оператор

select $m \in M$

означает выбор *произвольного* элемента m из множества M. Этот оператор часто необходим в «переборных» алгоритмах. Оператор

yield x

означает возврат значения x, но при этом выполнение функции не прекращается, а продолжается со следующего оператора. Этот оператор позволяет очень просто записать «генерирующие» алгоритмы, результатом которых является некоторое заранее неизвестное множество значений. Оператор

next for x

означает прекращение выполнения текущего фрагмента программы и продолжение выполнения со следующего шага цикла по переменной x. Этот оператор должен находиться в теле цикла по x (на любом уровне вложенности). Такого

[1] «Структурных» программистов, сомневающихся в допустимости использования оператора **goto** в учебнике, автор отсылает к статье Д. Кнута «Structured Programming with **go to** statements».

рода «структурные переходы» позволяют описывать обработку исключительных ситуаций в циклах непосредственно, без введения искусственных переменных, отслеживающих состояние вычислений в цикле.

Операторы **select**, **yield** и **next for** не являются чем-то необычным и новым: первый из них легко моделируется использованием датчика псевдослучайных чисел, второй — присоединением элемента к результирующему множеству, а третий — переходом по метке. Однако указанные операторы сокращают запись и повышают её наглядность.

Поскольку эта книга написана на «программно-математическом» языке, в ней не только математические обозначения используются в программах, но и некоторые программистские обозначения используются в математическом тексте.

Прежде всего обсудим способы введения обозначений объектов и операций. Если обозначение объекта или операции является глобальным (в пределах учебника), то используется знак $\stackrel{\text{Def}}{=}$, который следует читать «по определению есть». При этом слева от знака $\stackrel{\text{Def}}{=}$ может стоять обозначение объекта, а справа — выражение, определяющее этот объект. Например,

$$\mathbb{R}_+ \stackrel{\text{Def}}{=} \{x \in \mathbb{R} \mid x > 0\}.$$

В случае определения операции в левой части стоит *выражение*. В этом случае левую часть следует неформально понимать как заголовок процедуры выполнения определяемой операции, а правую часть — как тело этой процедуры. Например, формула

$$A = B \stackrel{\text{Def}}{=} A \subset B \;\&\; B \subset A$$

означает следующее: «чтобы вычислить значение выражения $A = B$, нужно вычислить значения выражений $A \subset B$ и $B \subset A$, а затем вычислить конъюнкцию этих значений».

Отметим широкое использование символа присваивания $:=$, который в математическом контексте можно читать как «положим». Если в левой части такого присваивания стоит простая переменная, а в правой части — выражение, определяющее конкретный объект, то это имеет очевидный смысл введения локального (в пределах доказательства, определения и т. п.) обозначения. Например, запись $Z' := Z \cup Z_{e_k} \setminus \{e_k\}$ означает: «положим, что Z' содержит все элементы Z и Z_{e_k} за исключением e_k». Наряду с обычным для математики «статическим» использованием знака $:=$, когда выражению в левой части придается постоянное в данном контексте значение (определение константы), знак присваивания используется и в «динамическом», программистском смысле. Например, пусть в графе G есть вершина v. Тогда формулу $G := G - v$ следует читать и понимать так: «удалим в графе G вершину v».

Некоторые обозначения являются глобальными, то есть означают одно и то же в любом контексте (в этой книге). Например, буква $\mathbb{R}$ всегда обозначает поле вещественных чисел, а пара вертикальных чёрточек слева и справа от объекта $|X|$ — количество элементов в объекте X. Таких стандартных обозначений не

так уж много. Они собраны в указателе обозначений в конце книги и используются в тексте учебника без дополнительных пояснений. Если смысл какого-то обозначения не ясен и не определён в данном контексте, то следует посмотреть это обозначение в указателе обозначений.

Одной из задач книги является выработка у студентов навыка чтения математических текстов. Поэтому, начиная с самой первой страницы, без дополнительных объяснений интенсивно используются язык исчисления предикатов и другие общепринятые математические обозначения. При этом стиль записи формул совершенно свободный и неформальный, но соответствующий общепринятой практике записи формул в математических текстах. Например, вместо формулы

$$\forall k \ ((k < n) \Longrightarrow P(k))$$

может быть написано

$$\forall k < n \ (P(k))$$

или даже

$$\forall k < n \ P(k)$$

в предположении, что читатель знает или догадается, какие синтаксические упрощения используются. В первых главах книги основные утверждения (формулировки теорем) дублируются, то есть приводятся на естественном языке и на языке формул. Тем самым на примерах объясняется используемый язык формул. Но в последних частях книги и в доказательствах использование естественного языка сведено к минимуму. Это существенно сокращает объём текста, но требует внимания при чтении.

Для краткости в тексте книги иногда используются обороты, которые подразумевают восстановление читателем опущенных слов по контексту. Например, если символ A означает множество, то вместо аккуратного, но длинного оборота «где объект x является элементом множества A», может быть использована более короткая фраза «где x — элемент A» или даже формула «где $x \in A$».

В целом используемые обозначения строго следуют классическим образцам, принятым в учебной математической и программистской литературе. Непривычным может показаться только совместное использование этих обозначений в одном тексте. Но такое взаимопроникновение обозначений продиктовано основной задачей книги.

Выделения в тексте

В учебнике имеются следующие основные виды текстов: определения, теоремы, леммы и следствия, доказательства и обоснования, замечания, отступления, алгоритмы и примеры. Фактически, обычный связующий текст сведен к минимуму в целях сокращения объёма книги.

Определения никак специально не выделяются, поскольку составляют львиную долю основного текста книги. Вместо этого курсивом выделяются *определяющие вхождения* терминов, а текст, соседствующий с термином, и есть определение.

Все определяющие вхождения вынесены в предметный указатель, помещённый в конце книги. Таким образом, если при чтении попадается незнакомый термин, следует найти его определение с помощью указателя (или убедиться, что определения в книге нет). В третьем издании математические термины в предметном указателе переведены на английский язык. Автор надеется, что предметный указатель послужит читателю как краткий толковый математический русско-английский словарь.

Формулировки теорем, лемм и следствий в соответствии с общепринятой в математической литературе практикой выделены *курсивом*. При этом формулировке предшествует соответствующее ключевое слово: «теорема», «лемма» или «следствие». Как правило, утверждения не нумеруются, за исключением случаев вхождения нескольких однородных утверждений в один подраздел. Для ссылок на утверждения используются либо номера подразделов, в которых утверждения сформулированы, либо собственные имена утверждений. Дело в том, что в данном учебнике теоремами оформлены как простые утверждения, являющиеся непосредственными следствиями определений, так и глубокие факты, действительно заслуживающие статуса теоремы. В последнем случае приводится собственное имя (название теоремы), которое либо выносится в название подраздела (или раздела), либо указывается в скобках после слова «теорема».

Доказательства относятся к предшествующим утверждениям, а *обоснования* — к предшествующим алгоритмам. В учебнике практически нет недоказанных утверждений и приведенных без достаточного обоснования алгоритмов. Из этого правила имеется несколько исключений, то есть утверждений, доказательства которых не приводятся, поскольку автор считает их технически слишком сложными для выбранного уровня изложения. Отсутствие технически сложного доказательства всегда явно отмечается в тексте. Иногда же доказательство опускается, потому что утверждение легко может быть обосновано читателем самостоятельно. Такие случаи не отмечаются в тексте. Доказательства и обоснования начинаются с соответствующего ключевого слова («доказательство» или «обоснование») и заканчиваются специальным значком □. Если формулировка теоремы содержит несколько утверждений или если доказательство состоит из нескольких шагов (частей), то оно делится на абзацы, а в начале каждого абзаца указывается [в скобках], что именно в нем доказывается.

Замечания и *отступления* также начинаются с соответствующего ключевого слова: «замечание» или «отступление». Назначение этих абзацев различно. Замечание сообщает некоторые специальные или дополнительные сведения, прямо относящиеся к основному материалу учебника. Отступление не связано непосредственно с основным материалом, его назначение — расширить кругозор читателя и показать связь основного материала с вопросами, лежащими за пределами курса.

ЗАМЕЧАНИЕ

В очень редких случаях *курсив* используется не для выделения определяющих вхождений терминов и формулировок утверждений, а для того, чтобы сделать эмфатическое ударение на каком-то слове в тексте.

ОТСТУПЛЕНИЕ

Использование отступлений необходимо, в противном случае у читателя может сложиться ошибочное впечатление о замкнутости изучаемого предмета и его оторванности от других областей знания.

Алгоритмы, как уже было сказано, записаны на псевдокоде, синтаксис которого кратко описан выше. Как правило, перед текстом алгоритма на естественном языке указываются его назначение, а также входные и выходные данные. Ключевые слова в текстах алгоритмов выделяются полужирным шрифтом. Исполняемые операторы, как правило, комментируются, чтобы облегчить их понимание. После алгоритма приводятся его обоснование и иногда пример протокола выполнения.

Примеры, как правило, приводятся непосредственно вслед за определением понятия, поэтому не используется никаких связующих слов, поясняющих, к чему относятся примеры. В самих примерах интенсивно используются факты, которые должны быть известны читателю из курса математики средней школы, и понятия, рассмотренные в предшествующем тексте книги.

Благодарности

Автор выражает благодарность члену-корреспонденту РАН Святославу Сергеевичу Лаврову за сотни ценных замечаний по тексту первого издания книги, многочисленным студентам, прослушавшим этот курс, за выявление ошибок в тексте, и научному редактору, профессору Никите Юрьевичу Нецветаеву, за огромную помощь в работе над третьим изданием.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция). Можно обратиться непосредственно к автору по адресу fedornovikov@rambler.ru Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

Глава 1 Множества и отношения

Понятия «множество», «отношение», «функция» и близкие к ним составляют основной словарь дискретной (равно как и «непрерывной») математики. Именно эти базовые понятия рассматриваются в первой главе, тем самым закладывается необходимая основа для дальнейших построений. Особенность изложения состоит в том, что здесь рассматриваются почти исключительно *конечные* множества, а тонкие и сложные вопросы, связанные с рассмотрением бесконечных множеств, излагаются с «программистской» точки зрения. С другой стороны, значительное внимание уделяется «представлению» множеств в программах. Эти вопросы не имеют прямого отношения к собственно теории множеств в её классическом виде, но очень важны для практикующего программиста.

1.1. Множества

При построении доступной для рационального анализа картины мира часто используется термин «объект» для обозначения некой сущности, отделимой от остальных. Выделение объектов — это не более чем произвольный акт нашего сознания. В одной и той же ситуации объекты могут быть выделены по-разному, в зависимости от точки зрения, целей анализа и других обстоятельств. Но как бы то ни было, выделение объектов и их совокупностей — естественный (или даже единственно возможный) способ организации нашего мышления, поэтому неудивительно, что он лежит в основе главного инструмента описания точного знания — математики.

1.1.1. Элементы и множества

Понятие множества принадлежит к числу фундаментальных понятий математики. Можно сказать, что *множество* — это любая определённая совокупность объектов. Объекты, из которых составлено множество, называются его *элементами*. Элементы множества различны и отличимы друг от друга. Как множествам, так и элементам можно давать имена или присваивать символьные обозначения.

ЗАМЕЧАНИЕ

В современной математике основным способом определения фундаментальных понятий является *аксиоматический метод*. В рамках этого метода группа понятий, образующих основу некоторой теории, определяется путем постулирования набора *аксиом* (утверждений об этих понятиях, принимаемых без доказательства), так что остальные утверждения выводятся из аксиом с помощью логических доказательств. Применение аксиоматического метода предполагает наличие развитого логического языка для формулировки утверждений, и, как правило, требует значительных усилий, времени и места для построения строгих формальных доказательств. Теория множеств не является исключением — для неё предложено несколько систем аксиом, весьма поучительных и оказавших значительное влияние на развитие математики в целом. Альтернативой аксиоматическому методу является апелляция к интуиции, здравому смыслу и использование неопределяемых понятий. Применительно к теории множеств такой подход получил название «наивной теории множеств», элементы которой излагаются в этом учебнике.

Примеры

Множество S страниц в данной книге. Множество $\mathbb{N}$ *натуральных* чисел $\{1, 2, 3, \ldots\}$. Множество $\mathbb{P}$ *простых* чисел $\{2, 3, 5, 7, 11, \ldots\}$. Множество $\mathbb{Z}$ целых чисел $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. Множество $\mathbb{R}$ *вещественных* чисел. Множество A различных символов на этой странице.

Если объект x является элементом множества M, то говорят, что x принадлежит M. Обозначение: $x \in M$. В противном случае говорят, что x *не принадлежит* M. Обозначение: $x \notin M$.

ЗАМЕЧАНИЕ

Обычно множества обозначают прописными буквами латинского алфавита, а элементы множеств — строчными буквами.

Множество, не содержащее элементов, называется *пустым*. Обозначение: $\varnothing$.

ЗАМЕЧАНИЕ

Введение в рассмотрение пустого множества является сильным допущением. Например, известно, что синих лошадей в природе не бывает. Тем не менее мы позволяем себе рассматривать «множество синих лошадей» как полноправный объект, вводить для него обозначения и т. д.

ОТСТУПЛЕНИЕ

Понятия множества, элемента и принадлежности, которые на первый взгляд представляются интуитивно ясными, при ближайшем рассмотрении такую ясность утрачивают. Во-первых, даже отличимость элементов при более глубоком рассмотрении представляет некоторую проблему. Например, символы «а» и «*а*», которые встречаются на этой странице, — это один элемент множества A или два разных элемента? Или же, например, два вхождения символа «о» в слово «множество» — графически они неотличимы

невооружённым глазом, но это символы букв, которые обозначают звуки, а читаются эти две гласные по-разному: первая под ударением, а вторая — безударная. Во-вторых, проблематична возможность (без дополнительных усилий) указать, принадлежит ли данный элемент данному множеству. Например, является ли число 869584769215374850678574 67 простым?

Множества как объекты могут быть элементами других множеств. Множество, элементами которого являются множества, иногда называют *семейством*.

ЗАМЕЧАНИЕ

Семейства множеств часто обозначают прописными «рукописными» буквами латинского алфавита.

Совокупность объектов, которая *не* является множеством, называется *классом*. Неформально говоря, называя совокупность элементов классом, а не множеством, мы берём на себя сравнительно мéньшую ответственность за определённость, отличимость и бесповторность элементов.

ОТСТУПЛЕНИЕ

Термин «класс» в объектно-ориентированном программировании (ООП) употребляется правомерно. Можно было бы сказать, что класс в ООП — это множество объектов, являющихся экземплярами класса. Однако такое определение не вполне корректно. Например, возможность динамического изменения класса объекта, существующая в некоторых языках ООП, ставит под сомнение статическую определённость класса, а возможность клонирования объектов ставит под сомнение отличимость экземпляров как элементов класса.

Обычно в конкретных рассуждениях элементы всех рассматриваемых множеств (и семейств) берутся из некоторого одного, достаточно широкого множества U (своего для каждого случая), которое называется *универсальным* множеством (или *универсумом*).

1.1.2. Задание множеств

Чтобы задать множество, нужно указать, какие элементы ему принадлежат. Это указание заключают в пару фигурных скобок, оно может иметь одну из следующих основных форм:

перечисление элементов: $M := \{a, b, c, \ldots, z\}$;
характеристический предикат: $M := \{x \mid P(x)\}$;
порождающая процедура: $M := \{x \mid x := f\}$.

При задании множеств перечислением обозначения элементов разделяют запятыми. *Характеристический предикат* — это некоторое условие, выраженное в форме логического утверждения или процедуры, возвращающей логическое

значение, и позволяющее проверить, принадлежит ли любой данный элемент множеству. Если для данного элемента условие выполнено, то он принадлежит определяемому множеству, в противном случае — не принадлежит. Порождающая процедура — это процедура, которая в процессе работы порождает объекты, являющиеся элементами определяемого множества.

Примеры

1. $M_9 := \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
2. $M_9 := \{n \mid n \in \mathbb{N} \ \&\ n < 10\}$.
3. $M_9 := \{n \mid n := 0;$ **for** i **from** 1 **to** 9 **do** $n := n + 1;$ **yield** n **end for**$\}$.

При задании множеств перечислением обозначения элементов иногда снабжают индексами и указывают множество, из которого берутся индексы. В частности, запись $\{a_i\}_{i=1}^{k}$ означает то же, что $\{a_1, \ldots, a_k\}$, а запись $\mathcal{M} = \{M_\alpha\}_{\alpha \in A}$ означает, что $\mathcal{M}$ является семейством, элементами которого являются множества M_α, причём индекс α «пробегает» множество A. Знак многоточия (...), который употребляется при задании множеств, является сокращённой формой записи, в которой порождающая процедура для множества индексов считается очевидной.

Пример $\{a_1, a_2, a_3, \ldots\} = \{a_i\}_{i=1}^{\infty}$.

ЗАМЕЧАНИЕ

Множество целых чисел в диапазоне от m до n в этой книге обозначается так: $m..n$. То есть

$$m..n \stackrel{\text{Def}}{=} \{k \in \mathbb{Z} \mid m \leqslant k \ \&\ k \leqslant n\} = \{k \in \mathbb{Z} \mid \textbf{for } k \textbf{ from } m \textbf{ to } n \textbf{ do yield } k \textbf{ end for}\}.$$

Перечислением элементов можно задавать только *конечные* множества. *Бесконечные* множества задаются характеристическим предикатом или порождающей процедурой.

Пример $\mathbb{N} \stackrel{\text{Def}}{=} \{n \mid n := 0;$ **while true do** $n := n + 1;$ **yield** n **end while**$\}$.

1.1.3. Парадокс Рассела

Возможность задания множеств характеристическим предикатом зависит от предиката. Использование некоторых предикатов для этой цели может приводить к противоречиям. Например, все рассмотренные в примерах множества не содержат себя в качестве элемента. Рассмотрим множество Y всех множеств, не содержащих себя в качестве элемента:

$$Y \stackrel{\text{Def}}{=} \{X \mid X \notin X\}.$$

Если множество Y существует, то мы должны иметь возможность ответить на следующий вопрос: $Y \in Y$? Пусть $Y \in Y$, тогда $Y \notin Y$. Пусть $Y \notin Y$, тогда $Y \in Y$. Получается неустранимое логическое противоречие, которое известно как *парадокс Рассела*[1]. Вот три способа избежать этого конкретного парадокса.

1. Ограничить используемые характеристические предикаты видом

$$P(x) = x \in A \ \& \ Q(x),$$

где A — известное, заведомо существующее множество (*универсум*). Обычно при этом используют обозначение $\{x \in A \mid Q(x)\}$. Для Y универсум не указан, а потому Y множеством не является.

2. Теория типов. Объекты имеют тип 0, множества элементов типа 0 имеют тип 1, множества элементов типа 0 и 1 — тип 2 и т. д. Y не имеет типа и множеством не является.

3. Явный запрет принадлежности множества самому себе: $X \in X$ — недопустимый предикат. При аксиоматическом построении теории множеств соответствующая аксиома называется *аксиомой регулярности*.

ОТСТУПЛЕНИЕ

Существование и анализ парадоксов в теории множеств способствовали развитию так называемого *конструктивизма* — направления в математике, в рамках которого рассматриваются только такие объекты, для которых известны процедуры (алгоритмы) их порождения. В конструктивной математике исключаются из рассмотрения те понятия и методы классической математики, которые не заданы алгоритмически.

Парадокса Рассела можно избежать, ограничив рассматриваемые множества. Например, достаточно запретить рассматривать в качестве множеств классы, содержащие самих себя. При этом, однако, нет полной уверенности в том, что не обнаружатся другие противоречия. Полноценным выходом из ситуации являлось бы аксиоматическое построение теории множеств и доказательство непротиворечивости построенной формальной теории. Однако исследование парадоксов и непротиворечивости систем аксиом является технически трудной задачей и уводит далеко в сторону от программистской практики, для которой важнейшими являются конечные множества. Поэтому формальная аксиоматика теории множеств здесь не приводится. Мы излагаем необходимые сведения полуформально, опираясь везде, где это возможно, на программистскую интуицию читателя.

1.1.4. Мультимножества

В множестве все элементы различны, а значит, входят в множество ровно один раз. В некоторых случаях оказывается полезным рассматривать совокупности элементов, в которые элементы входят по нескольку раз.

[1] Бертран Рассел (1872–1970).

Пример В штатном расписании одна и та же должность встречается несколько раз.

Пусть $X = \{x_1, \ldots, x_n\}$ — некоторое (конечное) множество и пусть $a_1, \ldots, a_n$ — неотрицательные целые числа. Тогда *мультимножеством* $\widehat{X}$ (над множеством X) называется совокупность элементов множества X, в которую элемент x_i входит a_i раз, $a_i \geqslant 0$. Мультимножество обозначается одним из следующих способов:

$$\widehat{X} = [x_1^{a_1}, \ldots, x_n^{a_n}] = \langle \underbrace{x_1, \ldots, x_1}_{a_1}; \ldots; \underbrace{x_n, \ldots, x_n}_{a_n} \rangle = \langle a_1(x_1), \ldots, a_n(x_n) \rangle.$$

Пример Пусть $X = \{a, b, c\}$. Тогда $\widehat{X} = [a^0 b^3 c^4] = \langle b, b, b; c, c, c, c \rangle = \langle 0(a), 3(b), 4(c) \rangle$.

ЗАМЕЧАНИЕ

Элементы мультимножества, равно как и элементы множества, считаются неупорядоченными.

Пусть $\widehat{X} = \langle a_1(x_1), \ldots, a_n(x_n) \rangle$ — мультимножество над множеством $X = \{x_1, \ldots, x_n\}$. Тогда число a_i называется *показателем* элемента x_i, множество X — *носителем* мультимножества $\widehat{X}$, число $m = a_1 + \ldots + a_n$ — *мощностью* мультимножества $\widehat{X}$, а множество $\underline{\widehat{X}} = \{x_i \in X \mid a_i > 0\}$ называется *составом* мультимножества $\widehat{X}$.

Пример Пусть $\widehat{X} = [a^0 b^3 c^4]$ — мультимножество над множеством $X = \{a, b, c\}$. Тогда $\underline{\widehat{X}} = \{b, c\}$.

Мультимножество $\widehat{X} = \langle a_1(x_1), \ldots, a_n(x_n) \rangle$ над множеством $X = \{x_1, \ldots, x_n\}$ называется *индикатором*, если $\forall\, i \in 1..n \;\; (a_i = 0 \vee a_i = 1)$.

Пример Мультимножество $\widehat{X} = \langle b; c \rangle$ является индикатором над множеством $X = \{a, b, c\}$, причём $\underline{\widehat{X}} = \{b, c\}$.

1.2. Алгебра подмножеств

Самого по себе понятия множества ещё недостаточно — нужно определить способы конструирования новых множеств из уже имеющихся, то есть определить операции над множествами.

1.2.1. Сравнение множеств

Множество A *содержится* в множестве B (множество B *включает* множество A), если каждый элемент множества A есть элемент множества B:

$$A \subset B \stackrel{\text{Def}}{=} x \in A \Longrightarrow x \in B.$$

В этом случае A называется *подмножеством* B, B — *надмножеством* A. По определению $\forall M\ (\varnothing \subset M)$.

ЗАМЕЧАНИЕ

Нетрудно видеть, что понятие подмножества множества X равнообъёмно понятию индикатора над множеством X. Действительно, если $X = \{x_1, \ldots, x_n\}$ — множество, а Y — любое подмножество множества X, $Y \subset X$, то существует единственный индикатор $\widehat{Y} = \langle a_1(x_1), \ldots, a_n(x_n)\rangle$ над множеством X, такой, что $\forall i \in 1..n\ (a_i = 1 \iff x_i \in Y)$.

Два множества *равны*, если они являются подмножествами друг друга:

$$A = B \stackrel{\text{Def}}{=} A \subset B\ \&\ B \subset A.$$

ТЕОРЕМА *Включение множеств обладает следующими свойствами:*

1. $\forall A\ (A \subset A)$.
2. $\forall A, B\ (A \subset B\ \&\ B \subset A \Longrightarrow A = B)$.
3. $\forall A, B, C\ (A \subset B\ \&\ B \subset C \Longrightarrow A \subset C)$.

Доказательство

[1] Имеем $x \in A \Longrightarrow x \in A$, и значит, $A \subset A$.

[2] По определению.

[3] Если $x \in A \Longrightarrow x \in B$ и $x \in B \Longrightarrow x \in C$, то $x \in A \Longrightarrow x \in C$, и значит, $A \subset C$. □

Если $A \subset B$ и $A \neq B$, то множество A называется *собственным* подмножеством множества B, а B — *собственным* надмножеством множества A.

ЗАМЕЧАНИЕ

Если требуется различать собственные и несобственные подмножества, то для обозначения включения собственных подмножеств используется знак $\subsetneq$, а для несобственных — $\subseteq$.

СЛЕДСТВИЕ $A \subsetneq B \subseteq C \Longrightarrow A \subsetneq C$.

1.2.2. Равномощные множества

Говорят, что между множествами A и B установлено *взаимно-однозначное соответствие*, если каждому элементу множества A поставлен в соответствие один и только один элемент множества B и для каждого элемента множества B один и только один элемент множества A поставлен в соответствие этому элементу множества B. В этом случае говорят также, что множества A и B *изоморфны*, и используют обозначение $A \sim B$. Если при заданном соответствии элементу $a \in A$ соответствует элемент $b \in B$, то данное обстоятельство обозначают следующим образом: $a \mapsto b$.

ЗАМЕЧАНИЕ

Весьма общее понятие соответствия в этом учебнике трактуется с программистских позиций. Если сказано, что между множествами A и B установлено соответствие, то подразумевается, что задан способ по любому элементу $a \in A$ определить соответствующий ему элемент $b \in B$. Способ может быть любым, если возможность его применения не вызывает сомнений. Например, это может быть массив **array** $[A]$ **of** B, или процедура типа **proc** $(A) : B$, или же выражение, зависящее от переменной типа A и доставляющее значение типа B.

Пример Соответствие $n \mapsto 2n$ устанавливает взаимно-однозначное соответствие между множеством натуральных чисел $\mathbb{N}$ и множеством чётных натуральных чисел $2\mathbb{N}$, $\mathbb{N} \sim 2\mathbb{N}$.

Нетрудно видеть, что:

1) любое множество взаимно-однозначно соответствует самому себе: $A \sim A$ — достаточно рассмотреть соответствие $a \mapsto a$, где $a \in A$;
2) если $A \sim B$, то $B \sim A$ — достаточно использовать соответствие $a \mapsto b$ для построения соответствия $b \mapsto a$, где $a \in A, b \in B$;
3) если $A \sim B$ и $B \sim C$, то $A \sim C$ — соответствие устанавливается с использованием промежуточного элемента $b \in B$: $a \mapsto b \mapsto c$, где $a \in A, b \in B, c \in C$.

Если между двумя множествами A и B может быть установлено взаимно-однозначное соответствие, то говорят, что множества имеют одинаковую *мощность*, или что множества *равномощны*, и записывают это так: $|A| = |B|$. Другими словами,

$$|A| = |B| \stackrel{\text{Def}}{\Leftrightarrow} A \sim B.$$

Из определения и отмеченных свойств взаимно-однозначного соответствия непосредственно следует

ТЕОРЕМА *Равномощность множеств обладает следующими свойствами:*

1. $\forall A\ (|A| = |A|)$.
2. $\forall A, B\ (|A| = |B| \Longrightarrow |B| = |A|)$.
3. $\forall A, B, C\ (|A| = |B|\ \&\ |B| = |C| \Longrightarrow |A| = |C|)$.

Примеры

1. Множество десятичных цифр равномощно множеству пальцев на руках человека (для подавляющего большинства людей), но не равномощно множеству пальцев на руках и на ногах.
2. Множество чётных натуральных чисел равномощно множеству всех натуральных чисел.

1.2.3. Конечные и бесконечные множества

Для приложений дискретной математики в программировании наибольшее значение имеют множества с конечным количеством элементов. Вопрос о том, чем конечное отличается от бесконечного, неподготовленного человека может поставить в тупик. Здесь мы рассматриваем применительно к множествам один из возможных ответов на этот вопрос. С античных времен известен принцип: «часть меньше целого». Оказывается, этот принцип можно использовать в качестве характеристического свойства конечных множеств. Для бесконечных множеств этот принцип не имеет места.

Множество A называется *конечным*, если у него нет равномощного собственного подмножества:

$$\forall B\ ((B \subset A\ \&\ |B| = |A|) \Longrightarrow (B = A)).$$

Для конечного множества A используется запись $|A| < \infty$. Все остальные множества называются *бесконечными*. Взяв отрицание условия конечности, получаем, что для бесконечного множества A

$$\exists B\ (B \subset A\ \&\ |B| = |A|\ \&\ B \neq A),$$

то есть бесконечное множество равномощно некоторому своему собственному подмножеству. Для бесконечного множества A используется запись $|A| = \infty$.

Пример Множество натуральных чисел бесконечно, $|\mathbb{N}| = \infty$, поскольку оно равномощно своему собственному подмножеству чётных чисел.

ОТСТУПЛЕНИЕ

В компьютере *все* множества реальных объектов (множество адресуемых ячеек памяти, множество исполнимых программ, множество тактов работы процессора) конечны.

ТЕОРЕМА *Множество, имеющее бесконечное подмножество, бесконечно:*

$$(B \subset A\ \&\ |B| = \infty) \Longrightarrow (|A| = \infty).$$

Доказательство Множество B бесконечно, то есть существует взаимно-однозначное соответствие $B \sim C$ между множеством B и некоторым его собственным подмножеством C. Обозначим это соответствие $x \mapsto x'$. Построим соответствие между множеством A и его собственным подмножеством D:

$$x \mapsto \textbf{if } x \in B \textbf{ then } x' \textbf{ else } x \textbf{ end if}.$$

Другими словами, на элементах из B мы пользуемся заданным соответствием, а остальным элементам сопоставляем их самих (рис. 1.1). Это взаимно-однозначное соответствие между множеством A и его собственным подмножеством D, и значит, $|A| = \infty$. □

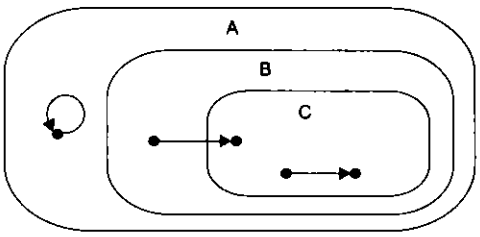

Рис. 1.1. К доказательству теоремы (к п. 1.2.3)

ЗАМЕЧАНИЕ

В обозначениях подраздела 1.2.6 $D = C \cup (A \setminus B)$.

СЛЕДСТВИЕ *Все подмножества конечного множества конечны.*

1.2.4. Добавление и удаление элементов

Если A — множество, а x — элемент, причём $x \notin A$, то элемент x можно *добавить* в множество A:

$$A + x \stackrel{\text{Def}}{=} \{y \mid y \in A \vee y = x\}.$$

Аналогично, если A — множество, а x — элемент, причём $x \in A$, то элемент x можно *удалить* из множества A:

$$A - x \stackrel{\text{Def}}{=} \{y \mid y \in A \;\&\; y \neq x\}.$$

Легко видеть, что при удалении и добавлении конечного числа элементов конечные множества остаются конечными, а бесконечные — бесконечными.

ОТСТУПЛЕНИЕ

На самом деле операции добавления и удаления элементов являются частными случаями операций объединения и разности множеств, рассматриваемых в подразделе 1.2.6 ($A + x = A \cup \{x\}, A - x = A \setminus \{x\}$), и потому, строго говоря, излишни. В стандартных учебниках по теории множеств такие операции не рассматриваются и не упоминаются. Здесь они введены для упрощения обозначений. Такой подход акцентирует «программистское» отношение к математике: мы вводим новые операции, если это практически удобно, даже если это теоретически излишне.

1.2.5. Мощность конечного множества

ТЕОРЕМА 1 *Любое непустое конечное множество равномощно некоторому отрезку натурального ряда:*

$$\forall A \; (A \neq \varnothing \;\&\; |A| < \infty \Longrightarrow \exists k \in \mathbb{N} \; (|A| = |1..k|)).$$

Доказательство Рассмотрим следующую программу:

$i := 0$ { счётчик элементов }
while $A \neq \varnothing$ **do**
  select $x \in A$ { выбираем элемент }
  $i := i + 1$ { увеличиваем счётчик }
  $x \mapsto i$ { ставим элементу в соответствие его номер }
  $A := A - x$ { удаляем элемент из множества }
end while

Если эта программа не заканчивает работу, то она даёт соответствие $B \sim \mathbb{N}$ для некоторого множества $B \subset A$, что невозможно ввиду конечности A. Значит, процедура заканчивает работу при $i = k$. Но в этом случае построено взаимно-однозначное соответствие $A \sim 1..k$. □

ЛЕММА *Любое непустое подмножество множества натуральных чисел содержит наименьший элемент.*

Доказательство Пусть A — произвольное подмножество множества натуральных чисел, конечное или бесконечное. Рассмотрим задание множества A следующей порождающей процедурой:

$A := \{n \in \mathbb{N} \mid n := 0;$ **while true do** $n := n + 1;$ **if** $n \in A$ **then yield** n **end if end while**$\}$.

Ясно, что множество A действительно порождается этой процедурой, причём тот элемент, который порождается первым, является наименьшим. □

ТЕОРЕМА 2 *Любой отрезок натурального ряда конечен:*

$$\forall n \in \mathbb{N}\ (|1..n| < \infty).$$

Доказательство От противного. Пусть существуют бесконечные отрезки натурального ряда. Рассмотрим наименьшее n такое, что $|1..n| = \infty$. Тогда отрезок $1..n$ равномощен некоторому своему собственному подмножеству A, $|1..n| = |A|$, то есть существует взаимно-однозначное соответствие между отрезком $1..n$ и подмножеством A. Пусть при этом соответствии $n \mapsto i$. Рассмотрим соответствие между отрезком $1..(n-1)$ и его собственным подмножеством $A - i$, задаваемое соответствием между $1..n$ и A. Это соответствие является взаимно-однозначным, а значит, отрезок $1..(n-1)$ изоморфен своему собственному подмножеству и является бесконечным, что противоречит выбору n. □

СЛЕДСТВИЕ *Различные отрезки натурального ряда неравномощны:*

$$n \neq m \Longrightarrow |1..n| \neq |1..m|.$$

Доказательство Пусть для определённости $n > m$. Тогда $1..m \subset 1..n$ и $1..m \neq 1..n$. Если $|1..n| = |1..m|$, то $|1..m| = \infty$, что противоречит теореме. □

Говорят, что конечное множество A имеет *мощность* k (обозначения: $|A| = k$, $\operatorname{card} A = k$, $\#A = k$), если оно равномощно отрезку $1..k$:

$$|A| = k \stackrel{\text{Def}}{=} A \sim 1..k.$$

ЗАМЕЧАНИЕ

Таким образом, если множество A конечно, $|A| = k$, то элементы A всегда можно *перенумеровать*, то есть поставить им в соответствие номера из отрезка $1..k$ с помощью некоторой процедуры. Наличие такой процедуры подразумевается, когда употребляется запись $A = \{a_1, \ldots, a_k\}$.

По определению $|\varnothing| \stackrel{\text{Def}}{=} 0$.

1.2.6. Операции над множествами

Обычно рассматриваются следующие операции над множествами:

объединение:

$$A \cup B \stackrel{\text{Def}}{=} \{x \mid x \in A \vee x \in B\};$$

пересечение:

$$A \cap B \stackrel{\text{Def}}{=} \{x \mid x \in A \;\&\; x \in B\};$$

разность:

$$A \setminus B \stackrel{\text{Def}}{=} \{x \mid x \in A \;\&\; x \notin B\};$$

симметрическая разность:

$$A \triangle B \stackrel{\text{Def}}{=} (A \cup B) \setminus (A \cap B) = \{x \mid (x \in A \;\&\; x \notin B) \vee (x \notin A \;\&\; x \in B)\};$$

дополнение:

$$\overline{A} \stackrel{\text{Def}}{=} \{x \mid x \notin A\}.$$

Операция дополнения подразумевает, что задан некоторый универсум U: $\overline{A} = U \setminus A$. В противном случае операция дополнения не определена.

Пример Пусть $A := \{1, 2, 3\}$, $B := \{3, 4, 5\}$. Тогда $A \cup B = \{1, 2, 3, 4, 5\}$, $A \cap B = \{3\}$, $A \setminus B = \{1, 2\}$, $A \triangle B = \{1, 2, 4, 5\}$. Если определён универсум $U := \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, то $\overline{A} = \{0, 4, 5, 6, 7, 8, 9\}$, $\overline{B} = \{0, 1, 2, 6, 7, 8, 9\}$.

На рис. 1.2 приведены *диаграммы Венна*[1], иллюстрирующие операции над множествами. Сами исходные множества изображаются фигурами (в данном случае овалами), а результат выделяется графически (в данном случае для выделения использована штриховка).

[1] Джон Венн (1834–1923).

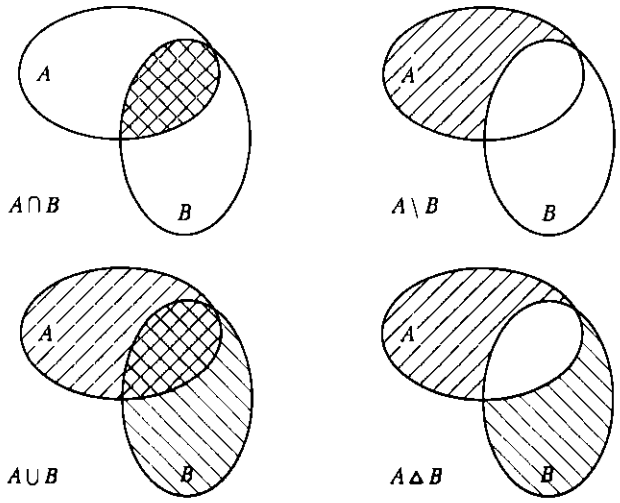

Рис. 1.2. Операции над множествами

Если множества A и B конечны, то из определений и рис. 1.2 нетрудно видеть, что

$$|A \cup B| = |A| + |B| - |A \cap B|,$$
$$|A \setminus B| = |A| - |A \cap B|,$$
$$|A \triangle B| = |A| + |B| - 2|A \cap B|.$$

Операции пересечения и объединения двух множеств допускают следующее обобщение. Пусть задано семейство множеств $\{A_i\}_{i \in I}$. Тогда

$$\bigcup_{i \in I} A_i \stackrel{\text{Def}}{=} \{x \mid \exists i \in I \ (x \in A_i)\}, \qquad \bigcap_{i \in I} A_i \stackrel{\text{Def}}{=} \{x \mid \forall i \in I \ (x \in A_i)\}.$$

1.2.7. Разбиения и покрытия

Пусть $\mathcal{E} = \{E_i\}_{i \in I}$ — некоторое семейство подмножеств множества M, $E_i \subset M$. Семейство $\mathcal{E}$ называется *покрытием* множества M, если каждый элемент M принадлежит хотя бы одному из E_i:

$$\forall x \in M \ (\exists i \in I \ (x \in E_i)).$$

Семейство $\mathcal{E}$ называется *дизъюнктным*, если элементы этого семейства попарно не пересекаются, то есть каждый элемент множества M принадлежит не более чем одному из множеств E_i:

$$\forall i, j \in I \ (i \neq j \Longrightarrow E_i \cap E_j = \varnothing).$$

Дизъюнктное покрытие называется *разбиением* множества M. Элементы разбиения, то есть подмножества множества M, часто называют *блоками* разбиения.

Пример Пусть $M := \{1,2,3\}$. Тогда $\{\{1,2\},\{2,3\},\{3,1\}\}$ является покрытием, но не разбиением; $\{\{1\},\{2\},\{3\}\}$ является разбиением (и покрытием), а семейство $\{\{1\},\{2\}\}$ является дизъюнктным, но не является ни покрытием, ни разбиением.

ТЕОРЕМА *Если $\mathcal{E} = \{E_i\}_{i\in I}$ есть дизъюнктное семейство подмножеств множества M, то существует разбиение $\mathcal{B} = \{B_i\}_{i\in I}$ множества M такое, что каждый элемент дизъюнктного семейства $\mathcal{E}$ является подмножеством блока разбиения $\mathcal{B}$:*

$$\forall i \in I \ (E_i \subset B_i).$$

Доказательство Выберем произвольный элемент $i_0 \in I$ и построим семейство $\mathcal{B} = \{B_i\}_{i\in I}$ следующим образом:

$$B_{i_0} = M \setminus \bigcup_{i\in I - i_0} E_i, \qquad \forall i \in I - i_0 \ (B_i = E_i).$$

Семейство $\mathcal{B}$ по построению является дизъюнктным покрытием, то есть разбиением. Ясно, что $E_{i_0} \subset B_{i_0}$, а для остальных элементов требуемое включение имеет место по построению. □

Пример Пусть $M := \{1,2,3\}$. Тогда элементы дизъюнктного семейства $\{\{1\},\{2\}\}$ являются подмножествами блоков разбиения $\{\{1\},\{2,3\}\}$.

1.2.8. Булеан

Множество всех подмножеств множества M называется *булеаном* множества M и обозначается 2^M:

$$2^M \stackrel{\text{Def}}{=} \{A \mid A \subset M\}.$$

ТЕОРЕМА *Если множество M конечно, то $|2^M| = 2^{|M|}$.*

Доказательство Индукция по $|M|$. База: если $|M| = 0$, то $M = \varnothing$ и $2^{\varnothing} = \{\varnothing\}$. Следовательно, $|2^{\varnothing}| = |\{\varnothing\}| = 1 = 2^0 = 2^{|\varnothing|}$. Индукционный переход: пусть $\forall M \ (|M| < k \Longrightarrow |2^M| = 2^{|M|})$. Рассмотрим $M = \{a_1,\ldots,a_k\}$, $|M| = k$. Положим

$$\mathcal{M}_1 := \{X \subset 2^M \mid a_k \notin X\} \text{ и } \mathcal{M}_2 := \{X \subset 2^M \mid a_k \in X\}.$$

Имеем $2^M = \mathcal{M}_1 \cup \mathcal{M}_2$, $\mathcal{M}_1 \cap \mathcal{M}_2 = \varnothing$, $\mathcal{M}_1 \sim \mathcal{M}_2$ за счёт взаимно-однозначного соответствия $X \mapsto X + a_k$ и $\mathcal{M}_1 \sim 2^{\{a_1,\ldots,a_{k-1}\}}$. По индукционному предположению $|2^{\{a_1,\ldots,a_{k-1}\}}| = 2^{k-1}$, и значит, $|\mathcal{M}_1| = |\mathcal{M}_2| = 2^{k-1}$. Следовательно, $|2^M| = |\mathcal{M}_1| + |\mathcal{M}_2| = 2^{k-1} + 2^{k-1} = 2^k = 2^{|M|}$. □

Пересечение, объединение и разность подмножеств множества U (универсума) являются его подмножествами, то есть применение операций к подмножествам универсума не выводит за его пределы. Множество всех подмножеств множества U с операциями пересечения, объединения, разности и дополнения образует *алгебру подмножеств* множества U.

1.2.9. Свойства операций над множествами

Операции над множествами обладают целым рядом важных свойств, которые рассматриваются в этом подразделе. Пусть задан универсум U. Тогда $\forall A, B, C \subset U$ и выполняются следующие равенства:

1) *идемпотентность*:
   $A \cup A = A, \quad A \cap A = A$;
2) *коммутативность*:
   $A \cup B = B \cup A, \quad A \cap B = B \cap A$;
3) *ассоциативность*:
   $A \cup (B \cup C) = (A \cup B) \cup C, \quad A \cap (B \cap C) = (A \cap B) \cap C$;
4) *дистрибутивность*:
   $A \cup (B \cap C) = (A \cup B) \cap (A \cup C), \quad A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
5) *поглощение*:
   $(A \cap B) \cup A = A, \quad (A \cup B) \cap A = A$;
6) свойства нуля:
   $A \cup \varnothing = A, \quad A \cap \varnothing = \varnothing$;
7) свойства единицы:
   $A \cup U = U, \quad A \cap U = A$;
8) *инволютивность*:
   $\overline{\overline{A}} = A$;
9) *законы де Моргана*[1]:
   $\overline{A \cap B} = \overline{A} \cup \overline{B}, \quad \overline{A \cup B} = \overline{A} \cap \overline{B}$;
10) свойства дополнения:
    $A \cup \overline{A} = U, \quad A \cap \overline{A} = \varnothing$;
11) выражение для разности:
    $A \setminus B = A \cap \overline{B}$.

В справедливости перечисленных равенств можно убедиться различными способами. Например, можно нарисовать диаграммы Венна для левой и правой частей равенства и убедиться, что они совпадают, или же провести формальное рассуждение для каждого равенства. Рассмотрим для примера первое равенство: $A \cup A = A$. Возьмем произвольный элемент x, принадлежащий левой части равенства, $x \in A \cup A$. По определению операции объединения $\cup$ имеем $x \in A \vee x \in A$. В любом случае $x \in A$. Взяв произвольный элемент из множества в левой части равенства, обнаружим, что он принадлежит множеству в правой части. Отсюда по определению включения множеств получаем, что $A \cup A \subset A$. Пусть теперь $x \in A$. Тогда, очевидно, верно $x \in A \vee x \in A$. Отсюда по определению операции объединения имеем $x \in A \cup A$. Таким образом, $A \subset A \cup A$. Следовательно, по определению равенства множеств, $A \cup A = A$. Это рассуждение можно записать короче:

$$x \in A \cup A \Longleftrightarrow x \in A \vee x \in A \Longleftrightarrow x \in A.$$

Аналогичные рассуждения нетрудно провести и для остальных равенств.

[1] Огастес Морган (1806–1871).

1.3. Представление множеств в программах

Термин «представление» применительно к программированию означает следующее. Представить в программе какой-либо объект (в данном случае множество) — это значит описать в терминах системы программирования структуру данных, используемую для хранения информации о представляемом объекте, и алгоритмы над выбранными структурами данных, которые реализуют присущие данному объекту операции. Таким образом, применительно к множествам определение представления подразумевает описание способа хранения информации о принадлежности элементов множеству и описание алгоритмов для вычисления объединения, пересечения и других введённых операций. Следует подчеркнуть, что, как правило, один и тот же объект может быть представлен многими разными способами, причём нельзя указать способ, который является наилучшим для всех возможных случаев. В одних случаях выгодно использовать одно представление, а в других — другое. Выбор представления зависит от целого ряда факторов: особенностей представляемого объекта, состава и относительной частоты использования операций в конкретной задаче и т. д. Умение выбрать наилучшее для данного случая представление является основой искусства практического программирования. Хороший программист отличается тем, что он знает *много* разных способов представления и умело выбирает наиболее подходящий.

1.3.1. Битовые шкалы

Пусть задан конечный универсум U и число элементов в нем не превосходит разрядности компьютера, $|U| \leqslant n$. Элементы универсума нумеруются:

$$U = \{u_1, \ldots, u_n\}.$$

Подмножество A универсума U представляется *кодом* (*машинным словом* или *битовой шкалой*) C : **array** $[1..n]$ **of** 0..1, в котором:

$$C[i] = \begin{cases} 1, & \text{если } u_i \in A, \\ 0, & \text{если } u_i \notin A. \end{cases}$$

ЗАМЕЧАНИЕ

Указанное представление можно описать короче, задав *инвариант* (условие, которому удовлетворяют все элементы): $\forall i \in 1..n \ (C[i] = u_i \in A)$. В дальнейшем предпочтение отдаётся более короткой форме записи описания представлений.

Код пересечения множеств A и B есть поразрядное логическое произведение кода множества A и кода множества B. Код объединения множеств A и B есть поразрядная логическая сумма кода множества A и кода множества B. Код дополнения множества A есть инверсия кода множества A. В большинстве компьютеров для этих операций есть соответствующие машинные команды. Таким образом, операции над небольшими множествами выполняются весьма эффективно. В некоторых языках программирования, например в Паскале, это представление множеств непосредственно включено в состав типов данных языка.

ЗАМЕЧАНИЕ

Если мощность универсума превосходит размер машинного слова, но не очень велика, то для представления множеств используются массивы битовых шкал. В этом случае операции над множествами реализуются с помощью циклов по элементам массива.

Эту же идею можно использовать для представления мультимножеств. Если $\widehat{X} = [x_1^{a_1}, \ldots, x_n^{a_n}]$ — мультимножество над над множеством $X = \{x_1, \ldots, x_n\}$, то массив A : **array** $[1..n]$ **of integer**, где $\forall i \in 1..n \ (A[i] = a_i)$, является представлением мультимножества $\widehat{X}$.

ЗАМЕЧАНИЕ

Представление индикатора совпадает с представлением его состава. Полезно сравнить это наблюдение с первым замечанием в подразделе 1.2.1.

1.3.2. Генерация всех подмножеств универсума

Во многих переборных алгоритмах требуется последовательно рассмотреть все подмножества заданного множества $\{a_1, \ldots, a_k\}$. В большинстве компьютеров целые числа представляются кодами в двоичной системе счисления, причём число $2^k - 1$ представляется кодом, содержащим k единиц, а все числа, меньшие $2^k - 1$, представляются кодами, имеющими не более k ненулевых разрядов. Таким образом, код числа 0 является представлением пустого множества $\varnothing$, код числа 1 является представлением подмножества, состоящего из первого элемента, и т. д., код числа $2^k - 1$ является представлением всего множества $\{a_1, \ldots, a_k\}$. Это наблюдение позволяет сформулировать следующий тривиальный алгоритм, который перечисляет все подмножества n-элементного множества.

Алгоритм 1.1 Генерация всех подмножеств n-элементного множества

Вход: $n \geqslant 0$ — мощность множества.
Выход: последовательность кодов подмножеств i.

```
for i from 0 to 2^n − 1 do
    yield i {код очередного подмножества}
end for
```

Обоснование Алгоритм выдает 2^n различных целых чисел, следовательно, 2^n различных кодов от 00...0 до 11...1. Таким образом, все подмножества генерируются, причём ровно по одному разу. □

Недостаток этого алгоритма состоит в том, что порядок генерации подмножеств никак не связан с их составом. Например, вслед за подмножеством с кодом 0111 (состоящим из трёх элементов) будет перечислено подмножество с кодом 1000 (состоящее из одного элемента).

ОТСТУПЛЕНИЕ

Во многих переборных задачах требуется рассмотреть все подмножества некоторого множества и найти среди них то, которое удовлетворяет заданному условию. При этом проверка условия часто может быть весьма трудоёмкой и зависеть от состава элементов очередного рассматриваемого подмножества. В частности, если очередное рассматриваемое подмножество незначительно отличается по составу элементов от предыдущего, то иногда можно воспользоваться результатами оценки элементов, которые рассматривались на предыдущем шаге перебора. В таком случае, если перебирать подмножества в подходящем порядке, можно значительно ускорить работу переборного алгоритма.

1.3.3. Алгоритм построения бинарного кода Грея

Алгоритм 1.2 генерирует последовательность всех подмножеств n-элементного множества, причём каждое следующее подмножество получается из предыдущего удалением или добавлением в точности одного элемента.

Алгоритм 1.2 Построение бинарного кода Грея

Вход: $n \geqslant 0$ — мощность множества.
Выход: последовательность кодов подмножеств B.

```
B : array [1..n] of 0..1 { битовая шкала для представления подмножеств }
for i from 1 to n do
  B[i] := 0 { инициализация }
end for
yield B { пустое множество }
for i from 1 to 2^n − 1 do
  p := Q(i) { определение номера элемента, подлежащего добавлению или
  удалению }
  B[p] := 1 − B[p] { добавление или удаление элемента }
  yield B { очередное подмножество }
end for
```

Функция Q, с помощью которой определяется номер разряда, подлежащего изменению, возвращает количество нулей на конце двоичной записи числа i, увеличенное на 1.

Алгоритм 1.3 Функция Q определения номера изменяемого разряда

Вход: i — номер подмножества.
Выход: номер изменяемого разряда.

```
q := 1; j := i
while j чётно do
  j := j/2; q := q + 1
end while
return q
```

Обоснование Для $n = 1$ искомая последовательность кодов есть $0, 1$. Пусть известна искомая последовательность кодов $B_1, \ldots, B_{2^k}$ для $n = k$. Тогда последовательность кодов $B_1 0, \ldots, B_{2^k} 0, B_{2^k} 1, \ldots, B_1 1$ будет искомой последовательностью для $n = k + 1$. Действительно, в последовательности $B_1 0, \ldots, B_{2^k} 0, B_{2^k} 1, \ldots, B_1 1$ имеется 2^{k+1} кодов, они все различны и соседние различаются ровно в одном разряде по построению. Именно такое построение и осуществляет данный алгоритм. На нулевом шаге алгоритм выдаёт правильное подмножество B (пустое). Пусть за первые $2^k - 1$ шагов алгоритм выдал последовательность значений B. При этом $B[k+1] = B[k+2] = \cdots = B[n] = 0$. На 2^k-м шаге разряд $B[k+1]$ изменяет своё значение с 0 на 1. После этого будет повторена последовательность изменений значений $B[1..k]$ в обратном порядке, поскольку $Q(2^k + m) = Q(2^k - m)$ для $0 \leqslant m \leqslant 2^k - 1$ (упражнение 1.3). □

Пример Протокол выполнения алгоритма 1.2 для $n = 3$.

i	p	B		
		0	0	0
1	1	0	0	1
2	2	0	1	1
3	1	0	1	0
4	3	1	1	0
5	1	1	1	1
6	2	1	0	1
7	1	1	0	0

1.3.4. Представление множеств упорядоченными списками

Если универсум очень велик (или бесконечен), а рассматриваемые подмножества универсума не очень велики, то представление с помощью битовых шкал не является эффективным с точки зрения экономии памяти. В этом случае множества обычно представляются списками элементов. Элемент списка при этом представляется записью с двумя полями: информационным и указателем на следующий элемент. Весь список задаётся указателем на первый элемент.

```
elem = record
  i: info; { информационное поле; тип info считается определённым}
  n: ↑ elem { указатель на следующий элемент }
end record
```

При таком представлении трудоёмкость операции $\in$ составит $O(n)$, а трудоёмкость операций $\subset$, $\cap$, $\cup$ составит $O(nm)$, где n и m — мощности участвующих в операции множеств.

ЗАМЕЧАНИЕ

Используемый здесь и далее символ O в выражениях вида $f(n) = O(g(n))$ читается «О большое» и означает, что функция f асимптотически ограничена по сравнению с g (ещё говорят, что f «имеет тот же порядок», что и g):

$$f(n) = O(g(n)) \stackrel{\text{Def}}{=} \exists n_0, c > 0 \ (\forall n > n_0 \ (f(n) \leqslant cg(n))) .$$

Если элементы в списках упорядочить, например, по возрастанию значения поля i, то трудоёмкость всех операций составит $O(n)$. Эффективная реализация операций над множествами, представленными в виде упорядоченных списков, основана на весьма общем алгоритме, известном как алгоритм *слияния*. Алгоритм слияния параллельно просматривает два множества, представленных упорядоченными списками, причём на каждом шаге продвижение происходит в том множестве, в котором текущий элемент меньше.

1.3.5. Проверка включения слиянием

Рассмотрим алгоритм слияния, который определяет, является ли множество A подмножеством множества B.

Алгоритм 1.4 Проверка включения слиянием

Вход: проверяемые множества A и B, которые заданы указателями a и b.
Выход: **true**, если $A \subset B$, в противном случае **false**.

```
pa := a; pb := b { инициализация }
while pa ≠ nil & pb ≠ nil do
  if pa.i < pb.i then
    return false { элемент множества A отсутствует в множестве B }
  else if pa.i > pb.i then
    pb := pb.n { элемент множества A, может быть, присутствует в
    множестве B }
  else
    pa := pa.n { здесь pa.i = pb.i, то есть }
    pb := pb.n { элемент множества A точно присутствует в множестве B }
  end if
end while
return pa = nil { true, если A исчерпано, false — в противном случае }
```

Обоснование На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A заведомо меньше, чем текущий и все последующие элементы множества B, а потому он не содержится в множестве B и можно завершить выполнение алгоритма. Во втором случае происходит продвижение по множеству B в надежде отыскать элемент, совпадающий с текущим элементом множества A. В третьем случае найдены

совпадающие элементы и происходит продвижение сразу в обоих множествах. По завершении основного цикла возможны два варианта: либо pa = **nil**, либо $pa \neq$ **nil**. Первый означает, что для всех элементов множества A удалось найти совпадающие элементы в множестве B. Второй — что множество B закончилось раньше, то есть не для всех элементов множества A удалось найти совпадающие элементы в множестве B. □

1.3.6. Вычисление объединения слиянием

Рассмотрим алгоритм слияния, который вычисляет объединение двух множеств, представленных упорядоченными списками.

Алгоритм 1.5 Вычисление объединения слиянием

Вход: объединяемые множества A и B, которые заданы указателями a и b.
Выход: объединение $C = A \cup B$, заданное указателем c.

```
pa := a; pb := b; c := nil; e := nil { инициализация }
while pa ≠nil & pb ≠nil do
  if pa.i < pb.i then
    d := pa.i; pa := pa.n { добавлению подлежит элемент множества A }
  else if pa.i > pb.i then
    d := pb.i; pb := pb.n { добавлению подлежит элемент множества B }
  else
    d := pa.i { здесь pa.i = pb.i, и можно взять любой из элементов }
    pa := pa.n; pb := pb.n { продвижение в обоих множествах }
  end if
  Append(c, e, d) { добавление элемента d в конец списка c }
end while
p := nil { указатель «хвоста» }
if pa ≠nil then
  p := pa { нужно добавить в результат оставшиеся элементы множества A }
end if
if pb ≠nil then
  p := pb { нужно добавить в результат оставшиеся элементы множества B }
end if
while p ≠nil do
  Append(c, e, p.i) { добавление элемента pa.i в конец списка c }
  p := p.n { продвижение указателя «хвоста» }
end while
```

Обоснование На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае в результирующий список добавляется текущий элемент множества A и происходит продвижение в этом множестве, во

втором — аналогичная операция производится с множеством B, а в третьем случае найдены совпадающие элементы и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все элементы обоих множеств, причём совпадающие элементы попадают ровно один раз. По завершении основного цикла один из указателей, pa и pb (но не оба вместе!), может быть не равен **nil**. В этом случае остаток соответствующего множества без проверки добавляется в результат. □

Вспомогательная процедура Append(c, e, d) присоединяет элемент d к хвосту e списка c.

Алгоритм 1.6 Процедура Append — присоединение элемента в конец списка

Вход: указатель c на первый элемент списка, указатель e на последний элемент списка, добавляемый элемент d.

Выход: указатель c на первый элемент списка, указатель e на последний элемент списка.

```
q := new(elem); q.i := d; q.n :=nil { новый элемент списка }
if c =nil then
  c := q { пустой список }
else
  e.n := q { непустой список }
end if
e := q
```

Обоснование До первого вызова функции Append имеем: $c =$ **nil** и $e =$ **nil**. После первого вызова указатель c указывает на первый элемент списка, а указатель e — на последний элемент (который после первого вызова является тем же самым элементом). Если указатели c и e не пусты и указывают на первый и последний элементы правильного списка, то после очередного вызова функции Append это свойство сохраняется. Из текста основной программы видно, что, кроме инициализации, все остальные манипуляции с этими указателями выполняются только с помощью функции Append. □

1.3.7. Вычисление пересечения слиянием

Рассмотрим алгоритм слияния, который вычисляет пересечение двух множеств, представленных упорядоченными списками.

Алгоритм 1.7 Вычисление пересечения слиянием

Вход: пересекаемые множества A и B, заданные указателями a и b.

Выход: пересечение $C = A \cap B$, заданное указателем c.

```
pa := a;  pb := b;  c :=nil;  e :=nil { инициализация }
while pa ≠nil & pb ≠nil do
```

```
  if pa.i < pb.i then
    pa := pa.n { элемент множества A не принадлежит пересечению }
  else if pa.i > pb.i then
    pb := pb.n { элемент множества B не принадлежит пересечению }
  else
    { здесь pa.i = pb.i — данный элемент принадлежит пересечению }
    Append(c, e, pa.i); pa := pa.n; pb := pb.n { добавление элемента }
  end if
end while
```

Обоснование На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A не принадлежит пересечению, он пропускается и происходит продвижение в этом множестве, во втором то же самое производится с множеством B. В третьем случае найдены совпадающие элементы, один экземпляр элемента добавляется в результат и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все совпадающие элементы обоих множеств, причём ровно один раз. □

1.3.8. Представление множеств итераторами

В предыдущих подразделах рассмотрены представления, в которых структуры данных предполагаются известными и используются при программировании операций. Иногда это бывает неудобно, поскольку, во-первых, возникают затруднения при совместном использовании нескольких альтернативных представлений для одного типа объектов, и, во-вторых, необходимо заранее определять набор операций, которым открыт доступ к внутренней структуре данных объектов. (Такие операции часто называют *первичными*.)

ОТСТУПЛЕНИЕ

Парадигма объектно-ориентированного программирования предполагает совместное согласованное определение структур данных и процедур доступа к ним (типов как множеств значений и первичных операций над значениями из этих множеств). Мы хотим напомнить читателю, что парадигма объектно-ориентированного программирования — весьма полезная, но отнюдь не универсальная и не единственная парадигма программирования.

Применительно к множествам следует признать, что понятие принадлежности является первичным, и потому вряд ли может быть запрограммировано без использования структуры данных для хранения множеств, но остальные операции над множествами нельзя признать первичными, и их можно запрограммировать независимо от представления множеств. Для иллюстрации этой идеи рассмотрим один из возможных способов реализации операций над множествами,

не зависящий от представления множеств. Видимо, можно считать, что с программистской точки зрения для представления множества достаточно *итератора* — процедуры, которая перебирает элементы множества и делает с ними то, что нужно. Синтаксически понятие итератора может быть реализовано самыми различными способами в зависимости от традиций, стиля и используемого языка программирования. Например, следующая программа выполняет процедуру S для всех элементов множества X (см. также доказательство теоремы 1 в подразделе 1.2.5):

while $X \neq \varnothing$ **do**
  select $x \in X$ { выбор произвольного элемента x из множества X }
  $S(x)$ { применение процедуры S к элементу x }
  $X := X - x$ {удаление элемента x с целью исключить его повторный выбор}
end while

В этом варианте реализации итератора исходное множество «исчезает», что не всегда желательно. Очевидно, что для всякого конкретного представления множества можно придумать такую реализацию итератора, которая обеспечивает перебор элементов ровно по одному разу и не портит исходного множества. Мы полагаем (здесь и далее во всей книге), что наличие итератора позволяет написать цикл

for $x \in X$ **do**
  $S(x)$ { где S — любой оператор или процедура, зависящие от x }
end for

и этот цикл означает применение оператора S ко всем элементам множества X по одному разу в некотором неопределённом порядке.

ЗАМЕЧАНИЕ

В доказательствах цикл **for** $x \in X$ **do** $S(x)$ **end for** используется и в том случае, когда $|X| = \infty$. Это не означает реального процесса исполнения тела цикла в дискретные моменты времени, а означает всего лишь мысленную возможность применить оператор S ко всем элементам множества X независимо от его мощности.

В таких обозначениях итератор пересечения множеств $X \cap Y$ может быть задан алгоритмом 1.8.

Алгоритм 1.8 Итератор пересечения множеств

```
for x ∈ X do
  for y ∈ Y do
    if x = y then
      S(x) { тело цикла }
      next for x { следующий x }
    end if
  end for
end for
```

Аналогично итератор разности множеств $X \setminus Y$ может быть задан алгоритмом 1.9.

Алгоритм 1.9 Итератор разности множеств

```
for x ∈ X do
  for y ∈ Y do
    if x = y then
      next for x { следующий x }
    end if
  end for
  S(x) { тело цикла }
end for
```

ЗАМЕЧАНИЕ

Оператор **next for** x — это оператор *структурного перехода*, выполнение которого в данном случае означает прерывание выполнения цикла по y и переход к следующему шагу в цикле по x.

Заметим, что $A \cup B = (A \setminus B) \cup B$ и $(A \setminus B) \cap B = \varnothing$. Поэтому достаточно рассмотреть итератор объединения дизъюнктных множеств $X \cup Y$ (алгоритм 1.10), где $X \cap Y = \varnothing$.

Алгоритм 1.10 Итератор объединения дизъюнктных множеств

```
for x ∈ X do
  S(x) { тело цикла }
end for
for y ∈ Y do
  S(y) { тело цикла }
end for
```

Стоит обратить внимание на то, что итератор пересечения реализуется вложенными циклами, а итератор дизъюнктного объединения — последовательностью циклов.

Нетрудно видеть, что сложность алгоритмов для операций с множествами при использовании предложенного представления, не зависящего от реализации, составляет в худшем случае $O(nm)$, где m и n — мощности участвующих множеств, в то время как для представления упорядоченными списками сложность в худшем случае равна $O(n + m)$.

ЗАМЕЧАНИЕ

На практике почти всегда оказывается так, что самая изощрённая реализация операции, не зависящая от представления, оказывается менее эффективной по сравнению с самой прямолинейной реализацией, ориентированной на конкретное представление.

ОТСТУПЛЕНИЕ

При современном состоянии аппаратной базы вычислительной техники оказывается, что во многих задачах можно пренебречь некоторым выигрышем в эффективности, например, различием между $O(nm)$ и $O(n+m)$. Другими словами, нынешние компьютеры настолько быстры, что часто можно не гнаться за самым эффективным представлением, ограничившись «достаточно эффективным».

1.4. Отношения

Обычное широко используемое понятие «отношение» имеет вполне точное математическое значение, которое вводится в этом разделе.

1.4.1. Упорядоченные пары и наборы

Если a и b — объекты, то через (a,b) обозначим *упорядоченную пару*. *Равенство* упорядоченных пар определяется следующим образом:

$$(a,b)=(c,d) \stackrel{\text{Def}}{=} a=c \;\&\; b=d.$$

Вообще говоря, $(a,b)\neq(b,a)$.

ЗАМЕЧАНИЕ

Формально упорядоченные пары можно определить как множества, если определить их так: $(a,b) \stackrel{\text{Def}}{=} \{a,\{a,b\}\}$. Таким образом, введённое понятие упорядоченной пары не выводит рассмотрение за пределы теории множеств.

Аналогично упорядоченным парам можно рассматривать упорядоченные тройки, четвёрки и т. д. В общем случае подобные объекты называются n-ками, *кортежами*, *наборами* или (конечными) *последовательностями*. Упорядоченный набор из n элементов обозначается $(a_1,\ldots,a_n)$. Набор $(a_1,\ldots,a_n)$ можно определить рекурсивно, используя понятие упорядоченной пары:

$$(a_1,\ldots,a_n) \stackrel{\text{Def}}{=} ((a_1,\ldots,a_{n-1}),a_n).$$

Количество элементов в наборе называется его *длиной* и обозначается следующим образом: $|(a_1,\ldots,a_n)|$.

ТЕОРЕМА *Два набора одной длины равны, если равны их соответствующие элементы:*

$$\forall n \left((a_1,\ldots,a_n)=(b_1,\ldots,b_n) \Longleftrightarrow \bigwedge_{i=1}^{n} a_i=b_i \right).$$

Доказательство Индукция по n. База: при $n = 2$ по определению равенства упорядоченных пар. Пусть теперь

$$(a_1, \ldots, a_{n-1}) = (b_1, \ldots, b_{n-1}) \Longleftrightarrow \bigwedge_{i=1}^{n-1} a_i = b_i.$$

Тогда

$$\begin{aligned}(a_1, \ldots, a_n) = (b_1, \ldots, b_n) &\iff ((a_1, \ldots, a_{n-1}), a_n) = ((b_1, \ldots, b_{n-1}), b_n) \\ &\iff (a_1, \ldots, a_{n-1}) = (b_1, \ldots, b_{n-1}) \ \& \ a_n = b_n \\ &\iff \bigwedge_{i=1}^{n-1} a_i = b_i. \end{aligned}$$

□

Отсюда следует, что порядок «отщепления» элементов в рекурсивном определении кортежа не имеет значения.

ЗАМЕЧАНИЕ

Наиболее естественным представлением в программе n-ки с элементами из множества A является массив **array** $[1..n]$ **of** A.

1.4.2. Прямое произведение множеств

Пусть A и B — два множества. *Прямым (декартовым) произведением* двух множеств A и B называется множество всех упорядоченных пар, в которых первый элемент принадлежит A, а второй принадлежит B:

$$A \times B \stackrel{\text{Def}}{=} \{(a, b) \mid a \in A \ \& \ b \in B\}.$$

ЗАМЕЧАНИЕ

Точка на плоскости может быть задана упорядоченной парой координат, то есть двумя точками на координатных осях. Таким образом, $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$. Своим появлением метод координат обязан Декарту[1], отсюда и название «декартово произведение».

ТЕОРЕМА *Для конечных множеств A и B $|A \times B| = |A|\,|B|$.*

Доказательство Первый компонент упорядоченной пары можно выбрать $|A|$ способами, второй — $|B|$ способами, причём независимо от выбора первого элемента. Таким образом, всего имеется $|A||B|$ различных упорядоченных пар. □

[1] Рене Декарт (1596–1650).

ЛЕММА $(A \times B) \times C \sim A \times (B \times C)$.

Доказательство

$$(A \times B) \times C = \{((a,b)c) \mid (a,b) \in A \times B \;\&\; c \in C\} =$$
$$= \{(a,b,c) \mid a \in A \;\&\; b \in B \;\&\; c \in C\} \overset{(a,b,c) \mapsto (a,(b,c))}{\sim}$$
$$\sim \{(a,(b,c)) \mid a \in A \;\&\; (b,c) \in B \times C\} = A \times (B \times C). \qquad \square$$

Понятие прямого произведения допускает обобщение. Прямое произведение множеств $A_1, \ldots, A_n$ — это множество наборов (*кортежей*):

$$A_1 \times \cdots \times A_n \overset{\text{Def}}{=} \{(a_1, \ldots, a_n) \mid a_1 \in A_1 \;\&\; \ldots \;\&\; a_n \in A_n\}.$$

Множества A_i не обязательно различны.

Степенью множества A называется его n-кратное прямое произведение самого на себя. Обозначение:

$$A^n \overset{\text{Def}}{=} \underbrace{A \times \cdots \times A}_{n \text{ раз}}.$$

Соответственно, $A^1 \overset{\text{Def}}{=} A$, $A^2 \overset{\text{Def}}{=} A \times A$ и вообще $A^n \overset{\text{Def}}{=} A \times A^{n-1}$.

СЛЕДСТВИЕ $|A^n| = |A|^n$.

1.4.3. Бинарные отношения

Пусть A и B — два множества. *Бинарным отношением* между множествами A и B называется тройка $\langle A, B, R \rangle$, где R — подмножество прямого произведения A и B:

$$R \subset A \times B.$$

R называется *графиком* отношения, A называется *областью отправления*, а B — *областью прибытия*. Если множества A и B определены контекстом, то часто просто говорят, что задано отношение R. При этом для краткости отношение обозначают тем же символом, что и график.

ЗАМЕЧАНИЕ

Принято говорить «отношение между множествами», хотя порядок множеств имеет значение, и отношение между A и B совсем не то же самое, что отношение между B и A. Поэтому иногда, чтобы подчеркнуть это обстоятельство, употребляют оборот «отношение R из множества A в множество B».

Среди элементов множеств A и B могут быть такие, которые не входят ни в одну из пар, определяющих отношение R. Множество элементов области отправления, которые присутствуют в парах отношения, называется *областью определения* отношения R и обозначается $\mathrm{Dom}\, R$, а множество элементов области прибытия, которые присутствуют в парах отношения, называется *областью значений* отношения R и обозначается $\mathrm{Im}\, R$:

$$\mathrm{Dom}\, R \stackrel{\mathrm{Def}}{=} \{a \in A \mid \exists b \in B \ \ ((a,b) \in R)\},$$

$$\mathrm{Im}\, R \stackrel{\mathrm{Def}}{=} \{b \in B \mid \exists a \in A \ \ ((a,b) \in R)\}.$$

Если $A = B$ (то есть $R \subset A^2$), то говорят, что R есть отношение *на множестве* A.

Для бинарных отношений обычно используется *инфиксная* форма записи:

$$aRb \stackrel{\mathrm{Def}}{=} (a,b) \in R \subset A \times B.$$

Инфиксная форма позволяет более кратко записывать некоторые формы утверждений относительно отношений:

$$aRbRc \stackrel{\mathrm{Def}}{=} (a,b) \in R \ \&\ (b,c) \in R.$$

Примеры

Пусть задано множество U. Тогда $\in$ (принадлежность) — отношение между элементами множества U и элементами булеана 2^U, а $\subset$ (включение) и $=$ (равенство) — отношения на 2^U. Хорошо известны отношения $=, <, \leqslant, >, \geqslant, \neq$, определённые на множестве вещественных чисел.

Пусть R есть отношение между A и B: $R \subset A \times B$. Введём следующие понятия:

Обратное отношение:	$R^{-1} \stackrel{\mathrm{Def}}{=} \{(a,b) \mid (b,a) \in R\} \subset B \times A.$
Дополнение отношения:	$\overline{R} \stackrel{\mathrm{Def}}{=} \{(a,b) \mid (a,b) \notin R\} \subset A \times B.$
Тождественное отношение:	$I \stackrel{\mathrm{Def}}{=} \{(a,a) \mid a \in A\} \subset A^2.$
Универсальное отношение:	$U \stackrel{\mathrm{Def}}{=} \{(a,b) \mid a \in A \ \&\ b \in B\} = A \times B.$

ЗАМЕЧАНИЕ

График тождественного отношения на множестве A иногда называют *диагональю* в $A \times A$.

Введем обобщённое понятие отношения: *n-местное* (*n-арное*) отношение R — это подмножество прямого произведения n множеств, то есть множество упорядоченных наборов (*кортежей*):

$$R \subset A_1 \times \cdots \times A_n \stackrel{\mathrm{Def}}{=} \{(a_1, \ldots, a_n) \mid a_1 \in A_1 \ \&\ \ldots \ \&\ a_n \in A_n\}.$$

ЗАМЕЧАНИЕ

Число n, то есть длину кортежей отношения, называют иногда *вместимостью*.

ОТСТУПЛЕНИЕ

Многоместные отношения используются, например, в теории баз данных. Само название «реляционная» база данных происходит от слова relation (отношение).

Далее рассматриваются только двуместные (бинарные) отношения, при этом слово «бинарные» опускается.

1.4.4. Композиция отношений

Пусть $R_1 \subset A \times C$ — отношение между A и C, а $R_2 \subset C \times B$ — отношение между C и B. *Композицией* двух отношений R_1 и R_2 называется отношение $R \subset A \times B$ между A и B, определяемое следующим образом:

$$R \stackrel{\text{Def}}{=} R_1 \circ R_2 \stackrel{\text{Def}}{=} \{(a,b) \mid a \in A \;\&\; b \in B \;\&\; \exists c \in C \;\; (aR_1c \;\&\; cR_2b)\}.$$

Другими словами,

$$aR_1 \circ R_2 b \Longleftrightarrow \exists c \in C \;\; (aR_1c \;\&\; cR_2b).$$

ТЕОРЕМА *Композиция отношений ассоциативна, то есть*

$$\forall R_1 \subset A \times B, R_2 \subset B \times C, R_3 \subset C \times D \;\; ((R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)).$$

Доказательство

$$\begin{aligned} a(R_1 \circ R_2) \circ R_3 d &\Longleftrightarrow \exists c \in C \;\; (aR_1 \circ R_2 c \;\&\; cR_3 d) \Longleftrightarrow \\ &\Longleftrightarrow \exists c \in C \;\; ((\exists b \in B \;\; (aR_1 b \;\&\; bR_2 c)) \;\&\; cR_3 d) \Longleftrightarrow \\ &\Longleftrightarrow \exists b \in B, c \in C \;\; (aR_1 b \;\&\; bR_2 c \;\&\; cR_3 d) \Longleftrightarrow \\ &\Longleftrightarrow \exists b \in B \;\; (aR_1 b \;\&\; (\exists c \in C \;\; (bR_2 c \;\&\; cR_3 d))) \Longleftrightarrow \\ &\Longleftrightarrow \exists b \in B \;\; (aR_1 b \;\&\; bR_2 \circ R_3 d) \Longleftrightarrow aR_1 \circ (R_2 \circ R_3) d. \end{aligned}$$

□

Композиция отношений на множестве A является отношением на множестве A.

Пример Композиция отношений $<$ и $\leqslant$ на множестве вещественных чисел совпадает с отношением $<$: $< \circ \leqslant \; = \; <$.

ЗАМЕЧАНИЕ

В общем случае композиция отношений не коммутативна, то есть $R_1 \circ R_2 \neq R_2 \circ R_1$.

Пример На множестве людей определены отношения кровного родства и супружества. Отношения «кровные родственники супругов» и «супруги кровных родственников» различны.

1.4.5. Степень отношения

Пусть R — отношение на множестве A. *Степенью* отношения R на множестве A называется его n-кратная композиция с самим собой. Обозначение:

$$R^n \stackrel{\text{Def}}{=} \underbrace{R \circ \cdots \circ R}_{n \text{ раз}}.$$

Соответственно, $R^0 \stackrel{\text{Def}}{=} I$, $R^1 \stackrel{\text{Def}}{=} R$, $R^2 \stackrel{\text{Def}}{=} R \circ R$ и вообще $R^n \stackrel{\text{Def}}{=} R^{n-1} \circ R$.

ТЕОРЕМА *Если некоторая пара (a, b) принадлежит какой-либо степени отношения R на множестве A мощности n, то эта пара принадлежит и некоторой степени R не выше $n - 1$:*

$$R \subset A^2 \ \&\ |A| = n \Longrightarrow \forall a, b \in A \ \left(\exists k \ \left(aR^k b\right) \Longrightarrow \exists k < n \ \left(aR^k b\right)\right).$$

Доказательство По определению

$$(a, b) \in R^k \Longrightarrow \exists c_1, \ldots, c_{k-1} \in A \ \ (c)_0 \, Rc_1 Rc_2 R \ldots Rc_{k-1} Rc_k.$$

Далее, если $|A| = n$ & $k \geqslant n$, то $\exists i, j \ \ (c_i = c_j \ \& \ i \neq j)$, и значит,

$$c_0 Rc_1 R \ldots Rc_i Rc_{j+1} R \ldots Rc_{k-1} Rc_k,$$

то есть $(a, b) \in R^{k-(j-i)}$. Обозначим через d процедуру, которая вычисляет пару чисел (i, j). Существование требуемого числа k (степени отношения R) обеспечивается следующим построением:

while $k \geqslant n$ **do**
  $c_0 := a$; $c_k := b$
  $(i, j) := d()$
  $k := k - (j - i)$
end while □

СЛЕДСТВИЕ $R \subset A^2 \ \&\ |A| = n \Longrightarrow \bigcup_{i=1}^{\infty} R^i = \bigcup_{i=1}^{n-1} R^i$.

1.4.6. Свойства отношений

Пусть $R \subset A^2$. Тогда отношение R называется

рефлексивным,	если $\forall a \in A \ \ (aRa)$;
антирефлексивным,	если $\forall a \in A \ \ (\neg aRa)$;
симметричным,	если $\forall a, b \in A \ \ (aRb \Longrightarrow bRa)$;
антисимметричным,	если $\forall a, b \in A \ \ (aRb \ \& \ bRa \Longrightarrow a = b)$;
транзитивным,	если $\forall a, b, c \in A \ \ (aRb \ \& \ bRc \Longrightarrow aRc)$;
линейным,	если $\forall a, b \in A \ \ (a = b \vee aRb \vee bRa)$.

ЗАМЕЧАНИЕ

Иногда линейное отношение R называют *полным*. Такой термин является оправданным, поскольку для любых двух различных элементов a и b либо пара (a,b), либо пара (b,a) принадлежит отношению R. Отношение, не обладающее свойством полноты, при этом вполне естественно называть *частичным*. Однако использование термина «полное отношение» может порождать терминологическую путаницу. Дело в том, что устойчивое словосочетание «вполне упорядоченное множество», рассматриваемое в подразделе 1.8.6, оказывается созвучным словосочетанию «множество с полным порядком», хотя эти словосочетания обозначают существенно различные объекты. Во избежание неоднозначности мы используем термин «линейное отношение» как антоним термина «частичное отношение».

ТЕОРЕМА *Пусть $R \subset A \times A$ — отношение на A. Тогда:*

1. *R рефлексивно* $\iff I \subset R$;
2. *R симметрично* $\iff R = R^{-1}$;
3. *R транзитивно* $\iff R \circ R \subset R$;
4. *R антисимметрично* $\iff R \cap R^{-1} = I$;
5. *R антирефлексивно* $\iff R \cap I = \varnothing$;
6. *R линейно* $\iff R \cup I \cup R^{-1} = U$.

Доказательство Используя определения свойств отношений, имеем:

[1. $\iff$] $\forall a \in A\ (aRa) \iff \forall a \in A\ ((a,a) \in R) \iff I \subset R$.

[2. $\iff$] $\forall a,b \in A\ (aRb \implies bRa) \iff \forall a,b \in A\ ((a,b) \in R \implies (b,a) \in R) \iff$
$\iff \forall a,b \in A\ (((a,b) \in R \implies (b,a) \in R)\ \&\ ((b,a) \in R \implies (a,b) \in R)) \iff$
$\iff \forall a,b \in A\ \big(((a,b) \in R \implies (a,b) \in R^{-1})\ \&\ ((a,b) \in R^{-1} \implies (a,b) \in R)\big) \iff$
$\iff R \subset R^{-1}\ \&\ R^{-1} \subset R \iff R = R^{-1}$.

[3. $\iff$] $\forall a,b,c \in A\ (aRb\ \&\ bRc \implies aRc) \iff$
$\iff \forall a,b,c \in A\ ((a,b) \in R\ \&\ (b,c) \in R \implies (a,c) \in R) \iff$
$\iff \forall a,c \in A\ (\exists b \in B\ ((a,b) \in R\ \&\ (b,c) \in R) \implies (a,c) \in R) \iff$
$\iff \forall a,c \in A\ ((a,c) \in R \circ R \implies (a,c) \in R) \iff$
$\iff \forall a,c \in A\ (aR \circ Rc \implies aRc) \iff R \circ R \subset R$.

[4. $\implies$] От противного. $R \cap R^{-1} \neq I \implies \exists a \neq b\ \big(aRb\ \&\ aR^{-1}b\big) \implies$
$\implies \exists a \neq b\ (aRb\ \&\ bRa)$.

[4. $\impliedby$] $R \cap R^{-1} = I \implies (aRb\ \&\ aR^{-1}b \implies a = b) \implies (aRb\ \&\ bRa \implies a = b)$.

[5. $\implies$] От противного. $R \cap I \neq \varnothing \implies$
$\implies \exists a \in A\ (aRa\ \&\ aIa) \iff \exists a \in A\ (aRa) \iff \neg\forall a \in A\ (\neg aRa)$.

[5. $\impliedby$] $R \cap I = \varnothing \implies \neg\exists a \in A\ (aRa) \implies \forall a \in A\ (\neg aRa)$.

[6. $\iff$] $\forall a,b \in A\ (a = b \vee aRb \vee bRa) \iff$
$\iff \forall a,b \in A\ \big((a,b) \in I \vee (a,b) \in R \vee (a,b) \in R^{-1}\big) \iff$
$\iff U \subset I \cup R \cup R^{-1} \iff U = R \cup I \cup R^{-1}$. □

1.4.7. Ядро отношения

Если $R \subset A \times B$ — отношение между множествами A и B, то композиция $R \circ R^{-1}$ называется *ядром* отношения R и обозначается $\ker R$. Другими словами,

$$a_1 \ker R\, a_2 \iff \exists b \in B \ (a_1 R b \ \&\ a_2 R b).$$

Ядро отношения R между A и B является отношением на A:

$$R \subset A \times B \Longrightarrow \ker R \subset A^2.$$

Примеры

1. Пусть отношение N_1 «находиться на расстоянии не более 1» на множестве целых чисел определено следующим образом:

   $$N_1 := \{(n, m) \mid |n - m| \leqslant 1\}.$$

   Тогда ядром этого отношения является отношение N_2 «находиться на расстоянии не более 2»:

   $$N_2 := \{(n, m) \mid |n - m| \leqslant 2\}.$$

2. Ядром отношения «быть знакомыми» является отношение «быть знакомыми или иметь общего знакомого».
3. Ядро отношения $\in$ между U и 2^U универсально: $\forall a, b \in U \ (\{a, b\} \in 2^U)$.
4. Ядро отношения $\subset$ на 2^U также универсально.
5. Рассмотрим отношение «тесного включения» на 2^U:

   $$X \overset{+1}{\subset} Y \overset{\text{Def}}{=} X \subset Y \ \&\ |X| + 1 = |Y|.$$

   Ядром отношения $\overset{+1}{\subset}$ является отношение «отличаться не более чем одним элементом»:

   $$\ker \overset{+1}{\subset} = \{X, Y \in 2^U \mid |X| = |Y| \ \&\ |X \cap Y| \geqslant |X| - 1\}.$$

ЗАМЕЧАНИЕ

Термин «ядро» и обозначение ker используются также и для других объектов. Их не следует путать — из контекста обычно ясно, в каком смысле используется термин «ядро».

ТЕОРЕМА *Ядро любого отношения рефлексивно и симметрично на области определения:*

$$\forall R \subset A \times B \ (\forall a \in \operatorname{Dom} R \ (a \ker R\, a) \ \&\ \forall a, b \in \operatorname{Dom} R \ (a \ker R\, b \Longrightarrow b \ker R\, a)).$$

Доказательство

[Рефлексивность] Пусть $a \in \operatorname{Dom} R$. Тогда $\exists b \in B \ (aRb) \Longrightarrow$
$\Longrightarrow \exists b \in B \ (aRb \ \&\ bR^{-1}a) \Longrightarrow (a, a) \in R \circ R^{-1} \Longrightarrow a \ker R\, a.$

[Симметричность] Пусть $a, b \in \operatorname{Dom} R \ \&\ a \ker R\, b$. Тогда $\exists c \in B \ (aRc \ \&\ cR^{-1}b) \Longrightarrow$
$\Longrightarrow \exists c \in B \ (bRc \ \&\ cR^{-1}a) \Longrightarrow (b, a) \in R \circ R^{-1} \Longrightarrow b \ker R\, a.$ □

1.4.8. Представление отношений в программах

Пусть R отношение на A, $R \subset A^2$ и $|A| = n$. Перенумеруем элементы множества A, $A = \{a_1, \ldots, a_n\}$. Тогда отношение R можно представить *булевой* матрицей $\boxed{R}$: **array** $[1..n, 1..n]$ **of** 0..1, где $\forall i, j \in 1..n \ \left(\boxed{R}[i,j] = a_i R a_j\right)$.

ЗАМЕЧАНИЕ

Термин «булева матрица» означает, что элементы матрицы — булевские значения и операции над ними выполняются по соответствующим правилам (глава 3). В частности, если $\boxed{A}$ и $\boxed{B}$ — булевы матрицы, то операции на ними определяются следующим образом:

транспонирование: $\boxed{A}^T[i,j] \stackrel{\text{Def}}{=} \boxed{A}[j,i]$;

вычитание: $\left(\boxed{A} - \boxed{B}\right)[i,j] \stackrel{\text{Def}}{=} \boxed{A}[i,j] \ \&\ \left(1 - \boxed{B}[i,j]\right)$;

инвертирование: $\overline{\boxed{A}}[i,j] \stackrel{\text{Def}}{=} 1 - \boxed{A}[i,j]$;

умножение: $\left(\boxed{A} \times \boxed{B}\right)[i,j] \stackrel{\text{Def}}{=} \bigvee_{k=1}^{n} \left(\boxed{A}[i,k] \ \&\ \boxed{B}[k,j]\right)$;

дизъюнкция: $\left(\boxed{R_1} \vee \boxed{R_2}\right)[i,j] \stackrel{\text{Def}}{=} \boxed{R_1}[i,j] \vee \boxed{R_2}[i,j]$.

В следующих утверждениях предполагается, что все рассматриваемые отношения определены на множестве A, причём $|A| = n$.

ТЕОРЕМА 1 $\boxed{R^{-1}} = \boxed{R}^T$.

Доказательство $(b,a) \in R^{-1} \Longleftrightarrow (a,b) \in R \Longleftrightarrow \boxed{R}[a,b] = 1 \Longleftrightarrow \boxed{R}^T[b,a] = 1$. □

В универсальное отношение U входят все пары элементов, поэтому все элементы матрицы универсального отношения $\boxed{U}$ равны 1.

ТЕОРЕМА 2 $\boxed{\overline{R}} = \boxed{U} - \boxed{R}$.

Доказательство $(a,b) \in \overline{R} \Longleftrightarrow (a,b) \notin R \Longleftrightarrow \boxed{R}[a,b] = 0 \Longleftrightarrow \boxed{U}[a,b] - \boxed{R}[a,b] = 1 \Longleftrightarrow \left(\boxed{U} - \boxed{R}\right)[a,b] = 1$. □

СЛЕДСТВИЕ $\boxed{\overline{R}} = \overline{\boxed{R}}$.

ТЕОРЕМА 3 $\boxed{R_1 \circ R_2} = \boxed{R_1} \times \boxed{R_2}$.

Доказательство $(a,b) \in R_1 \circ R_2 \iff \exists c \in A\ (aR_1c\ \&\ cR_2b) \iff \exists c \in A\ \left(\boxed{R_1}[a,c] = 1\ \&\ \boxed{R_2}[c,b] = 1\right) \iff \exists c \in A\ \left(\left(\boxed{R_1}[a,c]\ \&\ \boxed{R_2}[c,b]\right) = 1\right) \iff$
$\iff \left(\bigvee_{k=1}^{n} \boxed{R_1}[a,k]\ \&\ \boxed{R_2}[k,b]\right) = 1 \iff \left(\boxed{R_1} \times \boxed{R_2}\right)[a,b] = 1.$ □

СЛЕДСТВИЕ $\boxed{R^k} = \boxed{R}^k$.

ТЕОРЕМА 4 $\boxed{R_1 \cup R_2} = \boxed{R_1} \vee \boxed{R_2}$.

Доказательство $(a,b) \in R_1 \cup R_2 \iff aR_1b \vee aR_2b \iff \boxed{R_1}[a,b] = 1 \vee \boxed{R_2}[a,b] = 1 \iff \left(\boxed{R_1} \vee \boxed{R_2}\right)[a,b] = 1.$ □

ЗАМЕЧАНИЕ

Представление отношений с помощью булевых матриц — это только один из возможных способов представления отношений. Другие варианты рассматриваются при обсуждении представления графов в главе 7.

1.5. Замыкание отношений

Замыкание является весьма общим математическим понятием, рассмотрение конкретных вариантов которого начинается в этом разделе и продолжается в других главах книги. Неформально говоря, замкнутость означает, что многократное выполнение допустимых шагов не выводит за определённые границы. Например, предел сходящейся последовательности чисел из *замкнутого* интервала $[a,b]$ обязательно принадлежит этому интервалу, а предел сходящейся последовательности чисел из открытого (то есть *не* замкнутого) интервала (a,b) может лежать вне этого интервала. В некоторых случаях можно «расширить» незамкнутый объект так, чтобы он оказался замкнутым.

1.5.1. Транзитивное и рефлексивное замыкание

Пусть R и R' — отношения на множестве M. Отношение R' называется *замыканием* R относительно свойства C, если:

1) R' обладает свойством C: $C(R')$;
2) R' является надмножеством R: $R \subset R'$;
3) R' является наименьшим таким объектом: $C(R'')\ \&\ R \subset R'' \Longrightarrow R' \subset R''$.

Пусть R^+ — объединение положительных степеней R, а R^* — объединение неотрицательных степеней R:

$$R^+ \stackrel{\text{Def}}{=} \bigcup_{i=1}^{\infty} R^i, \qquad R^* \stackrel{\text{Def}}{=} \bigcup_{i=0}^{\infty} R^i.$$

ТЕОРЕМА *R^+ — транзитивное замыкание R.*

Доказательство Проверим выполнение свойств замыкания при условии, что свойство C — это транзитивность.

[$C(R^+)$] Пусть aR^+b & bR^+c. Тогда $\exists n\ (aR^nb)$ & $\exists m\ (bR^mc)$. Следовательно, $\exists c_1, \ldots, c_{n-1}\ (aRc_1R \ldots Rc_{n-1}Rb)$ и $\exists d_1, \ldots, d_{m-1}\ (bRd_1R \ldots Rd_{m-1}Rc)$. Таким образом, $aRc_1R \ldots Rc_{n-1}RbRd_1R \ldots Rd_{m-1}Rc \Longrightarrow aR^{n+m+1}c \Longrightarrow aR^+c$.

[$R \subset R^+$] $R = R^1 \Longrightarrow R \subset \bigcup_{i=1}^{\infty} R^i \Longrightarrow R \subset R^+$.

[$C(R'')$ & $R \subset R'' \Longrightarrow R^+ \subset R''$] $aR^+b \Longrightarrow \exists k\ (aR^kb) \Longrightarrow$
$\Longrightarrow \exists c_1, \ldots, c_{k-1}\ (aRc_1R \ldots Rc_{k-1}Rb)$; $R \subset R'' \Longrightarrow aR''c_1R'' \ldots R''c_{k-1}R''b \Longrightarrow$
$\Longrightarrow aR''b$. Таким образом, $R^+ \subset R''$. □

ТЕОРЕМА *R^* — рефлексивное транзитивное замыкание R.*

Доказательство Упражнение 1.5. □

Вычислить транзитивное замыкание заданного отношения можно следующим образом:

$$R^+ = \bigcup_{i=1}^{\infty} R^i = \bigcup_{i=1}^{n-1} R^i \Longrightarrow \boxed{R^+} = \bigvee_{i=1}^{n-1} \boxed{R^i} = \bigvee_{i=1}^{n-1} \boxed{R}^i.$$

Такое вычисление будет иметь сложность $O(n^4)$.

1.5.2. Алгоритм Уоршалла

Рассмотрим алгоритм Уоршалла[1] вычисления транзитивного замыкания отношения R на множестве M, $|M| = n$, имеющий сложность $O(n^3)$.

Алгоритм 1.11 Вычисление транзитивного замыкания отношения

Вход: отношение, заданное матрицей R: **array**$[1..n, 1..n]$ **of** 0..1.
Выход: транзитивное замыкание отношения, заданное матрицей
T: **array**$[1..n, 1..n]$ **of** 0..1.

```
S := R
for i from 1 to n do
  for j from 1 to n do
```

[1] Стефан Уоршалл (р. 1935).

```
    for k from 1 to n do
      T[j,k] := S[j,k] ∨ S[j,i] & S[i,k]
    end for
  end for
  S := T
end for
```

Обоснование На каждом шаге основного цикла (по i) в транзитивное замыкание добавляются такие пары элементов с номерами j и k (то есть $T[j,k] := 1$), для которых существует последовательность промежуточных элементов с номерами в диапазоне $1 \ldots i$, связанных отношением R. Действительно, последовательность промежуточных элементов с номерами в диапазоне $1 \ldots i$, связанных отношением R, для элементов с номерами j и k существует в одном из двух случаев: либо уже существует последовательность промежуточных элементов с номерами в диапазоне от $1 \ldots (i-1)$ для пары элементов с номерами j и k, либо существуют *две* последовательности элементов с номерами в диапазоне от $1 \ldots (i-1)$ — одна для пары элементов с номерами j и i и вторая для пары элементов с номерами i и k. Именно это отражено в операторе $T[j,k] := S[j,k] \vee S[j,i] \,\&\, S[i,k]$. После окончания основного цикла в качестве промежуточных рассмотрены все элементы, и, таким образом, построено транзитивное замыкание. □

1.6. Функции

Понятие «функция» является одним из основополагающих в математике. В данном случае подразумеваются прежде всего функции, отображающие одно конечное множество объектов в другое конечное множество. Термин «отображение» мы считаем синонимом термина «функция», но различаем контексты употребления. Термин «функция» используется в общем случае и в тех случаях, когда элементы отображаемых множеств не имеют структуры или она не важна. Термин «отображение» применяется, напротив, только в тех случаях, когда структура отображаемого множества имеет первостепенное значение. Например, мы говорим «булева функция», но «отображение групп». Такое предпочтение слову «функция» оказывается в расчёте на постоянное сопоставление читателем математического понятия функции с понятием функции в языках программирования.

1.6.1. Функциональные отношения

Пусть f — отношение между A и B, такое, что

$$\forall a \ ((a,b) \in f \,\&\, (a,c) \in f \Longrightarrow b = c).$$

Такое свойство отношения называется *однозначностью*, или *функциональностью*, а само отношение называется *функцией* из A в B, причём для записи используется одна из следующих форм:

$$f\colon A \to B \qquad \text{или} \qquad A \xrightarrow{f} B.$$

Если $f\colon A \to B$, то обычно используется *префиксная* форма записи:

$$b = f(a) \stackrel{\text{Def}}{=} (a, b) \in f.$$

Если $b = f(a)$, то a называют *аргументом*, а b — *значением* функции. Если из контекста ясно, о какой функции идет речь, то иногда используется обозначение $a \mapsto b$. Поскольку функция является отношением, для функции $f\colon A \to B$ можно использовать уже введенные понятия: множество A называется *областью отправления*, множество B — *областью прибытия* и

область определения функции: $\mathrm{Dom}\, f \stackrel{\text{Def}}{=} \{a \in A \mid \exists b \in B\ (b = f(a))\}$;

область значений функции: $\mathrm{Im}\, f \stackrel{\text{Def}}{=} \{b \in B \mid \exists a \in A\ (b = f(a))\}$.

Область определения является подмножеством области отправления, $\mathrm{Dom}\, f \subset A$, а область значений является подмножеством области прибытия, $\mathrm{Im}\, f \subset B$. Если $\mathrm{Dom}\, f = A$, то функция называется *тотальной*, а если $\mathrm{Dom}\, f \neq A$ — *частичной*. *Сужением* функции $f\colon A \to B$ на множество $M \subset A$ называется функция $f|_M$, определяемая следующим образом:

$$f|_M \stackrel{\text{Def}}{=} \{(a, b) \mid (a, b) \in f \ \&\ a \in M\} = f \cap (M \times B).$$

Функция f называется *продолжением* g, если g является сужением f.

ОТСТУПЛЕНИЕ

В математике, как правило, рассматриваются тотальные функции, а частичные функции объявлены «ущербными». Программисты же имеют дело с частичными функциями, которые определены не для всех допустимых значений. Например, математическая функция x^2 определена для всех x, а стандартная функция языка программирования sqr даёт правильный результат отнюдь не для всех возможных значений аргумента.

Далее, если явно не оговорено противное, речь идёт о тотальных функциях, поэтому области отправления и определения совпадают. Тотальная функция $f\colon M \to M$ называется *преобразованием* над M. Функция $f\colon A_1 \times \cdots \times A_n \to B$ называется функцией n *аргументов*, или n*-местной* функцией. Такая функция отображает кортеж $(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n$ в элемент $b \in B$, $b = f((a_1, \ldots, a_n))$. При записи лишние скобки обычно опускают: $b = f(a_1, \ldots, a_n)$. Понятие функции удобно использовать для построения самых разнообразных математических моделей.

Примеры

1. Всякому подмножеству A универсума U можно взаимно-однозначно сопоставить тотальную функцию $1_A\colon U \to 0..1$. Эта функция называется *характеристической* функцией подмножества и определяется следующим образом:

$$1_A(a) \stackrel{\text{Def}}{=} \begin{cases} 1, & \text{если } a \in A, \\ 0, & \text{если } a \notin A. \end{cases}$$

Характеристическая функция результата операции над множествами легко выражается через характеристические функции операндов:

$$1_{A\cup B} \stackrel{\text{Def}}{=} \max(1_A, 1_B), \quad 1_{A\cap B} \stackrel{\text{Def}}{=} \min(1_A, 1_B),$$
$$1_{\overline{A}} \stackrel{\text{Def}}{=} 1 - 1_A, \quad 1_{A\setminus B} \stackrel{\text{Def}}{=} \min(1_A, 1 - 1_B).$$

2. Всякому отношению R между A и B ($R \subset A \times B$) можно взаимно-однозначно сопоставить тотальную функцию $1_R\colon A \times B \to 0..1$ (эта функция называется *характеристической* функцией отношения), полагая

$$R(a,b) \stackrel{\text{Def}}{=} \begin{cases} 1, & \text{если } aRb, \\ 0, & \text{если } a\overline{R}b. \end{cases}$$

Характеристическая функция результата операции над отношениями легко выражается через характеристические функции операндов:

$$1_{R^{-1}}(a,b) \stackrel{\text{Def}}{=} 1_R(b,a), \quad 1_{\overline{R}} \stackrel{\text{Def}}{=} 1 - 1_R.$$

1.6.2. Инъекция, сюръекция и биекция

Пусть $f\colon A \to B$. Тогда функция f называется:

инъективной,	или *инъекцией,*	если $b = f(a_1)\ \&\ b = f(a_2) \Longrightarrow a_1 = a_2$;
сюръективной,	или *сюръекцией,*	если $\forall b \in B\ (\exists a \in A\ (b = f(a)))$;
биективной,	или *биекцией,*	если она инъективная и сюръективная.

ЗАМЕЧАНИЕ

Биективную функцию также называют *взаимно-однозначной.* Введенное в подразделе 1.2.2 взаимно-однозначное соответствие есть биекция.

Рисунок 1.3 иллюстрирует понятия отношения, функции, инъекции, сюръекции и биекции.

Понятия инъективности, сюръективности и тотальности применимы к любым отношениям, а не только к функциям. Таким образом, получаем четыре парных свойства отношения $f \subset A \times B$, которые для удобства запоминания можно свести в следующую таблицу.

A	B
Функциональность	Инъективность
$\forall a \in A\ ((a,b) \in f\ \&\ (a,c) \in f \Longrightarrow b = c)$	$\forall b \in B\ ((a,b) \in f\ \&\ (c,b) \in f \Longrightarrow a = c)$
Тотальность	Сюръективность
$\forall a \in A\ (\exists b \in B\ ((a,b) \in f))$	$\forall b \in B\ (\exists a \in A\ ((a,b) \in f))$

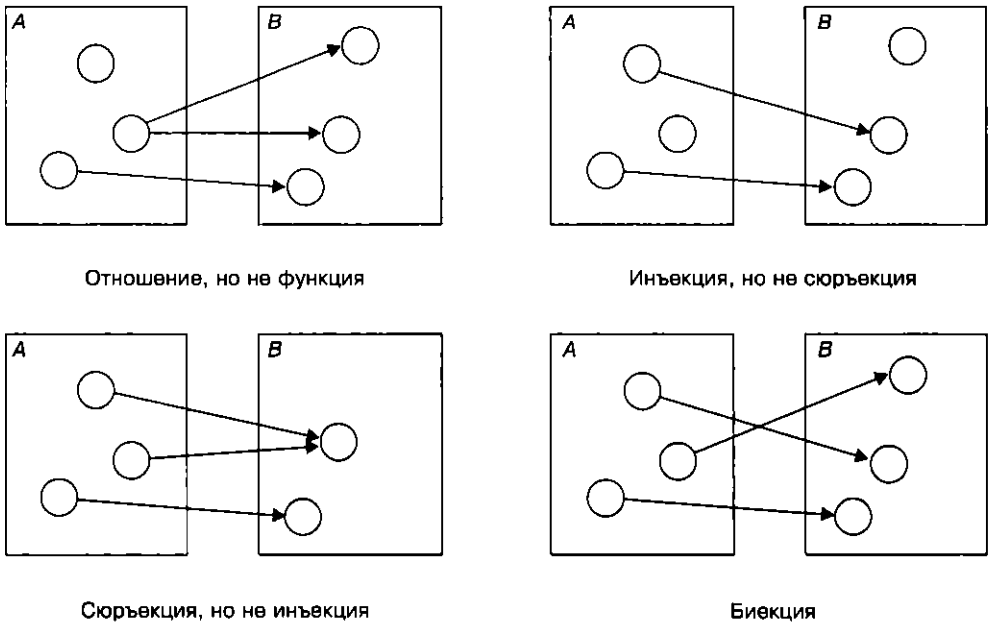

Рис. 1.3. Различные виды функций

ТЕОРЕМА *Если* $f\colon A \to B$ *— тотальная биекция, то отношение* $f^{-1} \subset B \times A$ *(обратная функция) является биекцией.*

Доказательство Поскольку f — биекция, имеем $\big(b_1 = f(a)\ \&\ b_2 = f(a) \Longrightarrow b_1 = b_2\big)\ \&\ \big(b = f(a_1)\ \&\ b = f(a_2) \Longrightarrow a_1 = a_2\big)\ \&\ \big(\forall b \in B\ \ (\exists a \in A\ \ (b = f(a)))\big)$. Покажем, что f^{-1} — функция. Имеем $f^{-1} \stackrel{\text{Def}}{=} \{(b, a) \mid a \in A\ \&\ b \in B\ \&\ b = f(a)\}$. Пусть $a_1 = f^{-1}(b)\ \&\ a_2 = f^{-1}(b)$. Тогда $b = f(a_1)\ \&\ b = f(a_2) \Longrightarrow a_1 = a_2$. Покажем, что f^{-1} — инъекция. Пусть $a = f^{-1}(b_1)\ \&\ a = f^{-1}(b_2)$. Тогда $b_1 = f(a)\ \&\ b_2 = f(a) \Longrightarrow b_1 = b_2$. Покажем от противного, что f^{-1} — сюръекция. Пусть $\exists a \in A\ \ \big(\neg\exists b \in B\ \ (a = f^{-1}(b))\big)$. Тогда $\exists a \in A\ \ \big(\forall b \in B\ \ (a \neq f^{-1}(b))\big)$. Обозначим этот элемент a_0. Имеем $\forall b\ \ (a_0 \neq f^{-1}(b)) \Longrightarrow \forall b\ \ (b \neq f(a_0)) \Longrightarrow$ $\Longrightarrow a_0 \notin f_A \subset A \Longrightarrow a_0 \notin A$. □

СЛЕДСТВИЕ *Если* $f\colon A \to B$ *— инъективная функция, то отношение* $f^{-1} \subset B \times A$ *является функцией (возможно, частичной),* $f^{-1}\colon B \to A$.

1.6.3. Образы и прообразы

Пусть $f\colon A \to B$ и пусть $A_1 \subset A$, $B_1 \subset B$. Тогда множество

$$F(A_1) \stackrel{\text{Def}}{=} \{b \in B \mid \exists a \in A_1\ \ (b = f(a))\}$$

называется *образом* множества A_1 (при отображении f), а множество

$$F^{-1}(B_1) \stackrel{\text{Def}}{=} \{a \in A \mid \exists b \in B_1 \ (b = f(a))\}$$

называется *прообразом* множества B_1 (при отображении f). Введём обозначения $f_A := \operatorname{Dom} f$, $f_B := \operatorname{Im} f$. Заметим, что F является отношением между множествами 2^{f_A} и 2^{f_B}:

$$F \stackrel{\text{Def}}{=} \{(A_1, B_1) \mid A_1 \subset A \ \& \ B_1 \subset B \ \& \ B_1 = F(A_1)\}.$$

ТЕОРЕМА *Если $f\colon A \to B$ — функция, то $F\colon 2^{f_A} \to 2^{f_B}$ (переход к образам) и $F^{-1}\colon 2^{f_B} \to 2^{f_A}$ (переход к прообразам) — тоже функции.*

Доказательство Упражнение 1.6. □

ЗАМЕЧАНИЕ

Фактически, это означает, что функцию можно применять не только к элементам области определения, а и к любым её подмножествам. На практике в целях упрощения записи для обозначения применения функции к подмножеству используют ту же самую букву, что и для применения функции к отдельному элементу.

1.6.4. Суперпозиция функций

Поскольку функции являются частным случаем отношений, для них определена композиция. Композиция функций называется *суперпозицией*. Для обозначения суперпозиции применяют тот же знак $\circ$, но операнды записывают в обратном порядке: если $f\colon A \to B$ и $g\colon B \to C$, то суперпозиция функций f и g записывается так: $g \circ f$. Такой способ записи суперпозиции функций объясняется тем, что обозначение функции принято писать слева от списка аргументов:

$$(f \circ g)(x) \stackrel{\text{Def}}{=} f(g(x)).$$

ТЕОРЕМА *Суперпозиция функций является функцией:*

$$f\colon A \to B \ \& \ g\colon B \to C \Longrightarrow g \circ f\colon A \to C.$$

Доказательство Пусть $b = (g \circ f)(a)$ и $c = (g \circ f)(a)$. Тогда $b = g(f(a))$ и $c = g(f(a))$. Но f и g функции, поэтому $b = c$. □

1.6.5. Представление функций в программах

Пусть $f\colon A \to B$, множество A конечно и не очень велико, $|A| = n$. Наиболее общим представлением такой функции является массив **array** [A] **of** B, где A — тип данных, значения которого представляют элементы множества A, а B — тип данных, значения которого представляют элементы множества B. Если среда программирования допускает массивы только с натуральными индексами, то элементы множества A нумеруются (то есть $A = \{a_1, \ldots, a_n\}$) и функция представляется с помощью массива **array** $[1..n]$ **of** B. Функция нескольких аргументов представляется многомерным массивом.

ОТСТУПЛЕНИЕ

Представление функции с помощью массива является эффективным по времени, поскольку реализация массивов в большинстве случаев обеспечивает вычисление значения функции для заданного значения аргумента (индекса) за постоянное время, не зависящее от размера массива и значения индекса.

Если множество A велико или бесконечно, то использование массивов для представления функций является неэффективным с точки зрения экономии памяти. В таком случае для представления функций используется особый вид процедур, возвращающих единственное значение для заданного значения аргумента. Обычно такие процедуры также называются «функциями». В некоторых языках программирования определения функций вводятся ключевым словом **function**. Многоместные функции представляются с помощью нескольких формальных параметров в определении функции. Свойство функциональности обеспечивается оператором *возврата*, часто обозначаемым ключевым словом **return**, который прекращает выполнение тела функции и одновременно возвращает значение.

ОТСТУПЛЕНИЕ

В языке программирования Фортран и в некоторых других языках вызов функции и обращение к массиву синтаксически неразличимы, что подчеркивает родственность этих понятий.

1.7. Отношения эквивалентности

Различные встречающиеся на практике отношения могут обладать (или не обладать) теми или иными свойствами. Свойства, введенные в подразделе 1.4.6, встречаются особенно часто (потому им и даны специальные названия). Более того, оказывается, что некоторые устойчивые комбинации этих свойств встречаются настолько часто, что заслуживают отдельного названия и специального изучения. Здесь рассматриваются классы отношений, обладающих определённым набором свойств. Такое «абстрактное» изучение классов отношений обладает тем преимуществом, что, один раз установив некоторые следствия из наличия у отношения определённого набора свойств, далее эти следствия можно автоматически распространить на все конкретные отношения, которые обладают данным набором свойств. Рассмотрение начинается отношениями эквивалентности (в этом разделе) и продолжается отношениями порядка (в следующем разделе).

1.7.1. Классы эквивалентности

Рефлексивное симметричное транзитивное отношение называется отношением *эквивалентности*. Обычно отношение эквивалентности обозначают знаком $\equiv$.

Примеры

Отношения равенства чисел, равенства множеств являются отношениями эквивалентности. Отношение равномощности множеств также является отношением эквивалентности. Другие, более интересные примеры отношений эквивалентности, приведены в последующих главах книги.

Пусть $\equiv$ — отношение эквивалентности на множестве M и $x \in M$. Подмножество элементов множества M, эквивалентных x, называется *классом эквивалентности* для x:

$$[x]_{\equiv} \stackrel{\text{Def}}{=} \{y \in M \mid y \equiv x\}.$$

Если отношение подразумевается, то нижний индекс у обозначения класса эквивалентности (знак отношения) обычно опускают.

ЛЕММА 1 $\forall a \in M \ ([a] \neq \varnothing)$.

Доказательство $a \equiv a \Longrightarrow a \in [a]$. □

ЛЕММА 2 $a \equiv b \Longrightarrow [a] = [b]$.

Доказательство Пусть $a \equiv b$. Тогда $x \in [a] \Longleftrightarrow x \equiv a \Longleftrightarrow x \equiv b \Longleftrightarrow x \in [b]$. □

ЛЕММА 3 $a \not\equiv b \Longrightarrow [a] \cap [b] = \varnothing$.

Доказательство От противного: $[a] \cap [b] \neq \varnothing \Longrightarrow \exists c \in M \ (c \in [a] \cap [b]) \Longrightarrow$
$\Longrightarrow c \in [a] \,\&\, c \in [b] \Longrightarrow c \equiv a \,\&\, c \equiv b \Longrightarrow a \equiv c \,\&\, c \equiv b \Longrightarrow a \equiv b$, противоречие. □

ТЕОРЕМА *Если $\equiv$ — отношение эквивалентности на множестве M, то классы эквивалентности по этому отношению образуют разбиение множества M, причём среди элементов разбиения нет пустых; и обратно, всякое разбиение $\mathcal{B} = \{B_i\}$ множества M, не содержащее пустых элементов, определяет отношение эквивалентности на множестве M, классами эквивалентности которого являются элементы разбиения.*

Доказательство

[$\Longrightarrow$] Построение требуемого разбиения обеспечивается следующим алгоритмом.

Вход: множество M, отношение эквивалентности $\equiv$ на M.
Выход: разбиение $\mathcal{B}$ множества M на классы эквивалентности.

```
𝔅 := ∅ { вначале разбиение пусто }
for a ∈ M do
  for B ∈ 𝔅 do
    select b ∈ B { берём представителя из B }
    if a ≡ b then
      B := B + a { пополняем существующий класс эквивалентности }
      next for a { переходим к рассмотрению следующего элемента из M }
    end if
  end for
  𝔅 := 𝔅 ∪ {{a}} { вводим новый класс эквивалентности }
end for
```

[$\Longleftarrow$] Положим $a \equiv b \stackrel{\text{Def}}{=} \exists i\ (a \in B_i\ \&\ b \in B_i)$. Тогда отношение $\equiv$ рефлексивно, поскольку $M = \bigcup B_i \Longrightarrow \forall a \in M\ (\exists i\ (a \in B_i))$ и $a \in B_i \Longrightarrow a \in B_i\ \&\ a \in B_i \Longrightarrow$ $\Longrightarrow a \equiv a$. Отношение $\equiv$ симметрично, поскольку $a \equiv b \Longrightarrow \exists i\ (a \in B_i\ \&\ b \in B_i) \Longrightarrow$ $\Longrightarrow \exists i\ (b \in B_i\ \&\ a \in B_i) \Longrightarrow b \equiv a$. Отношение $\equiv$ транзитивно, поскольку $a \equiv$ $\equiv b\ \&\ b \equiv c \Longrightarrow [a] = [b]\ \&\ [b] = [c] \Longrightarrow [a] = [c] \Longrightarrow a \equiv c$. □

1.7.2. Фактормножества

Если R — отношение эквивалентности на множестве M, то множество классов эквивалентности называется *фактормножеством* множества M относительно эквивалентности R и обозначается M/R:

$$M/R \stackrel{\text{Def}}{=} \{[x]_R\}_{x \in M}.$$

Фактормножество является подмножеством булеана: $M/R \subset 2^M$. Функция $\operatorname{nat} R\colon M \to M/R$ называется *отождествлением* и определяется следующим образом:

$$\operatorname{nat} R(x) \stackrel{\text{Def}}{=} [x]_R.$$

1.7.3. Ядро функционального отношения и множества уровня

Всякая функция f, будучи отношением, имеет ядро $f \circ f^{-1}$, которое является отношением на области определения функции.

ЗАМЕЧАНИЕ

Даже если f — функция, f^{-1} отнюдь не обязательно функция, поэтому здесь $\circ$ — знак композиции отношений, а не суперпозиции функций.

Термин «ядро» применительно к функции f и обозначение $\ker f$ обычно резервируются для других целей, поэтому в формулировке следующей теоремы вместо слова «функция» употреблен несколько тяжеловесный оборот «функциональное отношение».

ТЕОРЕМА *Ядро функционального отношения является отношением эквивалентности на множестве определения.*

Доказательство Пусть $f\colon A \to B$. Достаточно рассматривать случай, когда $\operatorname{Dom} f = A$ и $\operatorname{Im} f = B$.

[**Рефлексивность**] Пусть $a \in A$. Тогда $\exists b \in B\ (b = f(a)) \Longrightarrow a \in f^{-1}(b) \Longrightarrow$ $\Longrightarrow (a, b) \in f\ \&\ (b, a) \in f^{-1} \Longrightarrow (a, a) \in f \circ f^{-1}$.

[**Симметричность**] Пусть $(a_1, a_2) \in f \circ f^{-1}$. Тогда $\exists b\ \big(b = f(a_1)\ \&\ a_2 \in f^{-1}(b)\big) \Longrightarrow$ $\Longrightarrow a_1 \in f^{-1}(b)\ \&\ b = f(a_2) \Longrightarrow b = f(a_2)\ \&\ a_1 \in f^{-1}(b) \Longrightarrow (a_2, a_1) \in f \circ f^{-1}$.

[Транзитивность] Пусть $(a_1, a_2) \in f \circ f^{-1}$ и $(a_2, a_3) \in f \circ f^{-1}$. Это означает, что $\exists b_1 \in B \ \big(b_1 = f(a_1)\ \&\ a_2 \in f^{-1}(b_1)\big)$ и $\exists b_2 \in B \ \big(b_2 = f(a_2)\ \&\ a_3 \in f^{-1}(b_2)\big)$. Тогда $b_1 = f(a_1)\ \&\ b_1 = f(a_2)\ \&\ b_2 = f(a_2)\ \&\ b_2 = f(a_3) \Longrightarrow b_1 = b_2$. Положим $b := b_1$ (или $b := b_2$). Тогда $b = f(a_1)\ \&\ b = f(a_3) \Longrightarrow b = f(a_1)\ \&\ a_3 \in f^{-1}(b) \Longrightarrow$ $\Longrightarrow (a_1, a_3) \in f \circ f^{-1}$. □

Пусть $f\colon A \to B$ — функция и $f \circ f^{-1}$ — отношение эквивалентности на $\mathrm{Dom}\, f$. Рассмотрим фактормножество $\mathrm{Dom}\, f / \big(f \circ f^{-1}\big)$. Класс эквивалентности (элемент фактормножества) — это подмножество элементов A, которые отображаются в один и тот же элемент $b \in B$. Такие множества называются *множествами уровня* функции f. Ясно, что $\big|\mathrm{Dom}\, f / \big(f \circ f^{-1}\big)\big| = |\mathrm{Im}\, f|$. Неформально множество уровня функции f, соответствующее значению c, — это множество решений уравнения $f(x) = c$.

Пример Множество уровня 0 для функции $\sin(x)$ — это $\pi\mathbb{Z}$.

ОТСТУПЛЕНИЕ

Термин «множество уровня» опирается на очевидные геометрические ассоциации. Если нарисовать график функции одной переменной в обычной декартовой системе координат и провести горизонтальную прямую на определённом уровне, то абсциссы точек пересечения с графиком функции дадут множество уровня функции.

Пример Пусть задано множество M, $|M| = n$. Рассмотрим функциональное отношение P между булеаном 2^M и множеством неотрицательных целых чисел (мощность):

$$P \subset 2^M \times 0..n, \quad \text{где } P := \{(X, k) \mid X \subset M\ \&\ k \in 0..n\ \&\ |X| = k\}.$$

Тогда ядром такого функционального отношения является отношение равномощности, а множествами уровня — семейства равномощных подмножеств.

1.8. Отношения порядка

В этом разделе определяются различные варианты отношений порядка. Отношение порядка позволяет *сравнивать* между собой различные элементы одного множества. Фактически, интуитивные представления об отношениях порядка уже были использованы при описании работы с упорядоченными списками в подразделах 1.3.4–1.3.7.

1.8.1. Определения

Антисимметричное транзитивное отношение называется отношением *порядка*. Отношение порядка может быть рефлексивным, и тогда оно называется отношением *нестрогого порядка*. Отношение порядка может быть антирефлексивным, и тогда оно называется отношением *строгого порядка*. Отношение порядка может быть линейным, и тогда оно называется отношением *линейного порядка*. Отноше-

ние порядка может не обладать свойством линейности, и тогда оно называется отношением *частичного порядка*. Обычно отношение строгого порядка (линейного или частичного) обозначается знаком $<$, а отношение нестрогого порядка — знаком $\leqslant$. Отношение порядка в общем случае обозначается знаком $\prec$.

ЗАМЕЧАНИЕ

Для строгого порядка свойство антисимметричности можно определить следующим образом: $\forall a, b\ (a < b \Longrightarrow \neg(b < a))$.

Примеры

Отношение $<$ на множестве вещественных чисел является отношением строгого линейного порядка. Отношение $\leqslant$ на множестве вещественных чисел является отношением нестрогого линейного порядка. Отношение $\subset$ на булеане 2^M является отношением нестрогого частичного порядка.

Множество, на котором задано отношение частичного порядка, называется *частично упорядоченным*. Множество, на котором задано отношение линейного порядка, называется *линейно упорядоченным*.

Пример Множество вещественных чисел упорядочено линейно, а булеан упорядочен частично.

1.8.2. Минимальные элементы

Элемент x множества M с отношением порядка $\prec$ называется *минимальным* в множестве M, если в M не существует меньших элементов: $\neg\exists\, y \in M\ (y \prec x\ \&\ y \neq x)$.

Пример Пустое множество $\varnothing$ является минимальным элементом булеана по включению.

ТЕОРЕМА 1 *Во всяком конечном непустом частично упорядоченном множестве существует минимальный элемент.*

Доказательство От противного. Пусть $\neg(\exists x \in M\ (\neg\exists y \in M\ (y \prec x\ \&\ y \neq x)))$. Тогда $\forall x \in M\ (\exists y \in M\ (y \prec x)) \Longrightarrow \exists (u_i)_{i=1}^{\infty}\ (\forall i\ (u_{i+1} \prec u_i)\ \&\ u_{i+1} \neq u_i)$. Поскольку $|M| < \infty$, имеем $\exists i, j\ (i < j\ \&\ u_i = u_j)$. Но по транзитивности

$$u_i \succ u_{i+1} \succ \cdots \succ u_j \Longrightarrow u_{i+1} \succ u_j = u_i.$$

Таким образом, $u_{i+1} \prec u_i\ \&\ u_{i+1} \succ u_i \Longrightarrow u_{i+1} = u_i$ — противоречие. □

ЗАМЕЧАНИЕ

Линейно упорядоченное конечное множество содержит ровно один минимальный элемент, а в произвольном конечном частично упорядоченном множестве их может быть несколько.

ТЕОРЕМА 2 *Всякий частичный порядок на конечном множестве может быть дополнен до линейного.*

Доказательство См. следующий подраздел. □

ЗАМЕЧАНИЕ

В данном случае слова «может быть дополнен» означают, что существует отношение линейного порядка, которое является надмножеством заданного отношения частичного порядка.

1.8.3. Алгоритм топологической сортировки

Рассмотрим алгоритм дополнения частичного порядка до линейного на конечном множестве.

Алгоритм 1.12 Алгоритм топологической сортировки

Вход: конечное частично упорядоченное множество U.
Выход: линейно упорядоченное множество W.

```
while U ≠ ∅ do
  m := min(U) { функция min возвращает минимальный элемент }
  yield m { включаем элемент m в множество W }
  U := U − m { элемент m более не рассматривается }
end while
```

Обоснование Всякая процедура, генерирующая объекты с помощью оператора **yield**, определяет линейный порядок на множестве своих результатов. Действительно, линейный порядок — это последовательность, в которой объекты генерируются во время работы процедуры. □

Функция min возвращает минимальный элемент, существование которого гарантируется теоремой 1 подраздела 1.8.2.

Вход: конечное частично упорядоченное множество U.
Выход: минимальный элемент m.

```
select m ∈ U { выбираем любой элемент в качестве кандидата в минимальные }
for u ∈ U do
  if u ≺ m then
    m := u { меняем кандидата в минимальные }
  end if
end for
return m { элемент m минимальный }
```

Обоснование Рассмотрим последовательность элементов, которые присваивались m во время работы min. Эта последовательность не может быть бесконечной, и она линейно упорядочена. Рассмотрим последний элемент этой последовательности. Этот элемент m — минимальный. Действительно, от противного, пусть существует $u \prec m \;\&\; u \neq m$ и u не входит в последовательность. Тогда по транзитивности u меньше любого члена последовательности, и он был бы включен в последовательность в момент своего рассмотрения. □

ЗАМЕЧАНИЕ

Если отношение порядка представлено матрицей, то функцию min можно реализовать, например, так — найти в матрице отношения первую строку, содержащую только нули.

1.8.4. Верхние и нижние границы

Пусть $X \subset M$ — подмножество упорядоченного множества M с отношением порядка $\prec$. Элемент $m \in M$ называется *нижней границей* для подмножества X, если $\forall x \in X \;\; (m \prec x)$. Элемент $m \in M$ называется *верхней границей* для подмножества X, если $\forall x \in X \;\; (x \prec m)$. Верхние и нижние границы не обязаны существовать для любого множества и если существуют, то не всегда единственны. Если существует наибольшая нижняя граница, то она называется *инфимумом* и обозначается $\inf(X)$. Если существует наименьшая верхняя граница, то она называется *супремумом* и обозначается $\sup(X)$.

Пример Рассмотрим множество рациональных чисел $\mathbb{Q}$ с обычным отношением порядка $<$ и его подмножество $X := \{x \in \mathbb{Q} \mid x > 0 \;\&\; x^2 > 2\}$. Тогда $\frac{17}{12} \in X$, а $\frac{17}{13}$ является одной из нижних границ множества X. Инфимума это множество не имеет.

1.8.5. Монотонные функции

Пусть A и B — упорядоченные множества и $f\colon A \to B$. Тогда если

$$a_1 \prec a_2 \;\&\; a_1 \neq a_2 \Longrightarrow f(a_1) \prec f(a_2) \vee f(a_1) = f(a_2),$$

то функция f называется *монотонно возрастающей*. Если

$$a_1 \prec a_2 \;\&\; a_1 \neq a_2 \Longrightarrow f(a_2) \prec f(a_1) \vee f(a_1) = f(a_2),$$

то функция f называется *монотонно убывающей*. Если

$$a_1 \prec a_2 \;\&\; a_1 \neq a_2 \Longrightarrow f(a_1) \prec f(a_2) \;\&\; f(a_1) \neq f(a_2),$$

то функция f называется *строго монотонно возрастающей*. Если

$$a_1 \prec a_2 \;\&\; a_1 \neq a_2 \Longrightarrow f(a_2) \prec f(a_1) \;\&\; f(a_1) \neq f(a_2),$$

то функция f называется *строго монотонно убывающей*. Монотонно возрастающие и убывающие функции называются *монотонными*, а строго монотонно возрастающие и убывающие функции называются *строго монотонными* соответственно. Очевидно, что строго монотонная функция монотонна, но обратное неверно.

Примеры

1. Тождественная функция $y = x$ для чисел является строго монотонно возрастающей, а функция знака числа $\mathrm{sign}(x) \stackrel{\mathrm{Def}}{=}$ **if** $x < 0$ **then** -1 **elseif** $x > 0$ **then** 1 **else** 0 **end if** является монотонно возрастающей.
2. Пусть 2^M — булеан над линейно упорядоченным конечным множеством M, а частичный порядок на булеане — это включение. Тогда функция $\min(X)$, $X \subset M$ из алгоритма 1.12, доставляющая минимальный элемент, является монотонно убывающей, но не строго монотонной.
3. Пусть порядок на булеане 2^M — это собственное включение. Тогда функция, которая сопоставляет подмножеству $X \subset M$ его мощность, $X \mapsto |X|$, является строго монотонно возрастающей.

1.8.6. Вполне упорядоченные множества

Частично упорядоченное множество X называется *вполне упорядоченным*, если любое его непустое подмножество имеет минимальный элемент. В частности, для любых двух элементов $a, b \in X$ один из них обязан быть минимальным в подмножестве $\{a, b\}$, а значит, вполне упорядоченное множество упорядочено линейно.

ЗАМЕЧАНИЕ

Не надо путать вполне упорядоченные множества и множества, на которых определён линейный (или полный) порядок. Линейно упорядоченное множество может не быть вполне упорядоченным.

Примеры

1. Всякое конечное линейно упорядоченное множество вполне упорядочено.
2. Множество натуральных чисел $\mathbb{N}$ с обычным отношением порядка вполне упорядочено.
3. Множество рациональных чисел $\mathbb{Q}$ с обычным отношением порядка не является вполне упорядоченным, так как множество $X := \{x \in \mathbb{Q} \mid x^2 > 2\}$, равно как и само множество $\mathbb{Q}$, не имеет минимального элемента.
4. Множество вещественных чисел $\mathbb{R}$ с обычным отношением порядка не является вполне упорядоченным, так как множество $X := \{x \in \mathbb{R} \mid x > 0\}$, равно как и само множество $\mathbb{R}$, не имеет минимального элемента.

Говорят, что два линейно упорядоченных множества A и B *изоморфны* (обозначение $A \sim B$), если между ними существует взаимно-однозначное соответствие, сохраняющее порядок:

$$A \sim B \stackrel{\mathrm{Def}}{=} |A| = |B| \;\&\; (\forall a_1, a_2 \in A \;\; (a_1 \prec a_2 \;\&\; a_1 \mapsto b_1 \;\&\; a_2 \mapsto b_2 \Longrightarrow b_1 \prec b_2)).$$

Другими словами, два линейно упорядоченных множества A и B изоморфны, если между ними существует строго монотонно возрастающее взаимно-однозначное соответствие.

ЗАМЕЧАНИЕ

Поскольку вполне упорядоченные множества упорядочены линейно, понятие *изоморфизма* применимо и к вполне упорядоченным множествам.

Пример Множества натуральных чисел и чётных чисел с обычным порядком $<$ изоморфны, поскольку соответствие $n \mapsto 2n$ является строго монотонно возрастающим.

1.8.7. Индукция

Важность вполне упорядоченных множеств определяется тем, что для них можно обобщить принцип *индукции*, применяемый обычно для множества натуральных чисел.

ТЕОРЕМА (Индукция по вполне упорядоченному множеству) *Пусть X — вполне упорядоченное множество и x_0 — его минимальный элемент, а P — некоторый предикат, зависящий от элементов X. Тогда если*

$$P(x_0)\ \&\ \forall x_1 \in X\ ((\forall x \in X\ (x \prec x_1 \Longrightarrow P(x))) \Longrightarrow P(x_1)),$$

то

$$\forall x \in X\ (P(x)).$$

Доказательство Обозначим $\overline{P} := \{x \in X \mid \neg P(x)\}$, $\overline{P} \subset X$. Далее от противного. Пусть $\overline{P} \neq \varnothing$. Поскольку X вполне упорядочено, $\overline{P}$ имеет минимальный элемент, обозначим его x_1. Тогда $\forall x \in X\ (x \prec x_1 \Longrightarrow P(x))$ по выбору x_1 и значит $P(x_1)$ по условию теоремы, что противоречит выбору x_1. □

ЗАМЕЧАНИЕ

Обычная математическая индукция соответствует индукции по вполне упорядоченному множеству натуральных чисел $\mathbb{N}$.

Индукция по вполне упорядоченному множеству сильнее обычного принципа математической индукции в силу следующей замечательной теоремы.

ТЕОРЕМА *Любое множество может быть вполне упорядочено.*

Данное утверждение эквивалентно так называемой *аксиоме выбора* в теории множеств и само может быть принято за аксиому. Мы опускаем доказательство эквивалентности этой теоремы аксиоме выбора.

Пример Рассмотрим множество положительных рациональных чисел

$$\mathbb{Q}_+ \stackrel{\text{Def}}{=} \left\{ \frac{m}{n} \mid m, n \in \mathbb{N} \right\},$$

где m и n взаимно просты.

Определим отношение $\prec$ на множестве $\mathbb{Q}_+$ следующим образом:

$$\frac{m_1}{n_1} \prec \frac{m_2}{n_2} \stackrel{\text{Def}}{=} m_1 < m_2 \vee (m_1 = m_2 \,\&\, n_1 < n_2),$$

где $<$ — обычное отношение порядка на $\mathbb{N}$. Нетрудно видеть, что множество $\mathbb{Q}_+$ с отношением $\prec$ является вполне упорядоченным, в то время как с обычным отношением порядка $<$ оно таковым не является, см. 1.8.6.

1.8.8. Алфавит, слово и язык

В этом подразделе вводятся некоторые термины, которые не имеют прямого отношения к теории множеств, но часто используются в различных приложениях и в следующих главах, а потому рассматриваются в первой главе. Пусть задано конечное множество $A = \{a_1, \ldots, a_n\}$, которое называется *алфавитом*. Элементы алфавита называются *буквами*, или *символами*. Обычно отдельные символы обозначаются начальными буквами латинского алфавита. Конечная последовательность букв называется *словом* (в данном алфавите), или *цепочкой*. Буквы в слове линейно упорядочены (слева направо) и могут быть перенумерованы. Одна буква может входить в слово несколько раз, каждый отдельный экземпляр буквы в слове называется *вхождением* буквы в слово. Вхождения могут быть идентифицированы, например, номером буквы в слове. Обычно слова обозначаются начальными буквами греческого алфавита. Множество слов в алфавите A обозначается A^*. Если слово $\alpha = a_1 \ldots a_k \in A^*$, то количество букв в слове называется *длиной* слова: $|\alpha| = |a_1 \ldots a_k| = k$. *Пустое* слово обозначается ε, $|\varepsilon| \stackrel{\text{Def}}{=} 0$, $\varepsilon \notin A$, но $\varepsilon \in A^*$. Если пустое слово исключается из рассмотрения, то множество слов в алфавите A обозначается A^+. Таким образом, $A^* = A^+ + \varepsilon$, $A^+ = A^* - \varepsilon$. Для слов определена операция *конкатенации*, или *сцепления*. Конкатенацией слов α и β является слово $\alpha\beta$, полученное выписыванием подряд сначала всех букв слова α, а потом всех букв слова β. Обычно операция сцепления никак не обозначается. Если $\alpha = \alpha_1\alpha_2$, то α_1 называется *началом*, или *префиксом*, слова α, а α_2 — *окончанием*, или *постфиксом*, слова α. Если при этом $\alpha_1 \neq \varepsilon$ (соответственно, $\alpha_2 \neq \varepsilon$), то α_1 (соответственно, α_2) называется *собственным началом* (соответственно, *собственным окончанием*) слова α. Произвольное множество L слов в некотором алфавите называется *языком* в этом алфавите, $L \subset A^*$.

Комментарии

Основные сведения, изложенные в этой главе, можно найти в *любом* учебнике по (дискретной) математике. Алгоритм 1.1 тривиален. Алгоритмы 1.2 и 1.3 описаны в [8]. Алгоритмы 1.4–1.7 — общеизвестный программистский фольклор. Автору неизвестны другие публикации алгоритмов 1.8–1.10, несмотря на

их очевидность. Алгоритм 1.11 описан во многих источниках, например, в [2]. Алгоритм 1.12 описан в фундаментальной книге [14], которая входит в «золотой список» обязательного чтения любого программиста.

Упражнения

1.1. Существует ли множество всех множеств?

1.2. Доказать, что $A \cup B = (A \cap B) \cup (A \cap \overline{B}) \cup (\overline{A} \cap B)$.

1.3. Доказать, что $Q(2^k + m) = Q(2^k - m)$ для $0 \leqslant m \leqslant 2^k - 1$, где Q — функция из алгоритма 1.3.

1.4. Пусть $Q \stackrel{\text{Def}}{=} \{(m, n) \mid m, n \in \mathbb{N} \;\&\; m = n^2\}$. Какими свойствами обладает отношение Q?

1.5. Доказать вторую теорему из подраздела 1.5.1.

1.6. Доказать теорему из подраздела 1.6.3.

1.7. Доказать, что если $\equiv$ — отношение эквивалентности на конечном множестве M и $|M| = |M/\equiv|$, то $\forall x \in M \;\; (|[x]_{\equiv}| = 1)$.

1.8. Пусть A — линейно упорядоченное множество с отношением порядка R. Введем отношение $\vec{R}$ на множестве кортежей элементов из A следующим образом:

$$(a_1, \ldots, a_m)\vec{R}(b_1, \ldots, b_n) \stackrel{\text{Def}}{=} (m \leqslant n \;\&\; \forall i \in 1..m \;\; (a_i = b_i)) \vee$$
$$\vee \; (\exists i \in 1..m \;\; ((a_i R b_i \;\&\; \forall j \in 1..i-1 \;\; (a_i = b_j)))).$$

Такое отношение называется *лексикографическим*, или *алфавитным*, порядком. Доказать, что алфавитный порядок есть линейный порядок на множестве кортежей $A^+ \stackrel{\text{Def}}{=} \bigcup_{i=1}^{\infty} A^i$.

Глава 2 Алгебраические структуры

Алгебраические методы находят самое широкое применение при формализации различных предметных областей. Грубо говоря, при построении модели предметной области всё начинается с введения подходящих обозначений для операций и отношений с последующим исследованием их свойств. Владение алгебраической терминологией, таким образом, входит в арсенал средств, необходимых для абстрактного моделирования, предшествующего практическому программированию задач конкретной предметной области. Материал этой главы, помимо введения в терминологию общей алгебры, содержит некоторое количество примеров конкретных алгебраических структур. Вначале бегло рассматриваются классические структуры, которые обычно включаются в курсы общей алгебры, а затем обсуждаются некоторые более специальные структуры, наиболее часто применяемые в конкретных программных построениях. Особого внимания заслуживает понятие матроида, обсуждаемое в этой главе, поскольку матроиды тесно связаны с важнейшим классом эффективных алгоритмов, называемых *жадными*.

2.1. Алгебры и морфизмы

Словом «алгебра» называют, вообще говоря, не только раздел математики, но и один из конкретных объектов, изучаемых в этом разделе. Поскольку «алгебры» в узком смысле здесь не рассматриваются, для краткости и без риска возникновения недоразумений слово «алгебра» в этом учебнике используется как родовое понятие для обозначения разнообразных алгебраических структур.

2.1.1. Операции и их носитель

Всюду определённая (тотальная) функция $\varphi\colon M^n \to M$ называется *n-арной (n-местной) операцией* на M. Если операция φ — бинарная ($\varphi\colon M \times M \to M$), то будем писать в инфиксной форме $a\varphi b$ вместо $\varphi(a, b)$ или $a * b$, где $*$ — знак операции. Множество M вместе с набором операций $\Sigma = \{\varphi_1, \ldots, \varphi_m\}$, $\varphi_i\colon M^{n_i} \to M$, где n_i — арность операции φ_i, называется *алгебраической структурой, универсальной алгеброй* или просто *алгеброй*. При этом множество M называется *основным (несущим)* множеством или *основой (носителем)*; вектор арностей $(n_1, \ldots, n_m)$

называется *типом*; множество операций Σ называется *сигнатурой*. Запись: $\langle M;\Sigma\rangle$ или $\langle M;\varphi_1,\ldots,\varphi_m\rangle$. Операции φ_i *конечноместны*, сигнатура Σ конечна. Носитель не обязательно конечен, но не пуст.

ЗАМЕЧАНИЕ

Далее для обозначения алгебры везде, где это возможно, используется прописная рукописная буква, а для обозначения её носителя — соответствующая обычная прописная буква: $\mathcal{A} = \langle A;\Sigma\rangle$. Такое соглашение позволяет использовать вольности в обозначениях, не вводя явно две буквы для алгебры и для носителя, а подразумевая одну из них по умолчанию. Например, выражение «алгебра A» означает алгебру $\mathcal{A}$ с носителем A и сигнатурой, которая ясна из контекста, а запись $a \in \mathcal{A}$ означает элемент a из носителя A алгебры $\mathcal{A}$.

Если в качестве φ_i допускаются не только функции, но и отношения, то множество M вместе с набором операций и отношений называется *моделью*. В приложениях обычно используется следующее обобщение понятия алгебры. Пусть $M = \{M_1,\ldots,M_n\}$ — множество *основ*, $\Sigma = \{\varphi_1,\ldots,\varphi_m\}$ — сигнатура, причём $\varphi_i\colon M_{i_1}\times\cdots\times M_{i_{n_i}} \to M_j$. Тогда $\langle M;\Sigma\rangle$ называется *многоосновной* алгеброй. Другими словами, многоосновная алгебра имеет несколько носителей, а операция сигнатуры действует из прямого произведения некоторых носителей в некоторый носитель.

2.1.2. Замыкания и подалгебры

Подмножество носителя $X \subset M$ называется *замкнутым* относительно операции φ, если

$$\forall\, x_1,\ldots,x_n \in X\ \ (\varphi(x_1,\ldots,x_n) \in X)\,.$$

Если X замкнуто относительно всех $\varphi \in \Sigma$, то $\langle X;\Sigma_X\rangle$ называется *подалгеброй* $\langle M;\Sigma\rangle$, где $\Sigma_X = \{\varphi_i^X\}$, $\varphi_i^X = \varphi_i\big|_{X^k}$, $k = n_i$.

Примеры

1. Алгебра $\langle\mathbb{R};+,\cdot\rangle$ — *поле вещественных чисел*. Тип — $(2,2)$. Все конечные подмножества, кроме $\{0\}$, не замкнуты относительно сложения, и все конечные подмножества, кроме $\{0\}$ и $\{0,1\}$, не замкнуты относительно умножения. *Поле рациональных чисел* $\langle\mathbb{Q};+,\cdot\rangle$ образует подалгебру. *Кольцо целых чисел* $\langle\mathbb{Z};+,\cdot\rangle$ образует подалгебру алгебры рациональных и, соответственно, вещественных чисел.
2. Алгебра $\langle 2^M;\cup,\cap,^-\rangle$ — *алгебра подмножеств* над множеством M. Тип — $(2,2,1)$. При этом $\forall X \subset M\ \ \big(\langle 2^X;\cup,\cap,^-\rangle\big)$ — её подалгебра.
3. Алгебра гладких функций $\big\langle\{f \mid f\colon\mathbb{R}\to\mathbb{R}\}\,;\frac{d}{dx}\big\rangle$, где $\frac{d}{dx}$ — операция дифференцирования. Множество полиномов одной переменной x образует подалгебру, которая обозначается $\mathbb{R}[x]$.

ТЕОРЕМА *Непустое пересечение подалгебр некоторой алгебры образует подалгебру этой же алгебры.*

Доказательство Пусть $\langle X_i; \Sigma_{X_i}\rangle$ — подалгебра $\langle M; \Sigma\rangle$. Тогда

$$\forall i \left(\forall j \left(\varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in X_i\right)\right) \Longrightarrow \forall j \left(\varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in \bigcap X_i\right). \quad \square$$

Замыканием множества $X \subset M$ относительно сигнатуры Σ (обозначается $[X]_\Sigma$) называется множество всех элементов (включая сами элементы множества X), которые можно получить, применяя операции из Σ. Если сигнатура подразумевается, её можно не указывать.

Пример В кольце целых чисел $\langle \mathbb{Z}; +, \cdot\rangle$ замыканием числа 2 являются чётные числа, то есть $[\{2\}] = \{n \in \mathbb{Z} \mid n = 2k \ \&\ k \in \mathbb{Z}\}$.

Свойства замыкания:

1. $X \subset Y \Longrightarrow [X] \subset [Y]$.
2. $X \subset [X]$.
3. $[[X]] = [X]$.
4. $[X] \cup [Y] \subset [X \cup Y]$.

Пусть $\mathcal{A} = \langle A; \Sigma\rangle$ — некоторая алгебра и $X_1, \ldots, X_n \subset A$ — некоторые подмножества носителя, а $\varphi \in \Sigma$ — одна из операций алгебры. Тогда используется следующее соглашение об обозначениях:

$$\varphi(X_1, \ldots, X_n) \stackrel{\text{Def}}{=} \{\varphi(x_1, \ldots, x_n) \mid x_1 \in X_1 \ \&\ \ldots \ \&\ x_n \in X_n\},$$

то есть алгебраические операции можно применять не только к отдельным элементам, но и к множествам (подмножествам носителя), получая, соответственно, не отдельные элементы, а множества (подмножества носителя), подобно тому, как это определено в подразделе 1.6.3.

2.1.3. Система образующих

Множество $M' \subset M$ называется *системой образующих* алгебры $\langle M; \Sigma\rangle$, если $[M']_\Sigma = M$. Если алгебра имеет конечную систему образующих, то алгебра называется *конечно-порождённой*. Бесконечные алгебры могут иметь конечные системы образующих.

Пример Алгебра *натуральных чисел* — $\langle \mathbb{N}; +\rangle$ — имеет конечную систему образующих $1 \in \mathbb{N}$.

Пусть заданы набор функциональных символов $\Phi = \{\varphi_1, \ldots, \varphi_m\}$, служащих обозначениями функций некоторой сигнатуры Σ типа $N = (n_1, \ldots, n_m)$, и множество переменных $V = \{x_1, x_2, \ldots\}$. Определим множество *термов* T индуктивным образом:

1. $V \subset T$;
2. $t_1, \ldots, t_{n_i} \in T \;\&\; \varphi_i \in \Phi \Longrightarrow \varphi_i(t_1, \ldots, t_{n_i}) \in T$.

Алгебра $\langle T; \Phi \rangle$ называется *свободной алгеброй термов*. Носителем этой алгебры является множество термов, то есть формальных выражений, построенных с помощью знаков операций сигнатуры Σ. Заметим, что множество V является системой образующих свободной алгебры термов.

Пример Если $V = \{x\}$ и $\Sigma = \{+, \cdot\}$, то свободная алгебра термов — это множество всех выражений, которые можно построить из переменной x с помощью операций сложения и умножения, то есть алгебра полиномов от одной переменной с натуральными коэффициентами.

ОТСТУПЛЕНИЕ

Алгебры термов используются в программировании для определения абстрактных типов данных.

2.1.4. Свойства операций

Некоторые часто встречающиеся свойства операций имеют специальные названия. Пусть задана алгебра $\langle M; \Sigma \rangle$ и $a, b, c \in M$; $\circ, \diamond \in \Sigma$; $\circ, \diamond\colon M \times M \to M$. Тогда:

1. *Ассоциативность*: $(a \circ b) \circ c = a \circ (b \circ c)$.
2. *Коммутативность*: $a \circ b = b \circ a$.
3. *Дистрибутивность* $\diamond$ относительно $\circ$ слева: $a \diamond (b \circ c) = (a \diamond b) \circ (a \diamond c)$.
4. *Дистрибутивность* $\diamond$ относительно $\circ$ справа: $(a \circ b) \diamond c = (a \diamond c) \circ (b \diamond c)$.
5. *Поглощение* ($\diamond$ поглощает $\circ$): $(a \circ b) \diamond a = a$.
6. *Идемпотентность*: $a \circ a = a$.

Примеры

1. Ассоциативные операции: сложение и умножение чисел, объединение и пересечение множеств, композиция отношений. Неассоциативные операции: возведение чисел в степень, вычитание множеств.
2. Коммутативные операции: сложение и умножение чисел, объединение и пересечение множеств. Некоммутативные операции: умножение матриц, композиция отношений, возведение в степень.
3. Дистрибутивные операции: умножение относительно сложения чисел. Недистрибутивные операции: возведение в степень дистрибутивно относительно умножения справа, но не слева: $\left((ab)^c = a^c b^c, a^{bc} \neq a^b a^c\right)$.

4. Пересечение поглощает объединение, объединение поглощает пересечение. Сложение и умножение не поглощают друг друга.
5. Идемпотентные операции: наибольший общий делитель натуральных чисел, объединение и пересечение множеств. Неидемпотентные операции: сложение и умножение чисел.

2.1.5. Гомоморфизмы

Понятие гомоморфизма является одним из ключевых понятий алгебры. Каждая алгебраическая структура определённым образом выделяет класс «разумных» отображений между объектами с данной структурой, согласованных с операциями этой структуры. Алгебры с различными типами имеют различное *строение*. Гомоморфизм определяется для алгебр одного типа. Пусть $\mathcal{A} = \langle A; \varphi_1, \ldots, \varphi_m \rangle$ и $\mathcal{B} = \langle B; \psi_1, \ldots, \psi_m \rangle$ — две алгебры одного типа. Если существует функция $f\colon A \to B$, такая, что

$$\forall i \in 1..m \ \big(f(\varphi_i(a_1, \ldots, a_n)) = \psi_i(f(a_1), \ldots, f(a_n))\big),$$

то говорят, что f — *гомоморфизм* из $\mathcal{A}$ в $\mathcal{B}$.

ЗАМЕЧАНИЕ

Образно говорят, что гомоморфизм «уважает» операции.

Пусть $\mathcal{A} = \langle A; \varphi \rangle$, $\mathcal{B} = \langle B; \psi \rangle$, тип $= (1)$ и $f\colon A \to B$. Действие функций φ, ψ, f можно изобразить с помощью следующей диаграммы:

$$\begin{array}{ccc} A & \xrightarrow{\varphi} & A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ B & \xrightarrow{\psi} & B \end{array}$$

Пусть f — гомоморфизм. Тогда если взять конкретное значение $a \in A$ и двигаться по двум различным путям на диаграмме, то получится один и тот же элемент $b \in B$ (так как $f(\varphi(a)) = \psi(f(a))$). В таком случае диаграмма называется *коммутативной*. Коммутативной диаграмма называется потому, что условие гомоморфизма можно переписать так:

$$f \circ \varphi = \psi \circ f,$$

где $\circ$ — суперпозиция функций.

Пример Пусть $\mathcal{A} = \langle \mathbb{N}; + \rangle$, $\mathcal{B} = \langle \mathbb{N}_{10}; +_{10} \rangle$, где $\mathbb{N}_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, а $+_{10}$ — сложение по модулю 10. Тогда $f : a \mapsto (a \mod 10)$ — гомоморфизм из $\mathcal{A}$ в $\mathcal{B}$.

Гомоморфизмы, обладающие дополнительными свойствами, имеют специальные названия:

- Гомоморфизм, который является инъекцией, называется *мономорфизмом*.
- Гомоморфизм, который является сюръекцией, называется *эпиморфизмом*.

- Гомоморфизм, который является биекцией, называется *изоморфизмом*. Другими словами, изоморфизм является одновременно мономорфизмом и эпиморфизмом.
- Если $A = B$, то гомоморфизм называется *эндоморфизмом*, а изоморфизм называется *автоморфизмом*.

2.1.6. Изоморфизмы

Пусть $\mathcal{A} = \langle A; \varphi_1, \ldots, \varphi_m \rangle$ и $\mathcal{B} = \langle B; \psi_1, \ldots, \psi_m \rangle$ — две алгебры одного типа и $f\colon A \to B$ — изоморфизм.

ТЕОРЕМА 1 *Если $f\colon A \to B$ — изоморфизм, то $f^{-1}\colon B \to A$ — тоже изоморфизм.*

Доказательство Рассмотрим произвольную операцию φ из сигнатуры алгебры $\mathcal{A}$ и соответствующую ей операцию ψ из сигнатуры алгебры $\mathcal{B}$. Пусть вместимость этих операций n. Рассмотрим произвольные элементы $a_1, \ldots, a_n \in A$. Обозначим $b_1 := f(a_1), \ldots, b_n := f(a_n)$, где $b_1, \ldots, b_n \in B$. Поскольку f — биекция, имеем $a_1 = f^{-1}(b_1), \ldots, a_n = f^{-1}(b_n)$. Тогда

$$f^{-1}\big(\psi(b_1, \ldots, b_n)\big) = f^{-1}\Big(\psi\big(f(a_1), \ldots, f(a_n)\big)\Big) = f^{-1}\Big(f\big(\varphi(a_1, \ldots, a_n)\big)\Big) =$$
$$= \varphi(a_1, \ldots, a_n) = \varphi\big(f^{-1}(b_1), \ldots, f^{-1}(b_n)\big). \qquad \square$$

Если $f\colon A \to B$ — изоморфизм, то алгебры $\mathcal{A}$ и $\mathcal{B}$ называют *изоморфными* и обозначают так: $\mathcal{A} \overset{f}{\sim} \mathcal{B}$. Если f ясно из контекста или просто неважно в конкретном рассуждении, то пишут $\mathcal{A} \sim \mathcal{B}$.

ТЕОРЕМА 2 *Отношение изоморфизма на множестве однотипных алгебр является эквивалентностью.*

Доказательство

[Рефлексивность] $\mathcal{A} \overset{f}{\sim} \mathcal{A}$, где f — тождественное отображение.

[Симметричность] $\mathcal{A} \overset{f}{\sim} \mathcal{B} \Longrightarrow \mathcal{B} \overset{f^{-1}}{\sim} \mathcal{A}$.

[Транзитивность] $\mathcal{A} \overset{f}{\sim} \mathcal{B}\ \&\ \mathcal{B} \overset{g}{\sim} \mathcal{G} \Longrightarrow \mathcal{A} \overset{g \circ f}{\sim} \mathcal{G}$. $\square$

Примеры

1. Пусть $\mathcal{A} = \langle \mathbb{N}; + \rangle$, $\mathcal{B} = \langle \{n \mid n = 2k, k \in \mathbb{N}\}; + \rangle$ — *чётные* числа. Тогда $\mathcal{A} \overset{\times 2}{\sim} \mathcal{B}$.
2. $\mathcal{A} = \langle 2^M; \cap, \cup \rangle \overset{f}{\sim} \mathcal{B} = \langle 2^M; \cup, \cap \rangle$, $f(X) = \overline{X}$.
3. $\mathcal{A} = \langle \mathbb{R}_+; \cdot \rangle \overset{\log_a}{\sim} \mathcal{B} = \langle \mathbb{R}; + \rangle$.

Понятие изоморфизма является одним из центральных понятий, оправдывающих применимость алгебраических методов в различных областях. Действительно, пусть $\mathcal{A} = \langle A; \Sigma_\varphi \rangle$, $\mathcal{B} = \langle B; \Sigma_\psi \rangle$ и $\mathcal{A} \overset{f}{\sim} \mathcal{B}$. Пусть в алгебре $\mathcal{A}$ установлено свойство $\Phi_1 = \Phi_2$, где Φ_1 и Φ_2 — некоторые формулы в сигнатуре Σ_φ. Поскольку алгебры $\mathcal{A}$ и $\mathcal{B}$ изоморфны, отсюда немедленно следует, что в алгебре $\mathcal{B}$ справедливо свойство $\Psi_1 = \Psi_2$, где Ψ_1 и Ψ_2 — формулы, полученные из формул Φ_1 и Φ_2 заменой операций из сигнатуры Σ_φ соответствующими операциями из сигнатуры Σ_ψ. Таким образом, достаточно установить некоторое свойство в одной алгебре, и оно автоматически распространяется на все изоморфные алгебры. Алгебраические структуры принято рассматривать *с точностью до изоморфизма*.

Понятие изоморфизма применяется не только в алгебре, но и практически во всех областях математики. Например, в этом учебнике рассматривается изоморфизм множеств (1.2.2), линейно упорядоченных множеств (1.8.6), графов (7.1.6).

ОТСТУПЛЕНИЕ

Термин «морфизм», вынесенный в название раздела, используется как собирательное понятие, подобно тому, как в объектно-ориентированном программировании используется термин «абстрактный класс». Напомним, что абстрактный класс — это класс, не могущий иметь непосредственных экземпляров, но являющийся обобщением (предком) некоторых конкретных классов, которые могут иметь непосредственные экземпляры.

2.2. Алгебры с одной операцией

Естественно начать изучение алгебраических структур с наиболее простых. Самой простой структурой является алгебра с одной унарной операцией. Здесь этот случай не рассматривается, хотя он достаточно содержателен. Следующим по порядку сложности является случай алгебры с одной бинарной операцией $*\colon M \times M \to M$, который и рассматривается в этом разделе.

2.2.1. Полугруппы

Полугруппа — это алгебра с одной ассоциативной бинарной операцией:

$$a * (b * c) = (a * b) * c.$$

Примеры

1. Множество непустых слов A^+ в алфавите A образует полугруппу относительно операции *конкатенации* (см. 1.8.8).
2. Всякое множество тотальных функций одного аргумента $\{f \mid f\colon M \to M\}$, замкнутое относительно суперпозиции, является полугруппой.

Если в полугруппе существует система образующих, состоящая из одного элемента, то такая полугруппа называется *циклической*.

Пример $\langle \mathbb{N}; + \rangle$ является циклической полугруппой, поскольку $\{1\}$ является системой образующих.

2.2.2. Определяющие соотношения

Пусть $P = \langle M; * \rangle$ — полугруппа с конечной системой образующих $A, A \in M, A = \{a_1, \ldots, a_n\}$. Тогда

$$\forall x \in M \ (\exists y_1, \ldots, y_k \in A \ (x = y_1 * \ldots * y_k)).$$

Если опустить обозначение операции $*$, то всякий элемент $a \in M$ можно представить как слово α в алфавите A, то есть $M \subset A^+$. Обозначим через $\bar{\alpha}$ элемент, определяемый словом α. Слова α и β могут быть различными, но соответствующие им элементы $\bar{\alpha}$ и $\bar{\beta}$ могут быть равны: $\bar{\alpha} = \bar{\beta}$, $\alpha, \beta \in A^+$, $\bar{\alpha}, \bar{\beta} \in M$. Такие равенства называются *определяющими соотношениями*. Если в полугруппе нет определяющих соотношений и все различные слова, составленные из образующих, суть различные элементы носителя, то полугруппа называется *свободной*. Всякая конечно-порождённая полугруппа может быть получена из свободной введением определяющих соотношений. Пусть в конечно-порождённой полугруппе некоторый элемент $\bar{\alpha}$ определяется словом α, причём $\alpha = \beta\gamma\delta$ и задано определяющее соотношение $\gamma = \sigma$. Тогда вхождение слова γ в слово α можно заменить словом σ, причём полученное слово $\beta\sigma\delta$ будет определять тот же самый элемент $\bar{\alpha}$. Такое преобразование слова будем называть *применением* определяющего соотношения. Два слова в алфавите A считаются равными, если одно из другого получается применением определяющих соотношений, то есть слова равны, если равны определяемые ими элементы. Отношение равенства слов в полугруппе с определяющими соотношениями является отношением эквивалентности. Классы эквивалентности по этому отношению соответствуют элементам полугруппы.

Примеры

1. В полугруппе $\langle \mathbb{N}; + \rangle$ имеется конечная система образующих $\{1\}$. Другими словами, каждое натуральное число можно представить как последовательность знаков 1. Очевидно, что различные слова в алфавите $\{1\}$ суть различные элементы носителя, то есть эта полугруппа свободна.
2. Пусть полугруппа $\mathcal{P}$ задана системой двух образующих $\{a, b\}$ и двумя определяющими соотношениями, $aa = a$ и $bb = b$. Следующий элементарный алгоритм определяет, равны ли два слова, α и β, в полугруппе $\mathcal{P}$.

   Вход: входные слова α : **array** $[1..s]$ **of** $\{a, b\}$ и β : **array** $[1..t]$ **of** $\{a, b\}$;
   Выход: значение выражения $\alpha = \beta$ в полугруппе $\mathcal{P}$.

   ```
   if α[1] ≠ β[1] then
     return false { первые буквы не совпадают }
   end if
   N_α := 0 { счётчик изменений буквы в α }
   for i from 2 to s do
     if α[i] ≠ α[i − 1] then
   ```

```
      N_α := N_α + 1 { буква изменилась }
    end if
  end for
  N_β := 0 { счётчик изменений буквы в β }
  for i from 2 to t do
    if β[i] ≠ β[i − 1] then
      N_β := N_β + 1 { буква изменилась }
    end if
  end for
  return N_α = N_β { слова равны, если значения счётчиков одинаковы }
```

В предыдущем примере определяющие соотношения таковы, что равенство любых слов как элементов легко проверяется. Однако это не всегда так.

ТЕОРЕМА (Марков[1]–Пост[2]) *Существует полугруппа, в которой проблема распознавания равенства слов алгоритмически неразрешима.*

Доказательство (Без доказательства). □

ЗАМЕЧАНИЕ

Термин «алгоритмическая неразрешимость» принадлежит теории алгоритмов, которая не рассматривается в этом учебнике, а потому строгое определение этого термина здесь не приводится. Неформально можно сказать, что проблема называется *алгоритмически неразрешимой*, если не существует программы (алгоритма) её решения. Здесь словосочетание «не существует» означает не то, что программа ещё не составлена, а то, что требуемая программа не может быть составлена в принципе. В теории алгоритмов строго установлено, что алгоритмически неразрешимые проблемы существуют. Приведём простое наблюдение, которое не является строгим доказательством последнего утверждения, но может служить косвенным намёком на причины существования алгоритмически неразрешимых проблем. Пусть в конечном алфавите A задано слово α, $\alpha \in A^*$ и язык L, $L \subset A^*$. Проблема: определить, принадлежит ли слово α языку L? Обозначим через $P_L(\alpha)$ программу, которая для заданного слова $\alpha \in A^*$ выдаёт 1, если $\alpha \in L$, и выдаёт 0 в противном случае. Множество всех программ не более чем счётно (просто потому, что всякая программа — это слово в конечном алфавите). В то же время множество всех языков несчётно, потому что это множество всех подмножеств счётного множества. Таким образом, заведомо существуют языки, для которых не существует программы, распознающей все слова этого языка. Приведённое наблюдение не является доказательством, потому что мы не определили, как именно задаются объекты α, L и $P_L(\alpha)$.

Пример Г. С. Цейтин[3] нашел легко описываемый пример полугруппы, в которой проблема равенства слов алгоритмически неразрешима. В этой полугруппе всего пять образующих $\{a, b, c, d, e\}$ и семь определяющих соотношений:

$$ac = ca; \quad ad = da; \quad bc = cb; \quad bd = db; \quad abac = abace; \quad eca = ae; \quad edb = be.$$

[1] Андрей Андреевич Марков (1903–1979).

[2] Эмиль Леон Пост (1897–1954).

[3] Григорий Самуилович Цейтин (р. 1937).

Тем не менее невозможно составить программу, которая бы для любых заданных слов в алфавите $\{a, b, c, d, e\}$ проверяла, можно ли преобразовать одно слово в другое применением указанных определяющих соотношений.

ОТСТУПЛЕНИЕ

Некоторые программисты полагают, что если в условиях задачи всё дискретно и конечно, то для решения такой задачи программу можно составить в любом случае (используя метод «полного перебора»). Предшествующие теорема и пример показывают, что это мнение ошибочно.

2.2.3. Моноиды

Моноид — это полугруппа с *единицей*:

$$\exists e \ (\forall a \ (a * e = e * a = a)).$$

ЗАМЕЧАНИЕ

Единицу также называют *нейтральным элементом*.

Примеры

1. Множество слов A^* в алфавите A вместе с пустым словом ε образует моноид.
2. Пусть T — множество термов над множеством переменных V и сигнатурой Σ. *Подстановкой*, или *заменой переменных*, называется множество пар

   $$\sigma = \{t_i//v_i\}_{i \in I},$$

   где t_i — термы, а v_i — переменные. Результатом применения подстановки σ к терму t (обозначается $t\sigma$) называется терм, который получается заменой *всех* вхождений переменных v_i соответствующими термами t_i. *Композицией* подстановок $\sigma_1 = \{t_i//v_i\}_{i \in I}$ и $\sigma_2 = \{t_j//v_j\}_{j \in J}$ называется подстановка $\sigma_1 \circ \sigma_2 \stackrel{\text{Def}}{=} \{t_k//v_k\}_{k \in K}$, где $K = I \cup J$, а

   $$t_k = \begin{cases} t_i\sigma_2, & \text{если } k \in I, \\ t_j, & \text{если } k \notin I. \end{cases}$$

   Множество подстановок образует моноид относительно композиции, причём тождественная подстановка $\{v_i//v_i\}$ является единицей.

ТЕОРЕМА 1 *Единица моноида единственна.*

Доказательство Пусть существуют две единицы: e_1, e_2. Тогда $e_1 * e_2 = e_1$ & $e_1 * e_2 = e_2 \Longrightarrow e_1 = e_2$. □

ТЕОРЕМА 2 *Всякий моноид над M изоморфен некоторому моноиду преобразований над M.*

Доказательство Пусть $\mathcal{M} = \langle M; * \rangle$ — моноид над $M = \{e, a, b, c, \dots\}$. Построим $\mathcal{F} = \langle F; \circ \rangle$ — моноид преобразований над M, где

$$F := \{f_m : M \to M \mid f_m(x) := x * m\}_{m \in M},$$

а $\circ$ — суперпозиция функций, $h : M \to F$, $h(m) := f_m$. Тогда $\mathcal{F}$ — моноид, поскольку суперпозиция функций ассоциативна, f_e — тождественная функция (так как $f_e(x) = x * e = x$) и F замкнуто относительно $\circ$, так как $f_a \circ f_b = f_{a*b}$: $f_{a*b}(x) = x * (a * b) = (x * a) * b = f_a(x) * b = f_b(f_a(x))$. Далее, h — гомоморфизм, так как $h(a * b) = f_{a*b} = f_a \circ f_b = h(a) \circ h(b)$. И наконец, h — биекция, поскольку h — сюръекция по построению, и h — инъекция (так как $(f_a(e) = e * a = a \;\&\; f_b(e) = e * b = b) \Longrightarrow (a \neq b \Longrightarrow f_a \neq f_b)$). □

2.2.4. Группы

Теория групп, некоторые положения которой рассматриваются в этом подразделе, является ярчайшим примером мощи алгебраических методов, позволяющих получить из немногих базовых понятий и аксиом множество важнейших следствий. *Группа* — это моноид, в котором

$$\forall a \; \left(\exists a^{-1} \; \left(a * a^{-1} = a^{-1} * a = e\right)\right).$$

Элемент a^{-1} называется *обратным*, а операция $*$ называется умножением.

ЗАМЕЧАНИЕ

Так как операция умножения в группе не обязательно коммутативна, то, вообще говоря, свойства $a^{-1} * a = e$ и $a * a^{-1} = e$ не равносильны. Можно было бы различать обратные слева и обратные справа элементы. Введенная аксиома требует существования двустороннего обратного элемента. Можно показать, что она следует из более слабых аксиом существования левой единицы и левого обратного элемента.

Мощность носителя G группы $\mathcal{G}$ называется *порядком группы* и обозначается $|G|$.

Примеры

1. Множество невырожденных квадратных матриц порядка n образует группу относительно операции умножения матриц. Единицей группы является единичная матрица (единицы на главной диагонали, остальные элементы равны нулю). Обратным элементом является обратная матрица.
2. Множество перестановок на множестве M, то есть множество взаимно однозначных функций $f : M \to M$, является группой относительно операции суперпозиции. Единицей группы является тождественная функция, а обратным элементом — обратная функция.

ТЕОРЕМА 1 *Обратный элемент единственен.*

Доказательство Пусть $a * a^{-1} = a^{-1} * a = e \;\&\; a * b = b * a = e$. Тогда $a^{-1} = a^{-1} * e = a^{-1} * (a * b) = (a^{-1} * a) * b = e * b = b$. □

ТЕОРЕМА 2 *В любой группе выполняются следующие соотношения:*

1. $(a * b)^{-1} = b^{-1} * a^{-1}$.
2. $a * b = a * c \Longrightarrow b = c$.
3. $b * a = c * a \Longrightarrow b = c$.
4. $(a^{-1})^{-1} = a$.

Доказательство

[1] $(a * b) * (b^{-1} * a^{-1}) = a * (b * b^{-1}) * a^{-1} = a * e * a^{-1} = a * a^{-1} = e$.

[2] $b = e * b = (a^{-1} * a) * b = a^{-1} * (a * b) = a^{-1} * (a * c) = (a^{-1} * a) * c = e * c = c$.

[3] $b = b * e = b * (a * a^{-1}) = (b * a) * a^{-1} = (c * a) * a^{-1} = c * (a * a^{-1}) = c * e = c$.

[4] $a^{-1} * a = e$. □

ТЕОРЕМА 3 *В группе можно однозначно решить уравнение* $a * x = b$ *(решение:* $x = a^{-1} * b$*).*

Доказательство $a * x = b \Longrightarrow a^{-1} * (a * x) = a^{-1} * b \Longrightarrow (a^{-1} * a) * x = a^{-1} * b \Longrightarrow$ $\Longrightarrow e * x = a^{-1} * b \Longrightarrow x = a^{-1} * b$. □

Коммутативная группа, то есть группа, в которой

$$\forall a, b \;\; (a * b = b * a),$$

называется *абелевой*[1]. В абелевых группах обычно приняты следующие обозначения: групповая операция обозначается $+$, обратный элемент к a обозначается $-a$, единица группы обозначается 0 и называется *нулем* или *нейтральным элементом*.

Примеры

1. $\langle \mathbb{Z}; + \rangle$ — множество целых чисел образует абелеву группу относительно сложения. Нейтральным элементом группы является число 0. Обратным элементом является число с противоположным знаком: $x^{-1} \stackrel{\text{Def}}{=} -x$.
2. $\langle \mathbb{Q}_+; \cdot \rangle$ — множество положительных рациональных чисел образует абелеву группу относительно умножения. Нейтральным элементом группы является число 1. Обратным элементом является обратное число: $(m/n)^{-1} \stackrel{\text{Def}}{=} n/m$.

[1] Нильс Хенрик Абель (1802–1829).

3. $\langle 2^M; \Delta \rangle$ — булеан образует абелеву группу относительно симметрической разности. Нейтральным элементом группы является пустое множество $\varnothing$. Обратным элементом к элементу x является он сам: $x^{-1} \stackrel{\text{Def}}{=} x$.

2.2.5. Группа перестановок

В этом подразделе рассматривается одна из важнейших групп, называемая группой *перестановок*, или *симметрической* группой. Биективная функция $f\colon X \to X$ называется *перестановкой* множества X.

ЗАМЕЧАНИЕ

Если множество X конечно ($|X| = n$), то, не ограничивая общности, можно считать, что $X = 1..n$. В этом случае перестановку $f\colon 1..n \to 1..n$ удобно задавать таблицей из двух строк. В первой строке — значения аргументов, во второй — соответствующие значения функции. Такая таблица называется *подстановкой*. В сущности, перестановка и подстановка — синонимы.

Пример

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 1 & 4 & 3 \end{vmatrix}, \qquad g = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 2 & 3 & 5 \end{vmatrix}.$$

Произведением перестановок f и g (обозначается fg) называется их суперпозиция $g \circ f$.

Пример

$$fg = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 3 & 2 \end{vmatrix}.$$

Тождественная перестановка — это перестановка e, такая, что $e(x) = x$.

Пример

$$e = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{vmatrix}.$$

Обратная перестановка — это обратная функция, которая всегда существует, поскольку перестановка является биекцией.

ЗАМЕЧАНИЕ

Таблицу обратной подстановки можно получить, если просто поменять местами строки таблицы исходной подстановки.

Пример

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{vmatrix}, \qquad f^{-1} = \begin{vmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{vmatrix} = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{vmatrix}.$$

Таким образом, поскольку суперпозиция функций ассоциативна, а единичный и обратный элементы существуют, множество перестановок n-элементного множества образует группу относительно операции суперпозиции. Эта группа называется *симметрической* степени n и обозначается S_n.

2.3. Алгебры с двумя операциями

В этом разделе мы обращаемся к объектам, знакомым читателю со школы. Среди алгебр с двумя операциями наиболее важными являются кольца и поля, а основными примерами колец и полей являются множества целых, рациональных и вещественных чисел с операциями сложения и умножения.

2.3.1. Кольца

Кольцо — это множество M с двумя бинарными операциями $+$ и $*$ (они называются сложением и умножением соответственно), в котором:

1. $(a+b)+c=a+(b+c)$	сложение ассоциативно.
2. $\exists 0 \in M\ (\forall a\ (a+0=0+a=a))$	существует нуль.
3. $\forall a\ (\exists -a\ (a+(-a)=0))$	существует обратный элемент.
4. $a+b=b+a$	сложение коммутативно, то есть кольцо — абелева группа по сложению.
5. $a*(b*c)=(a*b)*c$	умножение ассоциативно, то есть кольцо — полугруппа по умножению.
6. $a*(b+c)=(a*b)+(a*c)$ $(a+b)*c=(a*c)+(b*c)$	умножение дистрибутивно относительно сложения слева и справа.

Кольцо называется *коммутативным*, если

7. $a*b=b*a$	умножение коммутативно.

Кольцо называется кольцом *с единицей*, если

8. $\exists 1 \in M\ (a*1=1*a=a)$	существует единица, то есть кольцо с единицей — моноид по умножению.

ТЕОРЕМА *В кольце выполняются следующие соотношения:*

1. $0*a=a*0=0$.
2. $a*(-b)=(-a)*b=-(a*b)$.
3. $(-a)*(-b)=a*b$.
4. $(-a)=a*(-1)$.
5. $-(a+b)=(-a)+(-b)$.
6. $a\neq 0 \Longrightarrow (a^{-1})^{-1}=a$.

Доказательство

[1] $0*a=(0+0)*a=(0*a)+(0*a) \Longrightarrow -(0*a)+(0*a)=-(0*a)+\big((0*a)+$
$+(0*a)\big)=\big(-(0*a)+(0*a)\big)+(0*a) \Longrightarrow 0=0+(0*a)=0*a.$

[2] $\big(a*(-b)\big)+(a*b)=a*(-b+b)=a*0=0,$
$(a*b)+\big((-a)*b\big)=\big(a+(-a)\big)*b=0*b=0.$

[3] $(-a)*(-b)=-\big(a*(-b)\big)=-\big(-(a*b)\big)=a*b.$

[4] $\big(a*(-1)\big)+a=\big(a*(-1)\big)+(a*1)=a*(-1+1)=a*0=0.$

[5] $(a+b)+\big((-a)+(-b)\big)=(a+b)+\big((-b)+(-a)\big)=a+\big(b+(-b)\big)+(-a)=$
$=a+0+(-a)=a+(-a)=0.$

[6] $a^{-1}*a=1.$ □

Пример $\langle\mathbb{Z};+,*\rangle$ — коммутативное кольцо с единицей. Кроме того, $\forall n$ $(\langle\mathbb{Z}_n;+,*\rangle)$ — коммутативное кольцо с единицей. В частности, машинная арифметика целых чисел $\langle\mathbb{Z}_{2^{15}};+,*\rangle$ — коммутативное кольцо с единицей.

2.3.2. Области целостности

Если в кольце для некоторых ненулевых элементов x, y выполняется равенство $x*y=0$, то x называется *левым*, а y — *правым делителем нуля*.

Пример В машинной арифметике $\langle\mathbb{Z}_{2^{16}};+,*\rangle$ имеем $256*128=2^8*2^7=2^{15}=0$.

Заметим, что если в кольце нет делителей нуля, то $\forall x\neq 0, y\neq 0$ $(x*y\neq 0)$. В группе $\forall a,b,c$ $\big((a*b=a*c\Longrightarrow b=c)\ \&\ (b*a=c*a\Longrightarrow b=c)\big)$, однако в произвольном кольце это не так.

Пример В машинной арифметике $\langle\mathbb{Z}_{2^{16}};+,*\rangle$ имеем $2^8*2^7=0$ и $0*2^7=0$, но $0\neq 2^8=256$.

ТЕОРЕМА $\forall a\neq 0,b,c\ \big((a*b=a*c\Longrightarrow b=c)\ \&\ (b*a=c*a\Longrightarrow b=c)\big)\iff$
$\iff\forall x\neq 0, y\neq 0\ (x*y\neq 0)$.

Доказательство

[$\Longrightarrow$] От противного. Пусть $\exists x\neq 0, y\neq 0$ $(x*y=0)$. Тогда $x\neq 0\ \&\ x*y=$
$=0\ \&\ x*0=0\Longrightarrow y=0$.

[$\Longleftarrow$] $0=(a*b)+\big(-(a*b)\big)=(a*b)+\big(-(a*c)\big)=(a*b)+\big(a*(-c)\big)=a*\big(b+(-c)\big)$,
$a*\big(b+(-c)\big)=0\ \&\ a\neq 0\Longrightarrow b+(-c)=0\Longrightarrow b=c.$ □

Коммутативное кольцо с единицей, не имеющее делителей нуля, называется *областью целостности*.

Пример Целые числа $\langle\mathbb{Z};+,*\rangle$ являются областью целостности, а машинная арифметика $\langle\mathbb{Z}_{2^{15}};+,*\rangle$ — не является.

2.3.3. Поля

Поле — это множество M с двумя бинарными операциями $+$ и $*$, такими, что:

1. $(a+b)+c=a+(b+c)$	сложение ассоциативно.
2. $\exists 0 \in M\ (a+0=0+a=a)$	существует нуль.
3. $\forall a\ (\exists\ -a\ (a+-a=0))$	существует обратный элемент по сложению.
4. $a+b=b+a$	сложение коммутативно, то есть поле — абелева группа по сложению.
5. $a*(b*c)=(a*b)*c$	умножение ассоциативно.
6. $\exists 1 \in M\ (a*1=1*a=a)$	существует единица.
7. $\forall a \neq 0\ \left(\exists a^{-1}\ \left(a^{-1}*a=1\right)\right)$	существует обратный элемент по умножению.
8. $a*b=b*a$	умножение коммутативно, то есть ненулевые элементы образуют абелеву группу по умножению.
9. $a*(b+c)=(a*b)+(a*c)$	умножение дистрибутивно относительно сложения.

Таким образом, поле — это коммутативное кольцо с единицей, в котором каждый элемент, кроме нейтрального элемента по сложению, имеет обратный элемент по умножению.

Примеры

1. $\langle \mathbb{R}; +, * \rangle$ — поле вещественных чисел.
2. $\langle \mathbb{Q}; +, * \rangle$ — поле рациональных чисел.
3. Пусть $E_2 \stackrel{\text{Def}}{=} \{0,1\}$. Определим операции $+, \cdot\colon E_2 \times E_2 \to E_2$ следующим образом: $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$, $1 \cdot 1 = 1$ (конъюнкция), $0+0=1+1=0$, $0+1=1+0=1$ (сложение по модулю 2). Тогда $\mathbb{Z}_2 \stackrel{\text{Def}}{=} \langle E_2; +, \cdot \rangle$ является полем и называется *двоичной арифметикой*. В двоичной арифметике нуль — это 0, единица — это 1, $-1 = 1^{-1} = 1$, а 0^{-1} — как всегда, не определён.
4. Если в условиях предыдущего примера взять в качестве сложения дизъюнкцию, то есть определить операции следующим образом: $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$, $1 \cdot 1 = 1$ (конъюнкция), $0+0=0$, $0+1=1+0=1+1=1$ (дизъюнкция), то структура $\langle E_2; \vee, \cdot \rangle$ не является полем, поскольку у элемента 1 нет обратного по сложению.

ТЕОРЕМА 1 *Поле является областью целостности:* $a * b = 0 \Longrightarrow a = 0 \vee b = 0$.

Доказательство $a*b=0\ \&\ a\neq 0 \Longrightarrow b = 1*b = (a^{-1}*a)*b = a^{-1}*(a*b) = a^{-1}*0 = 0$, $a*b=0\ \&\ b\neq 0 \Longrightarrow a = 1*a = (b^{-1}*b)*a = b^{-1}*(b*a) = b^{-1}*(a*b) = b^{-1}*0 = 0$. □

ТЕОРЕМА 2 *Если $a \neq 0$, то в поле единственным образом разрешимо уравнение $a * x + b = 0$ (решение: $x = -(a^{-1}) * b$).*

Доказательство $a*x+b=0 \Longrightarrow a*x+b+(-b)=0+(-b) \Longrightarrow a*x+\big(b+(-b)\big)=$ $=-b \Longrightarrow a*x+0=-b \Longrightarrow a*x=-b \Longrightarrow a^{-1}*(a*x)=a^{-1}*(-b) \Longrightarrow (a^{-1}*a)*x=$ $=-(a^{-1}*b) \Longrightarrow 1*x=-(a^{-1}*b) \Longrightarrow x=-(a^{-1}*b)$. □

2.4. Векторные пространства и модули

Понятие векторного пространства должно быть известно читателю из курса средней школы и других математических курсов. Обычно это понятие ассоциируется с геометрической интерпретацией векторов в пространствах $\mathbb{R}^2$ и $\mathbb{R}^3$. В этом разделе даны и другие примеры векторных пространств, которые используются в последующих главах для решения задач, весьма далеких от геометрической интерпретации.

2.4.1. Векторное пространство

Пусть $\mathcal{F} = \langle F; +, \cdot \rangle$ — поле с операцией сложения $+$, операцией умножения $\cdot$, *аддитивным* нейтральным элементом (нулем) 0 и *мультипликативным* (единицей) 1. Пусть $\mathcal{V} = \langle V; + \rangle$ — абелева группа с операцией $+$ и нейтральным элементом $\mathbf{0}$. Если существует операция $F \times V \to V$ (знак этой операции опускается), такая, что для любых $a, b \in F$ и для любых $\mathbf{x}, \mathbf{y} \in V$ выполняются соотношения:

1) $(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$,
2) $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$,
3) $(a \cdot b)\mathbf{x} = a(b\mathbf{x})$,
4) $1\mathbf{x} = \mathbf{x}$,

то $\mathcal{V}$ называется *векторным пространством* над полем $\mathcal{F}$, элементы F называются *скалярами*, элементы V называются *векторами*, нейтральный элемент группы V называется *нуль-вектором* и обозначается $\mathbf{0}$, а необозначенная операция $F \times V \to V$ называется *умножением вектора на скаляр*.

Примеры

1. Пусть $\mathcal{F} = \langle F; +, \cdot \rangle$ — некоторое поле. Рассмотрим множество кортежей F^n. Тогда $\mathcal{F}^n = \langle F^n; + \rangle$, где

$$(a_1, \ldots, a_n) + (b_1, \ldots, b_n) \stackrel{\text{Def}}{=} (a_1 + b_1, \ldots, a_n + b_n)$$

является абелевой группой, в которой

$$-(a_1, \ldots, a_n) \stackrel{\text{Def}}{=} (-a_1, \ldots, -a_n) \quad \text{и} \quad \mathbf{0} \stackrel{\text{Def}}{=} (0, \ldots, 0).$$

Положим

$$a(a_1, \ldots, a_n) \stackrel{\text{Def}}{=} (a \cdot a_1, \ldots, a \cdot a_n).$$

Тогда $\mathcal{F}^n$ является векторным пространством над $\mathcal{F}$ для любого (конечного) n. В частности, $\mathbb{R}^n$ является векторным пространством для любого n. Векторное пространство $\mathbb{R}^2$ (и $\mathbb{R}^3$) имеет естественную геометрическую интерпретацию (рис. 2.1).

2. Двоичная арифметика $\mathbb{Z}_2 = \langle E_2; +_2, \cdot \rangle$ является полем, а булеан $\langle 2^M; \triangle \rangle$ с симметрической разностью является абелевой группой. Положим $1X \stackrel{\text{Def}}{=} X$, $0X \stackrel{\text{Def}}{=} \varnothing$. Таким образом, булеан с симметрической разностью является векторным пространством над двоичной арифметикой.

3. Пусть $X = \{x_1, \ldots, x_n\}$ — произвольное конечное множество, а $\mathcal{F} = \langle F; +, \cdot \rangle$ — поле. Рассмотрим множество V_X всех формальных сумм вида

$$\sum_{x \in X} \lambda_x x,$$

где $\lambda_x \in F$.

Положим

$$\sum_{x \in X} \lambda_x x + \sum_{x \in X} \mu_x x \stackrel{\text{Def}}{=} \sum_{x \in X} (\lambda_x + \mu_x) x,$$
$$\alpha \sum_{x \in X} \lambda_x x \stackrel{\text{Def}}{=} \sum_{x \in X} (\alpha \cdot \lambda_x) x,$$

где $\alpha, \lambda_x, \mu_x \in F$.

Ясно, что V_X является векторным пространством, а X является его множеством образующих или базисом (см. 2.4.3). В таком случае говорят, что V_X — это векторное пространство, «натянутое» на множество X.

4. В условиях предыдущего примера ($X = \{x_1, \ldots, x_n\}$ — произвольное конечное множество, а $\mathcal{F} = \langle F; +, \cdot \rangle$ — поле) рассмотрим множество $\Phi := \{f \mid f\colon X \to F\}$. Положим

$$(f + g)(x) \stackrel{\text{Def}}{=} f(x) + g(x), \quad \text{где} \quad f, g \in \Phi,$$
$$(\alpha f)(x) \stackrel{\text{Def}}{=} \alpha \cdot f(x), \quad \text{где} \quad \alpha \in F, f \in \Phi.$$

Нетрудно видеть, что Φ является векторным пространством.

ТЕОРЕМА 1. $\forall \mathbf{x} \in V \ (0\mathbf{x} = \mathbf{0})$.

2. $\forall a \in F \ (a\mathbf{0} = \mathbf{0})$.

Доказательство

[1] $0\mathbf{x} = (1 - 1)\mathbf{x} = 1\mathbf{x} - 1\mathbf{x} = \mathbf{x} - \mathbf{x} = \mathbf{0}$.

[2] $a\mathbf{0} = a(\mathbf{0} - \mathbf{0}) = a\mathbf{0} - a\mathbf{0} = (a - a)\mathbf{0} = 0\mathbf{0} = \mathbf{0}$. □

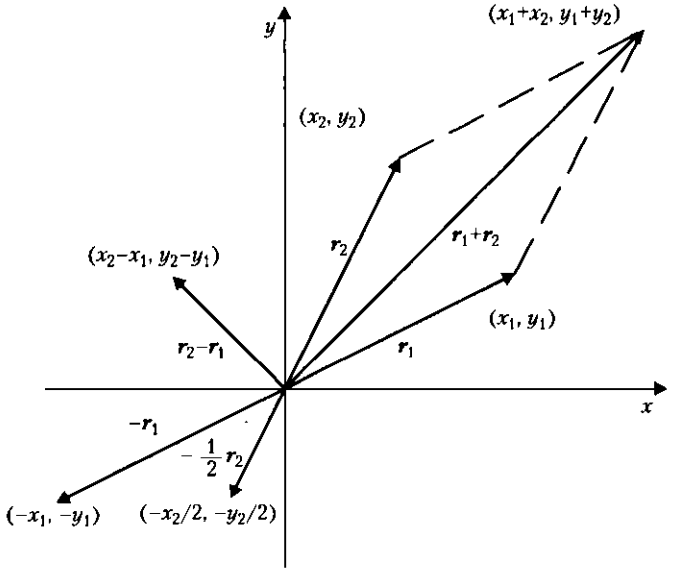

Рис. 2.1. Операции над векторами в $\mathbb{R}^2$

2.4.2. Линейные комбинации

Если $\mathcal{V}$ — векторное пространство над полем $\mathcal{F}$, S — некоторое множество векторов, $S \subset V$, то конечная сумма вида

$$\sum_{i=1}^{n} a_i \mathbf{x}_i, \quad a_i \in F, \quad \mathbf{x}_i \in S,$$

называется *линейной комбинацией* векторов из S. Пусть $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ — конечное множество векторов. Если

$$\sum_{i=1}^{k} a_i \mathbf{x}_i = \mathbf{0} \Longrightarrow \forall\, i \in 1..k \;\; (a_i = 0),$$

то множество X называется *линейно независимым*. В противном случае, то есть если

$$\exists\, a_1, \ldots, a_k \in F \;\; \left(\bigvee_{i=1}^{k} a_i \neq 0 \;\&\; \sum_{i=1}^{k} a_i \mathbf{x}_i = \mathbf{0} \right),$$

множество X называется *линейно зависимым*.

ТЕОРЕМА *Линейно независимое множество векторов не содержит нуль-вектора.*

Доказательство От противного. Пусть линейно независимое множество $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ содержит нуль-вектор и пусть для определённости $\mathbf{x}_1 = \mathbf{0}$.

Положим $a_1 := 1$, $a_2 := 0, \ldots, a_k := 0$. Имеем $\sum\limits_{i=1}^{n} a_i \mathbf{x}_i = 1\mathbf{0} + 0\mathbf{x}_2 + \cdots + 0\mathbf{x}_k = \mathbf{0} + \mathbf{0} + \ldots + \mathbf{0} = \mathbf{0}$, что противоречит линейной независимости S. □

2.4.3. Базис и размерность

Подмножество векторов $S \in V$, такое, что любой элемент V может быть представлен в виде линейной комбинации элементов S, называется *порождающим* множеством пространства $\mathcal{V}$ или *множеством образующих*. Линейно независимое порождающее множество называется *базисом* векторного пространства.

ТЕОРЕМА 1 *Пусть векторное пространство $\mathcal{V}$ имеет базис $B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. Тогда каждый элемент векторного пространства имеет единственное представление в данном базисе.*

Доказательство Пусть $\mathbf{x} = \sum\limits_{i=1}^{n} a_i \mathbf{e}_i$ и $\mathbf{x} = \sum\limits_{i=1}^{n} b_i \mathbf{e}_i$. Тогда $\mathbf{0} = \mathbf{x} - \mathbf{x} = \sum\limits_{i=1}^{n} a_i \mathbf{e}_i - \sum\limits_{i=1}^{n} b_i \mathbf{e}_i = \sum\limits_{i=1}^{n} (a_i - b_i)\mathbf{e}_i \Longrightarrow \forall\, i \in 1..n \;\; (a_i - b_i) = 0 \Longrightarrow \forall\, i \;\; (a_i = b_i)$. □

Коэффициенты разложения вектора в данном базисе называются его *координатами*.

ТЕОРЕМА 2 *Пусть $B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ — базис, а $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ — линейно независимое множество векторного пространства $\mathcal{V}$. Тогда $n \geqslant m$.*

Доказательство От противного. Пусть $n < m$ и B — базис. Тогда

$$\exists\, a_1, \ldots, a_n \in F \;\; (\mathbf{x}_1 = a_1\mathbf{e}_1 + \ldots + a_n\mathbf{e}_n).$$

Имеем $\mathbf{x}_1 \neq \mathbf{0} \Longrightarrow \bigvee\limits_{i=1}^{n} a_i \neq 0$. Пусть для определённости $a_1 \neq 0$. Тогда

$$\mathbf{e}_1 = a_1^{-1}\mathbf{x}_1 - (a_1^{-1}a_2)\mathbf{e}_2 - (a_1^{-1}a_n)\mathbf{e}_n.$$

Так как B порождает $\mathcal{V}$, то и $\{\mathbf{x}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}$ тоже порождает $\mathcal{V}$. Аналогично $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{e}_3, \ldots, \mathbf{e}_n\}$ порождает $\mathcal{V}$. Продолжая процесс, получаем, что $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ порождает $\mathcal{V}$. Следовательно,

$$\mathbf{x}_{n+1} = \sum_{i=1}^{n} b_i \mathbf{x}_i \Longrightarrow \mathbf{x}_{n+1} - \sum_{i=1}^{n} b_i \mathbf{x}_i = \mathbf{0}.$$

Таким образом, X — линейно зависимое множество, что противоречит условию. □

СЛЕДСТВИЕ *Если в векторном пространстве $\mathcal{V}$ множество векторов $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ — линейно независимо и векторы множества $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$ выражаются в виде линейных комбинаций векторов множества X, другими словами, $\forall i \in 1..m \left(\mathbf{y}_i = \sum_{j=1}^{n} a_{ij}\mathbf{x}_j\right)$, причём $m > n$, то множество Y — линейно зависимое.*

Доказательство Рассмотрим подпространство $\mathcal{V}_A$ векторного пространства $\mathcal{V}$, порождаемое векторами множества A. В подпространстве $\mathcal{V}_A$ множество A — базис, а векторы множества Y — элементы. Далее от противного. Пусть множество Y линейно независимое, тогда по теореме $n \geqslant m$, что противоречит условию. □

ТЕОРЕМА 3 *Если B_1 и B_2 — базисы векторного пространства $\mathcal{V}$, то $|B_1| = |B_2|$.*

Доказательство B_1 — базис и B_2 — линейно независимое множество. Следовательно, $|B_1| \geqslant |B_2|$. С другой стороны, B_2 — базис, и B_1 — линейно независимое множество. Следовательно, $|B_2| \geqslant |B_1|$. Имеем $|B_1| = |B_2|$. □

Если векторное пространство $\mathcal{V}$ имеет базис B, то количество элементов в базисе называется *размерностью* векторного пространства и обозначается $\dim \mathcal{V}$. Векторное пространство, имеющее конечный базис, называется *конечномерным*. Векторное пространство, не имеющее базиса (в смысле приведенного определения), называется *бесконечномерным*.

Примеры

1. Одноэлементные подмножества образуют базис булеана, $\dim 2^M = |M|$.

2. Кортежи вида $(0, \ldots, 1, 0, \ldots, 0)$ образуют базис пространства $\mathcal{F}^n$, $\dim \mathcal{F}^n = n$.

ЗАМЕЧАНИЕ

Если определить базис как максимальное линейно независимое множество (то есть множество, которое нельзя расширить, не нарушив свойства линейной независимости), то понятие базиса можно распространить и на бесконечномерные пространства.

2.4.4. Модули

Понятие модуля во многом аналогично понятию векторного пространства, с той лишь разницей, что векторы умножаются не на элементы из поля, а на элементы из произвольного кольца. Пусть $\mathcal{R} = \langle R; +, * \rangle$ — некоторое кольцо с операцией сложения $+$, операцией умножения $*$, нулевым элементом 0 и единичным элементом 1. Абелева группа $\mathcal{M} = \langle M; + \rangle$ называется *модулем* над кольцом $\mathcal{R}$, если задана операция умножения на скаляр $R \times M \to M$ (здесь никак не обозначаемая), такая что:

1) $(a+b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$;
2) $a(\mathbf{x}+\mathbf{y}) = a\mathbf{x} + a\mathbf{y}$;
3) $(a*b)\mathbf{x} = a(b\mathbf{x})$;
4) $1\mathbf{x} = \mathbf{x}$,

где $a, b \in R$, а $\mathbf{x}, \mathbf{y} \in M$. Как и в случае векторных пространств, элементы кольца $\mathcal{R}$ называются *скалярами*, а элементы группы $\mathcal{M}$ — *векторами*.

Примеры

1. Векторное пространство $\mathcal{V}$ над полем $\mathcal{F}$ является модулем над $\mathcal{F}$, так как поле, по определению, является кольцом.
2. Множество векторов с целочисленными координатами в $\mathbb{R}^n$ является модулем над кольцом $\mathbb{Z}$.
3. Любая абелева группа есть модуль над кольцом $\mathbb{Z}$, если операцию умножения на скаляр $n \in \mathbb{Z}, n > 0$ определить следующим образом:

$$n\mathbf{v} \stackrel{\text{Def}}{=} \underbrace{\mathbf{v} + \mathbf{v} + \cdots + \mathbf{v}}_{n \text{ раз}}.$$

2.5. Решётки

Решётки иногда называют «структурами», но слово «структура» перегружено, и мы не будем использовать его в этом значении. Решётки сами по себе часто встречаются в разных программистских задачах, но еще важнее то, что понятие решётки непосредственно подводит нас к понятию булевой алгебры, значение которой для основ современной двоичной компьютерной техники трудно переоценить.

2.5.1. Определения

Решётка — это множество M с двумя бинарными операциями $\cap$ и $\cup$, такими, что выполнены следующие условия (аксиомы решётки):

1. Идемпотентность:
   $a \cap a = a, \quad a \cup a = a.$
2. Коммутативность:
   $a \cap b = b \cap a, \quad a \cup b = b \cup a.$
3. Ассоциативность:
   $(a \cap b) \cap c = a \cap (b \cap c), \quad (a \cup b) \cup c = a \cup (b \cup c).$
4. Поглощение:
   $(a \cap b) \cup a = a, \quad (a \cup b) \cap a = a.$
5. Решётка называется *дистрибутивной*, если
   $a \cup (b \cap c) = (a \cup b) \cap (a \cup c), \quad a \cap (b \cup c) = (a \cap b) \cup (a \cap c).$

2.5.2. Ограниченные решётки

Если в решётке $\exists 0 \in M\ (\forall a\ (0 \cap a = 0))$, то 0 называется *нулем* (или *нижней гранью*) решётки. Если в решётке $\exists 1 \in M\ (\forall a\ (1 \cup a = 1))$, то 1 называется *единицей* (или *верхней гранью*) решётки. Решётка с верхней и нижней гранями называется *ограниченной*.

ТЕОРЕМА 1 *Если нижняя (верхняя) грань существует, то она единственна.*

Доказательство Пусть $0'$ — ещё один нуль решётки. Тогда $0' = 0 \cap 0' = 0' \cap 0 = 0$. □

ТЕОРЕМА 2 $a \cap b = b \iff a \cup b = a$.

Доказательство

[$\Longrightarrow$] $a \cup b = a \cup (a \cap b) = (a \cap b) \cup a = a$.

[$\Longleftarrow$] $a \cap b = (a \cup b) \cap b = (b \cup a) \cap b = b$. □

СЛЕДСТВИЕ $0 \cup a = 1 \cap a = a$.

2.5.3. Решётка с дополнением

В ограниченной решётке элемент a' называется *дополнением* элемента a, если $a \cap a' = 0$ и $a \cup a' = 1$. Если $\forall a \in M\ (\exists a' \in M\ (a \cap a' = 0\ \&\ a \cup a' = 1))$, то ограниченная решётка называется *решёткой с дополнением*. Вообще говоря, дополнение не обязано существовать и не обязано быть единственным.

ТЕОРЕМА (о свойствах дополнения) *В ограниченной дистрибутивной решётке с дополнением выполняется следующее:*

1. *Дополнение a' единственно.*
2. *Дополнение инволютивно: $a'' = a$.*
3. *Грани дополняют друг друга: $1' = 0, 0' = 1$.*
4. *Выполняются законы де Моргана: $(a \cap b)' = a' \cup b'$, $(a \cup b)' = a' \cap b'$.*

Доказательство

[1] Пусть x, y — дополнения a. Тогда $x = x \cap 1 = x \cap (a \cup y) = (x \cap a) \cup (x \cap y) = 0 \cup (x \cap y) = x \cap y = y \cap x = 0 \cup (y \cap x) = (y \cap a) \cup (y \cap x) = y \cap (a \cup x) = y \cap 1 = y$.

[2] $(a \cup a' = 1 \Longrightarrow a' \cup a = 1,\ a \cap a' = 0 \Longrightarrow a' \cap a = 0) \Longrightarrow a = a''$.

[3] $(1 \cap 0 = 0, 0' \cap 0 = 0) \Longrightarrow 1 = 0',\ (1 \cup 0 = 1, 1 \cup 1' = 1) \Longrightarrow 0 = 1'$.

[4] $(a \cap b) \cap (a' \cup b') = (a \cap b \cap a') \cup (a \cap b \cap b') = (0 \cap b) \cup (a \cap 0) = 0 \cup 0 = 0$,
$(a \cap b) \cup (a' \cup b') = (a \cup a' \cup b') \cap (b \cup a' \cup b') = (1 \cup b') \cap (1 \cup a') = 1 \cap 1 = 1$,
$(a \cup b) \cap (a' \cap b') = (a \cap a' \cap b') \cup (b \cap a' \cap b') = (0 \cap b') \cup (a' \cap 0) = 0 \cup 0 = 0$,
$(a \cup b) \cup (a' \cap b') = (a \cup b \cup a') \cap (a \cup b \cup b') = (1 \cup b) \cap (1 \cup a) = 1 \cap 1 = 1$. □

2.5.4. Частичный порядок в решётке

В любой решётке можно естественным образом ввести частичный порядок, а именно: $a \prec b \stackrel{\text{Def}}{=} a \cap b = a$.

ТЕОРЕМА 1 *Отношение $\prec$ является отношением частичного порядка.*

Доказательство

[Рефлексивность] $a \cap a = a \Longrightarrow a \prec a$.

[Антисимметричность] $a \prec b \;\&\; b \prec a \Longrightarrow a = a \cap b = b \cap a = b$.

[Транзитивность] $a \prec b \;\&\; b \prec c \Longrightarrow a \cap c = (a \cap b) \cap c = a \cap (b \cap c) = a \cap b = a \Longrightarrow$
$\Longrightarrow a \prec c$. □

Наличие частичного порядка в решётке не случайно, это её характеристическое свойство. Более того, обычно решётку определяют, начиная с частичного порядка, следующим образом. Пусть M — частично упорядоченное множество с частичным порядком $\prec$. Напомним, что элемент x называется *нижней границей* для a и b, если $x \prec a \;\&\; x \prec b$. Аналогично, y называется *верхней границей* для a и b, если $a \prec y \;\&\; b \prec y$. Элемент x называется *нижней гранью (наибольшей нижней границей)* элементов a и b, если x — нижняя граница элементов a и b и для любой другой нижней границы v элементов a и b выполняется $v \prec x$. Обозначение: $x = \inf(a, b)$. Аналогично, y называется *верхней гранью (наименьшей верхней границей)* элементов a и b, если y — верхняя граница элементов a и b и для любой другой верхней границы u элементов a и b выполняется $y \prec u$. Обозначение: $y = \sup(a, b)$ (см. также 1.8.4).

ТЕОРЕМА 2 *Если нижняя (верхняя) грань существует, то она единственна.*

Доказательство $x = \inf(a, b) \;\&\; y = \inf(a, b) \Longrightarrow y \prec x \;\&\; x \prec y \Longrightarrow x = y$. □

ТЕОРЕМА 3 *Если в частично упорядоченном множестве для любых двух элементов существуют нижняя и верхняя грани, то это множество образует решётку относительно* inf *и* sup.

Доказательство Положим $x \cap y \stackrel{\text{Def}}{=} \inf(x, y)$, $x \cup y \stackrel{\text{Def}}{=} \sup(x, y)$ и проверим выполнение аксиом решётки.

[1] $\inf(x, x) = x \Longrightarrow x \cap x = x$, $\sup(x, x) = x \Longrightarrow x \cup x = x$.

[2] $\inf(x, y) = \inf(y, x) \Longrightarrow x \cap y = y \cap x$, $\sup(x, y) = \sup(y, x) \Longrightarrow x \cup y = y \cup x$.

[3] $\inf\big(x, \inf(y, z)\big) = \inf\big(\inf(x, y), z\big) \Longrightarrow x \cap (y \cap z) = (x \cap y) \cap z$,
$\sup\big(x, \sup(y, z)\big) = \sup\big(\sup(x, y), z\big) \Longrightarrow x \cup (y \cup z) = (x \cup y) \cup z$.

[4] $\sup\big(\inf(a, b), a\big) = a \Longrightarrow (a \cap b) \cup a = a$, $\inf\big(\sup(a, b), a\big) = a \Longrightarrow (a \cup b) \cap a = a$. □

2.5.5. Булевы алгебры

Дистрибутивная ограниченная решётка, в которой для каждого элемента существует дополнение, называется *булевой алгеброй*. Свойства булевой алгебры:

1. $a \cup a = a, \quad a \cap a = a$
   по определению решётки.
2. $a \cup b = b \cup a, \quad a \cap b = b \cap a$
   по определению решётки.
3. $a \cup (b \cup c) = (a \cup b) \cup c, \quad a \cap (b \cap c) = (a \cap b) \cap c$
   по определению решётки.
4. $(a \cap b) \cup a = a, \quad (a \cup b) \cap a = a$
   по определению решётки.
5. $a \cup (b \cap c) = (a \cup b) \cap (a \cup c), \quad a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$
   по дистрибутивности.
6. $a \cup 1 = 1, \quad a \cap 0 = 0$
   по ограниченности.
7. $a \cup 0 = a, \quad a \cap 1 = a$
   по следствию из теоремы 2 подраздела 2.5.2.
8. $a'' = a$
   по теореме подраздела 2.5.3.
9. $(a \cap b)' = a' \cup b', \quad (a \cup b)' = a' \cap b'$
   по теореме подраздела 2.5.3.
10. $a \cup a' = 1, \quad a \cap a' = 0$
    по определению.

Примеры

1. $\langle 2^M; \cap, \cup, ^- \rangle$ — булева алгебра, причём M — верхняя грань, $\varnothing$ — нижняя грань, $\subset$ — естественный частичный порядок.
2. $\langle E_2; \&, \vee, \neg \rangle$ — булева алгебра, где $\&$ — конъюнкция, $\vee$ — дизъюнкция, $\neg$ — отрицание, причём 1 — верхняя грань, 0 — нижняя грань, а импликация $\Longrightarrow$ — естественный частичный порядок.
3. Пусть T — некоторое конечное множество взаимно простых чисел. Пусть P — множество всех возможных произведений различных чисел из T, включая пустое произведение, по определению равное 1. (Заметим, что $P \sim 2^T$). Тогда $\langle P; \text{НОД}, \text{НОК}, \text{ДОП} \rangle$ — булева алгебра, где НОД — наибольший общий делитель, НОК — наименьшее общее кратное, а

$$\text{ДОП}(n) \stackrel{\text{Def}}{=} \frac{1}{n} \prod_{q \in T} q,$$

причём $\prod\limits_{q \in T} q$ — верхняя грань, 1 — нижняя грань, отношение «n делится на m» — естественный частичный порядок.

2.6. Матроиды и жадные алгоритмы

В этом разделе рассматривается следующая простая и часто встречающаяся экстремальная задача. Задано конечное множество E, элементам которого приписаны положительные веса, и семейство $\mathcal{E} \subset 2^E$ подмножеств множества E. Требуется найти в семействе $\mathcal{E}$ элемент (подмножество E) максимального суммарного веса. Оказывается, что если семейство $\mathcal{E}$ обладает особой структурой (является матроидом), то для решения задачи достаточно применить очень простой и эффективный алгоритм (жадный алгоритм). Ясное понимание природы и области применимости жадных алгоритмов совершенно необходимо каждому программисту. Матроиды, рассматриваемые в этом разделе, вообще говоря, не являются алгебраическими структурами в том смысле, который придан этому понятию в первом разделе данной главы. Однако матроиды имеют много общего с рассмотренными алгебраическими структурами (в частности, с линейно независимыми множествами в векторных пространствах и модулях) и изучаются сходными методами.

2.6.1. Матроиды

Матроидом $M = \langle E, \mathcal{E} \rangle$ называется пара, состоящая из конечного множества E, $|E| = n$, и семейства его подмножеств $\mathcal{E} \subset 2^E$, таких, что выполняются следующие три аксиомы:

M_1: $\varnothing \in \mathcal{E}$;
M_2: $A \in \mathcal{E}\ \&\ B \subset A \Longrightarrow B \in \mathcal{E}$;
M_3: $A, B \in \mathcal{E}\ \&\ |B| = |A| + 1 \Longrightarrow \exists e \in B \setminus A\ \ (A + e \in \mathcal{E})$.

Элементы множества $\mathcal{E}$ называются *независимыми*, а остальные подмножества E (то есть элементы множества $2^E \setminus \mathcal{E}$) — *зависимыми* множествами.

ЗАМЕЧАНИЕ

Аксиома M_1 исключает из рассмотрения вырожденный случай $\mathcal{E} = \varnothing$.

Пример Семейство линейно независимых множеств векторов любого векторного пространства является матроидом. Действительно, по определению можно считать, что пустое множество векторов линейно независимо. Всякое подмножество линейно независимого множества векторов линейно независимо. Пусть $A := \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ и $B := \{\mathbf{b}_1, \ldots, \mathbf{b}_{m+1}\}$ — линейно независимые множества векторов. Если бы все векторы из множества B выражались в виде линейной комбинации векторов из множества A, то множество B было бы линейно зависимым по следствию к теореме 2 подраздела 2.4.3. Стало быть, среди векторов множества B есть по крайней мере один вектор $\mathbf{b}$, который не входит в множество A и не выражается в виде линейной комбинации векторов из множества A. Добавление вектора $\mathbf{b}$ к множеству A образует линейно независимое множество.

ЗАМЕЧАНИЕ

Само понятие матроида возникло в результате исследования линейной независимости в векторных пространствах.

2.6.2. Максимальные независимые подмножества

Пусть $M = \langle E, \mathcal{E} \rangle$ — матроид и $X \subset E$ — произвольное множество. *Максимальным* независимым подмножеством множества X называется множество Y, такое, что

$$Y \subset X \;\&\; Y \in \mathcal{E} \;\&\; \forall Z \in \mathcal{E}\ (Y \subset Z \subset X \Longrightarrow Z = Y).$$

Множество максимальных независимых подмножеств множества X обозначим $\underline{X}$:

$$\underline{X} \stackrel{\text{Def}}{=} \{Y \subset E \mid Y \subset X \;\&\; Y \in \mathcal{E} \;\&\; \forall Z \in \mathcal{E}\ (Y \subset Z \subset X \Longrightarrow Z = Y)\}.$$

Рассмотрим следующее утверждение:

$$M_4\colon\ \forall X\ (Y \in \underline{X} \;\&\; Z \in \underline{X} \Longrightarrow |Y| = |Z|),$$

то есть все максимальные независимые подмножества данного множества равномощны.

ТЕОРЕМА *Пусть $M = \langle E, \mathcal{E} \rangle$ и выполнены аксиомы M_1 и M_2. Тогда аксиома M_3 и утверждение M_4 эквивалентны, то есть*

$$A, B \in \mathcal{E} \;\&\; |B| = |A| + 1 \Longrightarrow \exists e \in B \setminus A\ (A + e \in \mathcal{E})$$

тогда и только тогда, когда

$$\forall X\ (Y \in \underline{X} \;\&\; Z \in \underline{X} \Longrightarrow |Y| = |Z|).$$

Доказательство

[$\Longrightarrow$] Пусть выполнены утверждения M_1, M_2, M_3 (то есть M — матроид). Покажем от противного, что выполняется и M_4. Пусть $Y, Z \in \underline{X}$, $|Y| \neq |Z|$ и для определённости $|Y| > |Z|$. Возьмем $Y' \subset Y$, так что $|Y'| = |Z| + 1$. Тогда по свойству M_3 имеем $\exists e \in Y' \setminus Z\ (W := Z + e \in \mathcal{E})$. Таким образом, имеем $W \in \mathcal{E}$, $Z \subset W$, $W \subseteq X$, $Z \neq W$, что противоречит предположению $Z \in \underline{X}$.

[$\Longleftarrow$] Пусть выполнены утверждения M_1, M_2, M_4. Покажем от противного, что выполняется и M_3. Возьмем $A, B \in \mathcal{E}$, так что $|B| = |A| + 1$. Допустим, что $\neg\exists e \in B \setminus A\ (A + e \in \mathcal{E})$, то есть $\forall e \in B \setminus A\ (A + e \notin \mathcal{E})$. Рассмотрим $C := A \cup B$. Имеем $A \in \underline{C}$. Но $B \in \mathcal{E}$, поэтому $\exists B'\ (B \subset B' \;\&\; B' \in \mathcal{E} \;\&\; B' \in \underline{C})$. По условию M_4 имеем $|B'| = |A|$. Но $|A| = |B'| \geqslant |B| = |A| + 1$ — противоречие. □

ЗАМЕЧАНИЕ

Таким образом, M_1, M_2, M_4 — эквивалентная система аксиом матроида.

2.6.3. Базисы

Максимальные независимые подмножества множества E называются *базисами* матроида $M = \langle E, \mathcal{E} \rangle$. Всякий матроид имеет базисы, что видно из следующего алгоритма.

Алгоритм 2.1 Построение базиса матроида

Вход: Матроид $M = \langle E, \mathcal{E} \rangle$.
Выход: Множество $B \subset E$ элементов, образующих базис.

```
B := ∅ { вначале базис пуст }
for e ∈ E do
  if B + e ∈ E then
    B := B + e { расширяем базис допустимым образом }
  end if
end for
```

Обоснование Пусть $B_0 = \varnothing$, $B_1, ..., B_k = B$ — последовательность значений переменной B в процессе работы алгоритма. По построению $\forall i \ (B_i \in \mathcal{E})$. Пусть $B \notin \underline{E}$, то есть B не является максимальным. Тогда $\exists B' \ (B \subset B' \ \& \ B' \neq B \ \& \ B' \in \mathcal{E})$. Возьмем $B'' \subset B'$ так что $|B''| = |B| + 1$, $B \subset B''$ и $B'' \in \mathcal{E}$. Рассмотрим $e \in B' \setminus B$. Элемент e не попал в множество B, но алгоритм просматривает *все* элементы, значит, элемент e был отвергнут на некотором шаге i, то есть $(B_{i-1} + e) \notin \mathcal{E}$. Но $e \in B'' \ \& \ B_{i-1} \subset B'' \Longrightarrow (B_{i-1} + e) \in \mathcal{E}$. По аксиоме M_2 имеем $(B_{i-1} + e) \in \mathcal{E}$ — противоречие. □

СЛЕДСТВИЕ *Все базисы матроида равномощны.*

2.6.4. Жадный алгоритм

Сформулируем более точно рассматриваемую экстремальную задачу. Пусть имеются конечное множество E, $|E| = n$, *весовая* функция $w\colon E \to \mathbb{R}_+$ и семейство $\mathcal{E} \subset 2^E$. Требуется найти $X \in \mathcal{E}$, такое, что

$$w(X) = \max_{Y \in \mathcal{E}} w(Y), \quad \text{где } w(Z) \stackrel{\text{Def}}{=} \sum_{e \in Z \subset E} w(e).$$

Другими словами, необходимо выбрать в указанном семействе подмножество наибольшего веса. Не ограничивая общности, можно считать, что $w(e_1) \geqslant \ldots \geqslant w(e_n) > 0$. Рассмотрим следующий алгоритм.

Алгоритм 2.2 Жадный алгоритм

Вход: множество $E = \{e_1, \ldots, e_n\}$, семейство его подмножеств $\mathcal{E}$ и весовая функция w. Множество E линейно упорядочено в порядке убывания весов элементов.
Выход: подмножество X, такое, что $X \in \mathcal{E}$ и $w(X) = \max\limits_{Y \in \mathcal{E}} w(Y)$.

```
X := ∅ { вначале множество X пусто }
for i from 1 to n do
  if X + e_i ∈ E then
    X := X + e_i { добавляем в X первый подходящий элемент }
  end if
end for
```

Алгоритм такого типа называется *жадным*. Совершенно очевидно, что по построению окончательное множество $X \in \mathcal{E}$. Также очевидно, что жадный алгоритм является чрезвычайно эффективным: количество шагов составляет $O(n)$, то есть жадный алгоритм является *линейным*. (Не считая затрат на сортировку множества E и проверку независимости $X + e_i \in \mathcal{E}$.) Возникает вопрос: в каких случаях жадный алгоритм действительно решает задачу, поставленную в начале подраздела? Другими словами: когда выгодно быть жадным?

Пример Пусть дана матрица

$$\begin{matrix} 7 & 5 & 1 \\ 3 & 4 & 3 \\ 2 & 3 & 1 \end{matrix}$$

Рассмотрим следующие задачи:

1. Выбрать по одному элементу *из каждого столбца* так, чтобы их сумма была максимальна. Нетрудно видеть, что жадный алгоритм выберет следующие элементы:

$$\begin{matrix} \boxed{7} & \boxed{5} & 1 \\ 3 & 4 & \boxed{3} \\ 2 & 3 & 1 \end{matrix}$$

   которые действительно являются решением задачи.

2. Выбрать по одному элементу *из каждого столбца и из каждой строки* так, чтобы их сумма была максимальна. Нетрудно видеть, что жадный алгоритм выберет следующие элементы:

$$\begin{matrix} \boxed{7} & 5 & 1 \\ 3 & \boxed{4} & 3 \\ 2 & 3 & \boxed{1} \end{matrix}$$

   которые не являются решением задачи, поскольку существует лучшее решение:

$$\begin{matrix} \boxed{7} & 5 & 1 \\ 3 & 4 & \boxed{3} \\ 2 & \boxed{3} & 1 \end{matrix}$$

ТЕОРЕМА *Если $M = \langle E, \mathcal{E} \rangle$ — матроид, то для любой функции w жадный алгоритм находит независимое множество X с наибольшим весом; обратно, если же $M = \langle E, \mathcal{E} \rangle$ не является матроидом, то существует такая функция w, что множество X, найденное жадным алгоритмом, не будет подмножеством наибольшего веса.*

Доказательство

[$\Longrightarrow$] Пусть $M = \langle E, \mathcal{E} \rangle$ — матроид и пусть $X = \{x_1, \ldots, x_k\}$ — множество, построенное жадным алгоритмом. По построению $w(x_1) \geqslant \ldots \geqslant w(x_k) > 0$. Согласно алгоритму 2.1, множество X — базис матроида M. Пусть теперь $Y = \{y_1, \ldots, y_m\} \in \mathcal{E}$ — некоторое независимое множество. Имеем $m \leqslant k$, так как

X — базис. Покажем от противного, что $\forall i \in 1..m \ (w(y_i) \leqslant w(x_i))$. Пусть $\exists i \in 1..m \ (w(y_i) > w(x_i))$. Рассмотрим независимые множества $A = \{x_1, \ldots, x_{i-1}\}$ и $B = \{y_1, \ldots, y_{i-1}, y_i\}$. Имеем $\exists j \leqslant i \ (\{x_1, \ldots, x_{i-1}, y_j\})$ — независимое множество. Тогда $w(y_j) \geqslant w(y_i) > w(x_i)$, откуда следует, что

$$\exists p \leqslant i \ (w(x_1) \geqslant \ldots \geqslant w(x_{p-1}) \geqslant w(y_j) \geqslant w(x_p)),$$

что противоречит тому, что x_p — элемент с наибольшим весом, добавление которого к множеству $\{x_1, \ldots, x_{p-1}\}$ не нарушает независимости. Следовательно, $\forall i \ (w(y_i)) \leqslant w(x_i)$ и, значит, $w(Y) \leqslant w(X)$.

[$\Longleftarrow$] Пусть $M = \langle E; \mathcal{E} \rangle$ не является матроидом. Если нарушено условие M_2, то есть $\exists A, B \ (A \subset B \in \mathcal{E})$, но $A \notin \mathcal{E}$, то определим функцию w следующим образом:

$$w(e) := \begin{cases} 1, & \text{если } e \in A; \\ 0, & \text{если } e \in E \setminus A. \end{cases}$$

Тогда по алгоритму 2.2 $A \not\subset X \Longrightarrow w(X) < w(A) = w(B)$. Если же условие M_2 выполнено, но нарушено условие M_3, то $\exists A, B \ (|A| = k) \ \& \ |B| = k + 1$ и для любого элемента $e \in B \setminus A$ множество $A + e$ — зависимое. Пусть $p := |A \cap B|$. Тогда $p < k$. Выберем число ε так, что $0 < \varepsilon < 1/(k - p)$. Определим функцию w следующим образом:

$$w(e) := \begin{cases} 1 + \varepsilon, & \text{если } e \in A; \\ 1, & \text{если } e \in B \setminus A; \\ 0, & \text{в остальных случаях.} \end{cases}$$

Заметим, что при таких весах жадный алгоритм сначала выберет все элементы из A и отбросит элементы из $B \setminus A$. В результате будет выбрано множество X, вес которого меньше веса множества B. Действительно, $w(X) = w(A) = k(1 + \varepsilon) = (k-p)(1+\varepsilon) + p(1+\varepsilon) < (k-p)(1+1/(k-p)) + p(1+\varepsilon) = (k-p+1) + p(1+\varepsilon) = w(B)$. □

ЗАМЕЧАНИЕ

Тот факт, что семейство $\mathcal{E}$ является матроидом, означает, что для решения поставленной экстремальной задачи можно применить жадный алгоритм, однако из этого не следует, что не может существовать ещё более эффективного алгоритма. С другой стороны, если семейство $\mathcal{E}$ не является матроидом, то это ещё не значит, что жадный алгоритм не найдет правильного решения — все зависит от свойств конкретной функции w.

ОТСТУПЛЕНИЕ

Жадные алгоритмы и их свойства были исследованы сравнительно недавно, но их значение в практике программирования чрезвычайно велико. Если удаётся свести конкретную экстремальную задачу к такой постановке, где множество допустимых вариантов (из которых нужно выбрать наилучший) является матроидом, то в большинстве случаев следует сразу применять жадный алгоритм, поскольку он достаточно эффективен в практическом смысле. Если же, наоборот, оказывается, что множество допустимых вариантов

не образует матроида, то это «плохой признак». Скорее всего, данная задача окажется трудноразрешимой. В этом случае целесообразно тщательно исследовать задачу для предварительного получения теоретических оценок сложности, чтобы избежать бесплодных попыток «изобрести» эффективный алгоритм там, где это на самом деле невозможно.

2.6.5. Примеры матроидов

Ниже приведены некоторые примеры матроидов, встречающихся в рассмотренных областях дискретной математики. В последних главах книги имеются дополнительные примеры матроидов, связанных с графами.

1. *Свободные матроиды.* Если E — произвольное конечное множество, то $M = \langle E, 2^E \rangle$ — матроид. Такой матроид называется *свободным*. В свободном матроиде каждое множество независимое.
2. *Матроиды разбиений.* Пусть $\{E_1, \ldots, E_k\}$ — некоторое разбиение множества E на непустые множества. Другими словами, $\bigcup_{i=1}^{k} E_k = E$, $E_i \cap E_j = \varnothing$, $E_j \neq \varnothing$. Положим $\mathcal{E} := \{A \subseteq E \mid 1 \geqslant |A \cap E_i|\}$, то есть в независимое множество входит не более чем один элемент каждого блока разбиения. Тогда $M = \langle E, \mathcal{E} \rangle$ — матроид. Действительно, условие M_2 выполнено. Если $A, B \in \mathcal{E}$ и $|B| = |A|+1$, то $\exists i \ (|E_i \cap A| = 0 \ \& \ |E_i \cap B| = 1)$. Обозначим $e := E_i \cap B$. Тогда $A + e \in \mathcal{E}$. Значит, выполнено условие M_3.
3. *Векторные пространства.* Линейно независимые множества векторов любого векторного пространства образуют матроид. Действительно, условия M_1 и M_2, очевидно, выполнены для линейно независимых множеств векторов. Условие M_4 выполнено по теореме подраздела 2.4.3.
4. *Матроид трансверсалей.* Пусть E — некоторое множество и $\mathcal{E}$ — семейство подмножеств этого множества (не обязательно различных), $\mathcal{E} \subset 2^E$. Подмножество $A \subset E$ называется *частичной трансверсалью* семейства $\mathcal{E}$, если A содержит не более чем по одному элементу для каждого подмножества E_i семейства $\mathcal{E}$. Частичные трансверсали над E образуют матроид (см. [8]).

Комментарии

Так же как и в предыдущей главе, сведения, изложенные здесь, настолько общеизвестны, что конкретные ссылки ниже следует рассматривать как примеры, а не как единственно рекомендуемые источники. Конкретные алгебраические структуры разделов 2.2–2.5 описаны в [24], [7], [31]. Книги [31] и [7] могут быть использованы как введение в общую алгебру. Учебник [13] также содержит необходимые сведения из общей алгебры, включённые в контекст изложения дискретной математики и комбинаторики. Монография [3] исчерпывающим образом освещает важные для приложений вопросы, бегло затронутые в подразделе 2.5.5. Описание жадных алгоритмов приводится в [8].

Упражнения

2.1. Пусть на множестве вещественных чисел заданы две операции:

$$\bar{*}(a,b) := a + b \quad \text{и} \quad \bar{+}(a,b) := \max(a,b).$$

Показать, что обе эти операции ассоциативны и коммутативны, операция $\bar{+}$ идемпотентна, а операция $\bar{*}$ дистрибутивна относительно $\bar{+}$.

2.2. Пусть $L \stackrel{\text{Def}}{=} \{ax + b \mid a, b \in \mathbb{R}\}$ — множество линейных функций. Операция *линейной замены переменной* определяется следующим образом:

$$(ax + b) \circ (cx + d) \stackrel{\text{Def}}{=} (a(cx + d) + b) = (ac)x + (ad + b)).$$

Доказать, что множество линейных функций образует группу относительно линейной замены переменной.

2.3. Пусть на множестве пар вещественных чисел $\mathbb{R}^2$ определены операции

$$+, *\colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$$

следующим образом:

$$\begin{aligned} (a,b) + (c,d) &\stackrel{\text{Def}}{=} (a + c, b + d), \\ (a,b) * (c,d) &\stackrel{\text{Def}}{=} (ac, bd). \end{aligned}$$

Доказать, что $\mathcal{K} \stackrel{\text{Def}}{=} \langle \mathbb{R}^2; +, * \rangle$ образует коммутативное кольцо.

2.4. Показать, что множество вещественных чисел $\mathbb{R}$ с операцией $+$ является векторным пространством над полем рациональных чисел $\mathbb{Q}$, где операция умножения на скаляр — это обычное умножение. Доказать, что это пространство не имеет базиса.

2.5. Доказать, что булевы алгебры из примеров 1 и 3 подраздела 2.5.5 изоморфны, если $|M| = |T|$.

2.6. Доказать, что семейство подмножеств $\mathcal{B} \subset 2^E$ является базисом некоторого матроида $\mathcal{E}$ над множеством E тогда и только тогда, когда

$$\forall B_1, B_2 \in \mathcal{B} \ (e_1 \in B_1 \setminus B_2 \Longrightarrow \exists e_2 \in B_2 \setminus B_1 \ ((B_1 + e_1 + e_2) \in \mathcal{E})).$$

Глава 3 Булевы функции

Данная глава имеет двойное назначение. С одной стороны, помимо основных фактов из теории булевых функций здесь затрагиваются весьма общие понятия, такие как реализация функций формулами, нормальные формы, двойственность, полнота. Эти понятия затруднительно описать исчерпывающим образом на выбранном элементарном уровне изложения, но знакомство с ними необходимо. Поэтому рассматриваются частные случаи указанных понятий на простейшем примере булевых функций. С другой стороны, материал этой главы служит для «накопления фактов» и овладения базисной логической техникой, что должно создать у читателя необходимый эффект «узнавания знакомого» при изучении довольно абстрактного и формального материала следующей главы.

3.1. Элементарные булевы функции

Подобно тому как в классической математике знакомство с основами анализа начинают с изучения элементарных функций (рациональные функции от вещественной переменной x, e^x, $\ln x$ их суперпозиции), так и изложение теории булевых функций естественно начать с выделения, идентификации и изучения элементарных булевых функций (дизъюнкция, конъюнкция и т. д.).

3.1.1. Функции алгебры логики

Функции $f\colon E_2^n \to E_2$, где $E_2 \stackrel{\text{Def}}{=} \{0,1\}$, называются *функциями алгебры логики*, или *булевыми функциями* от n переменных, по имени Дж. Буля[1]. Множество булевых функций от n переменных обозначим P_n, $P_n \stackrel{\text{Def}}{=} \{f \mid f\colon E_2^n \to E_2\}$.

Булеву функцию от n переменных можно задать *таблицей истинности*:

[1] Джордж Буль (1815–1864).

x_1	...	x_{n-1}	x_n	$f(x_1,\ldots,x_n)$
0	...	0	0	$f(0,\ldots,0,0)$
0		0	1	$f(0,\ldots,0,1)$
0		1	0	$f(0,\ldots,1,0)$
...		...	...	...
1	...	1	1	$f(1,\ldots,1,1)$

Если число переменных равно n, то в таблице истинности имеется 2^n строк, соответствующих всем различным комбинациям значений переменных. Следовательно, существует 2^{2^n} различных столбцов, каждый из которых определяет булеву функцию от n переменных. Таким образом, число булевых функций от n переменных с ростом n растет весьма быстро:

$$|P_n| = 2^{2^n}.$$

ЗАМЕЧАНИЕ

Если нужно задать несколько (например, k) булевых функций от n переменных, то это удобно сделать с помощью одной таблицы, использовав несколько столбцов:

x_1	...	x_n	$f_1(x_1,\ldots,x_n)$	...	$f_k(x_1,\ldots,x_n)$
0	...	0	$f_1(0,\ldots,0)$	...	$f_k(0,\ldots,0)$
...		...	...	...	...
1	...	1	$f_1(1,\ldots,1)$	...	$f_k(1,\ldots,1)$

Такая таблица будет иметь 2^n строк, n столбцов для значений переменных и ещё k столбцов для значений функций. Вообще говоря, значения переменных можно не хранить, если принять соглашение о перечислении наборов переменных в определённом порядке. Во всех таблицах истинности в этом учебнике переменные всегда перечисляются в лексикографическом порядке, а кортежи булевых значений — в порядке возрастания целых чисел, задаваемых кортежами как двоичными шкалами (см. алгоритм 1.1), что совпадает с лексикографическим порядком на кортежах. Такой порядок мы называем *установленным*.

Если $k > 2^n$ (что, как показывает предыдущая формула, не редкость), то таблицу истинности можно «транспонировать», выписывая наборы значений в столбцах, а значения функций — в строках:

x_1	0	...	1
...	...		...
x_n	0	...	1
f_1	$f_1(0,\ldots,0)$	...	$f_1(1,\ldots,1)$
...	...		...
f_k	$f_k(0,\ldots,0)$	...	$f_k(1,\ldots,1)$

Именно такой способ записи таблицы истинности использован в подразделах 3.1.3 и 3.1.4.

3.1.2. Существенные и несущественные переменные

Булева функция $f \in P_n$ *существенно зависит* от переменной x_i, если существует такой набор значений $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n$, что

$$f(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n) \neq f(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n).$$

В этом случае x_i называют *существенной переменной*, в противном случае x_i называют *несущественной (фиктивной) переменной.*

Пример Пусть булевы функции $f_1(x_1, x_2)$ и $f_2(x_1, x_2)$ заданы таблицей истинности:

x_1	x_2	f_1	f_2
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

Для этих функций переменная x_1 — существенная, а переменная x_2 — несущественная.

Пусть заданы две булевы функции $f_1(x_1, \ldots, x_{n-1})$ и $f_2(x_1, \ldots, x_{n-1}, x_n)$ и пусть переменная x_n — несущественная для функции f_2, а при одинаковых значениях остальных переменных значения функций совпадают:

$$\forall a_1, \ldots, a_{n-1}, a_n \ (f_1(a_1, \ldots, a_{n-1}) = f_2(a_1, \ldots, a_{n-1}, a_n)).$$

В таком случае говорят, что f_2 получается из f_1 *введением* несущественной переменной x_n, а f_1 получается из f_2 *удалением* несущественной переменной x_n.

ЗАМЕЧАНИЕ

Процедурно введение и удаление несущественных переменных выполняются достаточно просто. Чтобы ввести несущественную переменную, нужно продублировать каждую строку таблицы истинности, добавить новый столбец и заполнить этот столбец чередующимися значениями 0 и 1 (или 1 и 0, что несущественно). Удаление несущественной переменной выполняется аналогично: нужно отсортировать таблицу истинности так, чтобы она состояла из пар строк, различающихся только в разряде несущественной переменной, после чего удалить столбец несущественной переменной и удалить каждую вторую строку таблицы.

Всюду в дальнейшем булевы функции рассматриваются с точностью до несущественных переменных. Это позволяет считать, что все булевы функции (в данной системе функций) зависят от одних и тех же переменных. Для этого в данной системе булевых функций можно сначала удалить все несущественные переменные, а потом добавить несущественные переменные так, чтобы уравнять количество переменных у всех функций.

3.1.3. Булевы функции одной переменной

В следующей таблице собраны все булевы функции одной переменной. Две из них, фактически, являются константами, поскольку их значения не зависят от значения аргумента.

	Переменная x	0	1	
Название	Обозначение			Несущественные
Нуль	0	0	0	x
Тождественная	x	0	1	
Отрицание	$\neg x$, $\bar{x}$, x', $\sim x$	1	0	
Единица	1	1	1	x

3.1.4. Булевы функции двух переменных

В следующей таблице собраны все булевы функции двух переменных. Из них две являются константами, четыре зависят от одной переменной и только десять существенно зависят от обеих переменных.

	Переменная x	0	0	1	1	
	Переменная y	0	1	0	1	
Название	Обозначение					Несущественные
Нуль	0	0	0	0	0	x, y
Конъюнкция	$\cdot$, &, $\wedge$	0	0	0	1	
		0	0	1	0	
		0	0	1	1	y
		0	1	0	0	
		0	1	0	1	x
Сложение по модулю 2	$+$, $+_2$, $\not\equiv$, $\oplus$, Δ	0	1	1	0	
Дизъюнкция	$\vee$	0	1	1	1	
Стрелка Пирса[1]	$\downarrow$	1	0	0	0	
Эквивалентность	$\equiv$	1	0	0	1	
		1	0	1	0	x
		1	0	1	1	
		1	1	0	0	y
Импликация	$\rightarrow$, $\Rightarrow$, $\supset$	1	1	0	1	
Штрих Шеффера[2]	$\mid$	1	1	1	0	
Единица	1	1	1	1	1	x, y

ЗАМЕЧАНИЕ

Пустоты в столбцах «Название» и «Обозначение» в предыдущей таблице означают, что булева функция редко используется, а потому не имеет специального названия и обозначения.

[1] Чарльз Сандерс Пирс (1839–1914).

[2] Генри М. Шеффер (1882–1964).

3.2. Формулы

В этом разделе обсуждается целый ряд важных понятий, которые часто считаются самоочевидными, а потому их объяснение опускается. Такими понятиями являются, в частности, реализация функций формулами, интерпретация формул для вычисления значений функций, равносильность формул, подстановка и замена в формулах. Между тем программная реализация работы с формулами требует учёта некоторых тонкостей, связанных с данными понятиями, которые целесообразно явно указать и обсудить.

3.2.1. Реализация функций формулами

Начнём обсуждение с привычного понятия *формулы*. Вообще говоря, формула — это цепочка символов, имеющих заранее оговорённый смысл. Обычно формулы строятся по определённым правилам из обозначений объектов и вспомогательных символов — разделителей. Применительно к рассматриваемому случаю объектами являются переменные и булевы функции, перечисленные в таблицах подразделов 3.1.3 и 3.1.4, а разделителями — скобки «(», «)» и запятая «,». Мы считаем, что обозначения всех объектов заранее определены и синтаксически отличимы друг от друга и от разделителей, а правила составления формул общеизвестны.

ЗАМЕЧАНИЕ

Для обозначения объектов используются как отдельные символы, так и группы символов, в том числе со специальным начертанием: верхние и нижние индексы, наклон шрифта и т. п. Допущение о синтаксической различимости остаётся в силе для всех таких особенностей.

Пример Операцию сложения вещественных чисел принято обозначать знаком «+», $+\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, а функцию «синус» — словом «sin», которое записывается прямым шрифтом и считается одним символом, $\sin\colon \mathbb{R} \to \mathbb{R}$.

ОТСТУПЛЕНИЕ

В математических текстах часто используется *нелинейная* форма записи формул, при которой формула *не* является цепочкой символов. Примерами могут служить дроби, радикалы, суммы, интегралы. Подобные нелинейные формы записи формул привычны и удобны. Использование нелинейной записи формул не является принципиальным расширением языка формул. Можно ограничиться только линейными цепочками символов, что блестяще подтверждает система TEX, с помощью которой подготовлена эта книга.

Пусть $F = \{f_1, \ldots, f_m\}$ — некоторое множество булевых функций от n переменных. *Формулой* $\mathcal{F}$ над F называется выражение (цепочка символов) вида

$$\mathcal{F}[F] = f(t_1, \ldots, t_n),$$

где $f \in F$ и t_i — либо переменная, либо формула над F. Множество F называется *базисом*, функция f называется *главной* (*внешней*) операцией (функцией), а t_i называются *подформулами*. Если базис F ясен из контекста, то его обозначение опускают.

ЗАМЕЧАНИЕ

Обычно при записи формул, составленных из элементарных булевых функций, обозначения бинарных булевых функций записываются как знаки операций в инфиксной форме, отрицание записывается как знак унарной операции в префиксной форме, тождественные функции записываются как константы 0 и 1. Кроме того, устанавливается приоритет операций ($\neg$, $\&$, $\vee$, $\rightarrow$) и лишние скобки опускаются.

Всякой формуле $\mathcal{F}$ однозначно соответствует некоторая функция f. Это соответствие задается алгоритмом *интерпретации*, который позволяет вычислить значение формулы $\mathcal{F}$ при заданных значениях переменных.

Алгоритм 3.1 Интерпретация формул — рекурсивная функция Eval

Вход: формула $\mathcal{F}$, множество F функций базиса,
значения переменных $x_1, \ldots, x_n$.

Выход: значение формулы $\mathcal{F}$ на значениях $x_1, \ldots, x_n$ или значение **fail**, если значение формулы не может быть определено.

```
if F = 'x_i' then
  return x_i { значение переменной задано }
end if
if F = 'f(t_1, ..., t_n)' then
  if f ∉ F then
    return fail { функция не входит в базис }
  end if
  for t ∈ {t_1, ..., t_n} do
    y_i := Eval (t_i, F, x_1, ..., x_n) { значение i-го аргумента }
    if y_i = fail then return fail end if
  end for
  return f(y_1, ..., y_n) { вычисленное значение главной операции }
end if
return fail { это не формула }
```

ОТСТУПЛЕНИЕ

Некоторые программистские замечания по поводу процедуры Eval.

1. При программной реализации алгоритма интерпретации формул важно учитывать, что в общем случае результат вычисления значения формулы может быть не определён (**fail**). Это имеет место, если формула построена синтаксически неправильно или если в ней используются функции (операции), способ вычисления которых не задан, то есть они не входят в базис. Таким образом, необходимо либо проверять правильность формулы до начала работы алгоритма интерпретации, либо предусматривать невозможность вычисления значения в самом алгоритме.

2. Это не единственный возможный алгоритм вычисления значения формулы, более того, он не самый лучший. Для конкретных классов формул, например, для некоторых классов формул, реализующих булевы функции, известны более эффективные алгоритмы (например, алгоритм 3.4).
3. Сама идея этого алгоритма: «сначала вычисляются значения аргументов, а потом значение функции»,— не является догмой. Например, можно построить такой алгоритм интерпретации, который вычисляет значения только *некоторых* аргументов, а потом вычисляет значение формулы, в результате чего получается новая *формула*, реализующая функцию, которая зависит от меньшего числа аргументов. Такой алгоритм интерпретации называется *смешанными вычислениями*.
4. Порядок вычисления аргументов *считается* неопределённым, то есть предполагается, что базисные функции не имеют *побочных эффектов*. Если же базисные функции имеют побочные эффекты, то результат вычисления значения функции может зависеть от порядка вычисления значений аргументов. Побочные эффекты могут иметь различные причины. Наиболее частая: использование глобальных переменных. Например, если

$$f(x) \stackrel{\text{Def}}{=} a := a + 1; \textbf{return } x + a,$$

$$g(x) \stackrel{\text{Def}}{=} a := a * 2; \textbf{return } x * a,$$

причём переменная a глобальна, то (если $a \neq 6$)

$$f(2) + g(3) \neq g(3) + f(2).$$

Если формула $\mathcal{F}$ и базис F заданы (причём $\mathcal{F}$ является правильно построенной формулой над базисом F), то процедура $\text{Eval}(\mathcal{F}, F, x_1, \ldots, x_n)$ является некоторой булевой функцией f переменных $x_1, \ldots, x_n$. В этом случае говорят, что формула $\mathcal{F}$ *реализует* функцию f: $\text{func}\,\mathcal{F} = f$.

ЗАМЕЧАНИЕ

Для обозначения реализуемости применяют и другие приёмы. Выбранное обозначение обладает тем достоинством, что согласовано с другими обозначениями в книге.

Зная таблицы истинности для функций базиса, можно вычислить таблицу истинности той функции, которую реализует данная формула.

Примеры

1. $F_1 := (x_1 \wedge x_2) \vee ((x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2))$

x_1	x_2	$x_1 \wedge \overline{x}_2$	$\overline{x}_1 \wedge x_2$	$(x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2)$	$x_1 \wedge x_2$	F_1
0	0	0	0	0	0	0
0	1	0	1	1	0	1
1	0	1	0	1	0	1
1	1	0	0	0	1	1

Таким образом, формула F_1 реализует дизъюнкцию.

2. $F_2 := (x_1 \wedge x_2) \to x_1$

x_1	x_2	$x_1 \wedge x_2$	F_2
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

Таким образом, формула F_2 реализует константу 1.

3. $F_3 := ((x_1 \wedge x_2) + x_1) + x_2$

x_1	x_2	$x_1 \wedge x_2$	$(x_1 \wedge x_2) + x_1$	$((x_1 \wedge x_2) + x_1) + x_2$
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	0	1

Таким образом, формула F_3 также реализует дизъюнкцию.

3.2.2. Равносильные формулы

Одна функция может иметь множество реализаций (над данным базисом). Формулы, реализующие одну и ту же функцию, называются *равносильными*:

$$\mathcal{F}_1 = \mathcal{F}_2 \stackrel{\text{Def}}{=} \exists f \ (\text{func}\,\mathcal{F}_1 = f \ \& \ \text{func}\,\mathcal{F}_2 = f)\,.$$

Отношение равносильности формул является эквивалентностью. Имеют место, в частности, следующие равносильности:

1. $a \vee a = a$, $a \wedge a = a$.
2. $a \vee b = b \vee a$, $a \wedge b = b \wedge a$.
3. $a \vee (b \vee c) = (a \vee b) \vee c$, $a \wedge (b \wedge c) = (a \wedge b) \wedge c$.
4. $(a \wedge b) \vee a = a$, $(a \vee b) \wedge a = a$.
5. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$, $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.
6. $a \vee 1 = 1$, $a \wedge 0 = 0$.
7. $a \vee 0 = a$, $a \wedge 1 = a$.
8. $\neg\neg a = a$.
9. $\neg(a \wedge b) = \neg a \vee \neg b$, $\neg(a \vee b) = \neg a \wedge \neg b$.
10. $a \vee \neg a = 1$, $a \wedge \neg a = 0$.

Все они могут быть проверены построением соответствующих таблиц истинности. Таким образом, $\langle E_2; \vee, \wedge, \neg \rangle$ — булева алгебра (см. 2.5.5).

ЗАМЕЧАНИЕ

Ввиду выполнения равносильностей 2 и 3 для кратных дизъюнкций и конъюнкций используются следующие обозначения:

$$\bigvee_{i=1}^{n} x_i \stackrel{\text{Def}}{=} x_1 \vee \ldots \vee x_n, \qquad \bigwedge_{i=1}^{n} x_i \stackrel{\text{Def}}{=} x_1 \wedge \cdots \wedge x_n.$$

3.2.3. Подстановка и замена

Если в формулу $\mathcal{F}$ входит переменная x, то это обстоятельство обозначается так: $\mathcal{F}(\ldots x \ldots)$. Соответственно, запись $\mathcal{F}(\ldots \mathcal{G} \ldots)$ обозначает, что в формулу $\mathcal{F}$ входит подформула $\mathcal{G}$. Вместо подформулы (в частности, вместо переменной) в формулу можно подставить другую формулу (в частности, переменную), в результате чего получится новая правильно построенная формула. Наряду с оборотом «подставить подформулу в формулу» используется оборот «заменить подформулу формулой». Если подстановка формулы $\mathcal{G}$ производится вместо *всех* вхождений заменяемой переменной x (или подформулы), то результат подстановки обозначается следующим образом: $\mathcal{F}(\ldots x \ldots)\{\mathcal{G}//x\}$. Если же подстановка производится вместо *некоторых* вхождений (в том числе вместо одного), то результат подстановки обозначается следующим образом: $\mathcal{F}(\ldots \mathcal{G}_1 \ldots)\{\mathcal{G}_2/\mathcal{G}_1\}$.

Примеры

1. Замена всех вхождений переменной: $x \vee \neg x\{y \wedge z//x\} = (y \wedge z) \vee \neg(y \wedge z)$.
2. Замена всех вхождений подформулы: $x \vee y \vee z\{\neg x//y \vee z\} = x \vee \neg x$.
3. Замена первого вхождения переменной: $x \vee \neg x\{y/x\} = y \vee \neg x$.
4. Замена первого вхождения подформулы: $x \vee y \vee z\{\neg x/y \vee z\} = x \vee \neg x$.

Известны два правила: *правило подстановки* и *правило замены*,— которые позволяют преобразовывать формулы с сохранением равносильности.

ТЕОРЕМА 1 (Правило подстановки) *Если в двух равносильных формулах вместо всех вхождений некоторой переменной x подставить одну и ту же формулу, то получатся равносильные формулы:*

$$\forall \mathcal{G}\ (\mathcal{F}_1(\ldots x \ldots) = \mathcal{F}_2(\ldots x \ldots) \Longrightarrow \mathcal{F}_1(\ldots x \ldots)\{\mathcal{G}//x\} = \mathcal{F}_2(\ldots x \ldots)\{\mathcal{G}//x\}).$$

Доказательство Чтобы доказать равносильность двух формул, нужно показать, что они реализуют одну и ту же функцию. А это можно сделать, если взять произвольный набор значений переменных и убедиться, что значения, полученные при вычислении формул, совпадают.

Рассмотрим произвольный набор значений $a_1, \ldots, a_x, \ldots, a_n$ переменных $x_1, \ldots, x, \ldots, x_n$. Обозначим $a := \mathrm{Eval}(\mathcal{G}, F, a_1, \ldots, a_x, \ldots, a_n)$. По определению алгоритма интерпретации

$$\mathrm{Eval}(\mathcal{F}_1\{\mathcal{G}//x\}, F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_1, F, a_1, \ldots, a, \ldots, a_n)$$

и, аналогично,

$$\mathrm{Eval}(\mathcal{F}_2\{\mathcal{G}//x\}, F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2, F, a_1, \ldots, a, \ldots, a_n).$$

Но $\mathcal{F}_1 = \mathcal{F}_2$, и значит,

$$\mathrm{Eval}(\mathcal{F}_1, F, a_1, \ldots, a, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2, F, a_1, \ldots, a, \ldots, a_n),$$

откуда

$$\mathrm{Eval}(\mathcal{F}_1\{\mathcal{G}//x\}, F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2\{\mathcal{G}//x\}, F, a_1, \ldots, a_x, \ldots, a_n). \quad \square$$

ЗАМЕЧАНИЕ

В правиле подстановки условие замены *всех* вхождений существенно: например, $x \vee \neg x = 1$ и $x \vee \neg x\{y//x\} = y \vee \neg y = 1$, но $x \vee \neg x\{y/x\} = y \vee \neg x \neq 1$!

ТЕОРЕМА 2 (Правило замены) *Если в формуле заменить некоторую подформулу равносильной формулой, то получится формула, равносильная исходной:*

$$\forall \mathcal{F}(\ldots \mathcal{G}_1 \ldots)\ (\mathcal{G}_1 = \mathcal{G}_2 \Longrightarrow \mathcal{F}(\ldots \mathcal{G}_1 \ldots) = \mathcal{F}(\ldots \mathcal{G}_1 \ldots)\{\mathcal{G}_2/\mathcal{G}_1\}).$$

Доказательство Рассмотрим произвольный набор значений $a_1, \ldots, a_n$ переменных $x_1, \ldots, x_n$. Имеем $\mathcal{G}_1 = \mathcal{G}_2$, и значит,

$$\mathrm{Eval}(\mathcal{G}_1, F, a_1, \ldots, a_n) = \mathrm{Eval}(\mathcal{G}_2, F, a_1, \ldots, a_n),$$

откуда

$$\mathrm{Eval}(\mathcal{F}\{\mathcal{G}_1/\mathcal{G}_1\}, F, a_1, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}\{\mathcal{G}_2/\mathcal{G}_1\}, F, a_1, \ldots, a_n). \quad \square$$

Пусть $F = \{f_1, \ldots, f_m\}$ и $G = \{g_1, \ldots, g_m\}$. Тогда говорят, что формулы $\mathcal{F}[F]$ и $\mathcal{G}[G]$ имеют *одинаковое строение*, если $\mathcal{F}$ совпадает с результатами подстановки в формулу $\mathcal{G}$ функций f_i вместо функций g_i:

$$\mathcal{F}[F] = \mathcal{G}[G]\{f_i//g_i\}_{i=1}^m.$$

3.2.4. Алгебра булевых функций

Булевы функции $\vee$, $\wedge$, $\neg$ (и любые другие) могут рассматриваться как операции на множестве булевых функций, $\vee, \wedge\colon P_n \times P_n \to P_n$, $\neg\colon P_n \to P_n$.

Действительно, пусть формулы $\mathcal{F}_1$ и $\mathcal{F}_2$ равносильны и реализуют функцию f, а формулы $\mathcal{G}_1$ и $\mathcal{G}_2$ равносильны и реализуют функцию g:

$$\mathrm{func}\,\mathcal{F}_1 = f, \qquad \mathrm{func}\,\mathcal{F}_2 = f, \qquad \mathrm{func}\,\mathcal{G}_1 = g, \qquad \mathrm{func}\,\mathcal{G}_2 = g.$$

Тогда, применяя правило замены нужное число раз, имеем:

$$\mathcal{F}_1 \vee \mathcal{G}_1 = \mathcal{F}_2 \vee \mathcal{G}_2, \quad \mathcal{F}_1 \wedge \mathcal{G}_1 = \mathcal{F}_2 \wedge \mathcal{G}_2, \quad \neg\mathcal{F}_1 = \neg\mathcal{F}_2.$$

Таким образом, если взять любые формулы $\mathcal{F}$ и $\mathcal{G}$, реализующие функции f и g соответственно, то каждая из формул $\mathcal{F} \wedge \mathcal{G}$, $\mathcal{F} \vee \mathcal{G}$ и $\neg\mathcal{F}$ реализует одну и ту

же функцию, независимо от выбора реализующих формул $\mathcal{F}$ и $\mathcal{G}$. Следовательно, функции, которые реализуются соответствующими формулами, можно по определению считать результатами применения соответствующих операций. Другими словами, если

$$\operatorname{func}\mathcal{F} = f, \qquad \operatorname{func}\mathcal{G} = g,$$

то

$$f \wedge g \stackrel{\text{Def}}{=} \operatorname{func}(\mathcal{F} \wedge \mathcal{G}), \qquad f \vee g \stackrel{\text{Def}}{=} \operatorname{func}(\mathcal{F} \vee \mathcal{G}), \qquad \neg f \stackrel{\text{Def}}{=} \operatorname{func}(\neg\mathcal{F}).$$

Алгебраическая структура $\langle P_n; \vee, \wedge, \neg\rangle$ называется *алгеброй булевых функций.*

ТЕОРЕМА *Алгебра булевых функций является булевой алгеброй.*

Доказательство Действительно, пусть равносильности, перечисленные в подразделе 3.2.2, проверены путем построения таблиц истинности. Ясно, что эти таблицы не зависят от того, откуда взялись значения a, b, c. Таким образом, вместо a, b, c можно подставить любые функции, а значит, любые реализующие их формулы, если только выполнено правило подстановки. Следовательно, аксиомы булевой алгебры выполнены в алгебре $\langle P_n; \vee, \wedge, \neg\rangle$. □

Пусть $[\mathcal{F}]$ — множество формул, равносильных $\mathcal{F}$ (то есть класс эквивалентности по отношению равносильности). Рассмотрим множество $\mathcal{K}$ классов эквивалентности по отношению равносильности: $\mathcal{K} \stackrel{\text{Def}}{=} \{[\mathcal{F}]\}_{\mathcal{F}}$. Пусть операции

$$\vee, \wedge\colon \mathcal{K} \times \mathcal{K} \to \mathcal{K}, \quad \neg\colon \mathcal{K} \to \mathcal{K}$$

определены (на множестве классов эквивалентности формул по отношению равносильности) следующим образом:

$$[\mathcal{F}_1] \vee [\mathcal{F}_2] \stackrel{\text{Def}}{=} [\mathcal{F}_1 \vee \mathcal{F}_2], \ [\mathcal{F}_1] \wedge [\mathcal{F}_2] \stackrel{\text{Def}}{=} [\mathcal{F}_1 \wedge \mathcal{F}_2], \ \neg[\mathcal{F}_1] \stackrel{\text{Def}}{=} [\neg\mathcal{F}_2].$$

Тогда алгебра классов равносильных формул $\langle \mathcal{K}; \wedge, \vee, \neg\rangle$ (*алгебра Линденбаума–Тарского*[1]) изоморфна алгебре булевых функций и является булевой алгеброй. Носитель этой алгебры — множество классов формул.

ОТСТУПЛЕНИЕ

На практике мы говорим о функциях, а пишем формулы, хотя формулы и функции — разные вещи. Например, формул бесконечно много, а функций (булевых функций n переменных) — только конечное число, и свободная алгебра формул *не изоморфна* алгебре функций. Но алгебра функций *изоморфна* алгебре классов равносильных формул, что позволяет манипулировать формулами, имея в виду функции.

[1] Альфред Тарский (1902–1984).

3.3. Двойственность

Понятие двойственности с успехом применяется в самых разных областях математики. Мы рассматриваем двойственность на простейшем примере булевых функций.

3.3.1. Двойственная функция

Пусть $f(x_1,\ldots,x_n) \in P_n$ — булева функция. Тогда функция $f^*(x_1,\ldots,x_n)$, определённая следующим образом:

$$f^*(x_1,\ldots,x_n) \stackrel{\text{Def}}{=} \overline{f(\overline{x}_1,\ldots,\overline{x}_n)},$$

называется *двойственной* к функции f.

Из определения легко видно, что двойственность инволютивна: $f^{**} = f$, и по этой причине отношение «быть двойственной к» на множестве булевых функций симметрично, то есть если $f^* = g$, то $g^* = f$.

Если в таблице истинности булевой функции f инвертировать *все* значения, то получим таблицу истинности двойственной функции f^*.

Пример

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$(x_1 \wedge x_2)^*$
1	1	1
1	0	1
0	1	1
0	0	0

$=$

x_1	x_2	$(x_1 \wedge x_2)^*$
0	0	0
0	1	1
1	0	1
1	1	1

Таким способом можно определить двойственную функцию для любой булевой функции.

Пример Двойственные функции:

f	1	0	$x_1 \vee x_2$	$x_1 \wedge x_2$	x	$\overline{x}$
f^*	0	1	$x_1 \wedge x_2$	$x_1 \vee x_2$	x	$\overline{x}$

Функция называется *самодвойственной*, если $f^* = f$.

Пример Тождественная функция и отрицание самодвойственны, а дизъюнкция и конъюнкция — нет.

3.3.2. Реализация двойственной функции

Формула, реализующая двойственную функцию, определённым образом связана с формулой, реализующей исходную функцию.

ЗАМЕЧАНИЕ

Далее используется обозначение $\overline{f}(\cdots) \stackrel{\text{Def}}{=} \overline{f(\cdots)}$.

ТЕОРЕМА *Если функция* $\varphi(x_1,\ldots,x_n)$ *реализована формулой*

$$f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n)),$$

то формула

$$f^*(f_1^*(x_1,\ldots,x_n),\ldots,f_n^*(x_1,\ldots,x_n))$$

реализует функцию $\varphi^*(x_1,\ldots,x_n)$.

Доказательство

$$\begin{aligned}\varphi^*(x_1,\ldots,x_n) &= \overline{\varphi}(\overline{x}_1,\ldots,\overline{x}_n) = \overline{f}(f_1(\overline{x}_1,\ldots,\overline{x}_n),\ldots,f_n(\overline{x}_1,\ldots,\overline{x}_n)) = \\ &= \overline{f}(\overline{\overline{f_1}}(\overline{x}_1,\ldots,\overline{x}_n),\ldots,\overline{\overline{f_n}}(\overline{x}_1,\ldots,\overline{x}_n)) = \\ &= \overline{f}(\overline{f_1^*}(x_1,\ldots,x_n),\ldots,\overline{f_n^*}(x_1,\ldots,x_n)) = \\ &= f^*(f_1^*(x_1,\ldots,x_n),\ldots,f_n^*(x_1,\ldots,x_n)).\end{aligned}$$

□

3.3.3. Принцип двойственности

Принцип двойственности устанавливает связь между структурами формул, реализующих пару двойственных функций. Рассмотрим две системы булевых функций $F = \{f_1,\ldots,f_m\}$ и $F^* = \{f_1^*,\ldots,f_m^*\}$ и введём обозначение $\mathcal{F}^*[F^*] \stackrel{\text{Def}}{=} \stackrel{\text{Def}}{=} \mathcal{F}[F]\{f_i^*//f_i\}_{i=1}^m$.

ТЕОРЕМА (Принцип двойственности) *Пусть* $F = \{f_1,\ldots,f_m\}$ *— система булевых функций, а* $F^* = \{f_1^*,\ldots,f_m^*\}$ *— система двойственных функций. Тогда если формула* $\mathcal{F}$ *над базисом* F *реализует функцию* f*, то формула* $\mathcal{F}^*$ *над базисом* F^**, полученная заменой функций* f_i *двойственными функциями* f_i^**, реализует функцию* f^**:*

$$\operatorname{func}\mathcal{F}[F] = f \Longrightarrow \operatorname{func}\mathcal{F}^*[F^*] = f^*.$$

Доказательство Индукция по структуре формулы $\mathcal{F}$. База: если формула $\mathcal{F}$ имеет вид $f(x_1,\ldots,x_n)$, где $f \in F$, то формула $\mathcal{F}^* = f^*(x_1,\ldots,x_n)$ реализует функцию f^* по определению. Индукционный переход по предыдущей теореме. □

ОТСТУПЛЕНИЕ

Хорошо известен принцип математической индукции для натуральных чисел:

$$(P(1)\ \&\ (P(n) \Longrightarrow P(n+1))) \Longrightarrow \forall n \in \mathbb{N}\ (P(n)).$$

Этот принцип является справедливым и для других множеств, упорядоченных более сложным образом, нежели натуральные числа (см. 1.8.7). Например, в доказательстве предыдущей теоремы был использован принцип индукции в следующей форме.

Пусть задана некоторая иерархия (ориентированное дерево, см. 9.2.1). Предположим, что

1) некоторое утверждение P справедливо для всех узлов иерархии нижнего уровня (листьев дерева);
2) из того, что утверждение P справедливо для всех узлов, подчиненных данному узлу, следует, что утверждение P справедливо для данного узла.

Тогда утверждение P справедливо для всех узлов иерархии (дерева).

СЛЕДСТВИЕ $\mathcal{F}_1 = \mathcal{F}_2 \Longrightarrow \mathcal{F}_1{}^* = \mathcal{F}_2{}^*$.

Пример Из $\overline{x_1 \wedge x_2} = \overline{x}_1 \vee \overline{x}_2$ по принципу двойственности сразу имеем $\overline{x_1 \vee x_2} = \overline{x}_1 \wedge \overline{x}_2$.

3.4. Нормальные формы

В данном разделе на примере булевых функций обсуждается важное понятие «нормальной формы», то есть синтаксически однозначного способа записи формулы, реализующей заданную функцию.

3.4.1. Разложение булевых функций по переменным

Пусть $x^y \stackrel{\text{Def}}{=} x \wedge y \vee \overline{x} \wedge \overline{y}$. Очевидно, что

$$x^y = \begin{cases} \overline{x}, & \text{если } y = 0, \\ x, & \text{если } y = 1, \end{cases} \qquad x^y = \begin{cases} 1, & \text{если } x = y, \\ 0, & \text{если } x \neq y, \end{cases} \qquad x^y = x \equiv y.$$

ТЕОРЕМА (О разложении булевой функции по переменным)

$$f(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) = \bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \wedge \ldots \wedge x_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n),$$

где дизъюнкция берётся по всем возможным наборам значений $(\sigma_1, \ldots, \sigma_m)$.

Доказательство Рассмотрим значение формулы в правой части на наборе значений $a_1, \ldots, a_n$. Имеем

$$\left(\bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \wedge \ldots \wedge x_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n) \right) (a_1, \ldots, a_n) = \bigvee_{(\sigma_1, \ldots, \sigma_m)} a_1^{\sigma_1} \wedge \ldots \wedge a_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, a_{m+1}, \ldots, a_n).$$

Все конъюнкции, в которых $\exists i \; a_i \neq \sigma_i$ равны 0 и их можно опустить, поэтому в дизъюнкции остаётся только одно слагаемое, для которого $\forall i \in 1..n \;\; (a_i = \sigma_i)$, и окончательно имеем

$$a_1^{a_1} \wedge \ldots \wedge a_n^{a_n} \wedge f(a_1, \ldots, a_m, a_{m+1}, \ldots, a_n) = f(a_1, \ldots, a_n). \qquad \square$$

ЗАМЕЧАНИЕ

Здесь доказывается, что некоторая формула реализует заданную функцию. Для этого достаточно взять *произвольный* набор значений аргументов функции, вычислить на этом наборе значение формулы, и если оно окажется равным значению функции на этом наборе аргументов, то из этого следует доказываемое утверждение.

СЛЕДСТВИЕ 1

$$f(x_1,\ldots,x_{n-1},x_n)=(x_n\wedge f(x_1,\ldots,x_{n-1},1))\vee(\overline{x}_n\wedge f(x_1,\ldots,x_{n-1},0)).$$

СЛЕДСТВИЕ 2 $f(x_1,\ldots,x_n)=\bigvee\limits_{\{(\sigma_1,\ldots,\sigma_n)\mid f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1}\wedge\ldots\wedge x_n^{\sigma_n}.$

3.4.2. Совершенные нормальные формы

Реализация булевой функции $f(x_1,\ldots,x_n)$ в виде формулы

$$\bigvee_{\{(\sigma_1,\ldots,\sigma_n)\mid f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1}\wedge\ldots\wedge x_n^{\sigma_n}$$

называется *совершенной дизъюнктивной нормальной формой* (СДНФ).

ЗАМЕЧАНИЕ

СДНФ называется совершенной, потому что каждое слагаемое в дизъюнкции включает все переменные; дизъюнктивной, потому что главная операция — дизъюнкция, а почему она называется нормальной, объяснено далее, в отступлении в конце подраздела.

Если при записи СДНФ используется установленный порядок (см. 3.1.1), то СДНФ однозначно определяет множество наборов значений переменных, на которых функция, реализуемая СДНФ, принимает значение 1. Тем самым СДНФ однозначно определяет таблицу истинности реализуемой функции, а значит, любые две различные СДНФ над одним и тем же набором переменных неравносильны.

ЗАМЕЧАНИЕ

При рассмотрении дизъюнктивных (и конъюнктивных) форм мы имеем дело с формулами вида

$$\bigvee_{i\in I} S_i \quad\text{и}\quad \bigwedge_{i\in I} S_i,$$

где I — некоторое множество индексов, а S_i — некоторые формулы. Множество индексов может быть пусто. В этом случае удобно считать, что пустая дизъюнкция имеет значение 0, а пустая конъюнкция — значение 1:

$$I=\varnothing\Longrightarrow\bigvee_{i\in I} S_i=0,\quad I=\varnothing\Longrightarrow\bigwedge_{i\in I} S_i=1.$$

ТЕОРЕМА 1 *Всякая булева функция имеет единственную СДНФ.*

Доказательство

$$\begin{aligned}f(x_1,\ldots,x_n)&=\bigvee_{\sigma_1,\ldots,\sigma_n} x_1^{\sigma_1}\wedge\ldots\wedge x_n^{\sigma_n}\wedge f(\sigma_1,\ldots,\sigma_n)=\\&=\bigvee_{\{(\sigma_1,\ldots,\sigma_n)\mid f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1}\wedge\ldots\wedge x_n^{\sigma_n}\wedge f(\sigma_1,\ldots,\sigma_n)=\\&=\bigvee_{\{(\sigma_1,\ldots,\sigma_n)\mid f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1}\wedge\ldots\wedge x_n^{\sigma_n}.\end{aligned}$$

□

СЛЕДСТВИЕ *Всякая булева функция может быть выражена через дизъюнкцию, конъюнкцию и отрицание:*

$$\forall f \in P_n \ (\exists \mathcal{F}[\{\vee, \wedge, \neg\}] \ (f = \operatorname{func} \mathcal{F})) .$$

ЗАМЕЧАНИЕ

Если $f = 0$, то $\{(\sigma_1, \ldots, \sigma_n) \mid f(\sigma_1, \ldots, \sigma_n) = 1\} = \varnothing$. В соответствии с соглашением предыдущего замечания пустая формула считается СДНФ нуля.

ТЕОРЕМА 2 *Всякая булева функция имеет единственную совершенную конъюнктивную нормальную форму (СКНФ):*

$$f(x_1, \ldots, x_n) = \bigwedge_{\{(\sigma_1,\ldots,\sigma_n) \mid f^*(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1} \vee \ldots \vee x_n^{\sigma_n}.$$

Доказательство По принципу двойственности из предыдущей теоремы. □

ОТСТУПЛЕНИЕ

Говорят, что некоторый класс формул $\mathcal{K}$ *имеет нормальную форму*, если задан другой класс формул $\mathcal{K}'$, которые называются нормальными формами, такой, что любая формула класса $\mathcal{K}$ имеет *единственную* равносильную формулу из класса $\mathcal{K}'$.

Если задан алгоритм, позволяющий для любой формулы построить её нормальную форму, то наличие у класса формул нормальной формы обеспечивает *разрешимость*, то есть наличие алгоритма проверки равносильности. Действительно, в этом случае достаточно сравнить нормальные формы двух формул.

Один и тот же класс $\mathcal{K}$ может иметь несколько различных нормальных форм, то есть несколько различных классов $\mathcal{K}'$.

Примеры

1. СДНФ и СКНФ являются нормальными формами для булевых формул над любым базисом.
2. Формулы вида

$$\sum_{i=0}^{n} a_i x^i \quad \text{и} \quad (\ldots(a_n x + a_{n-1})x + \ldots + a_1)x + a_0$$

являются нормальными формами для полиномов одной переменной степени n.

3. Множество формул, построенных из рациональных функций одной переменной, e^x и $\ln x$ (то есть множество формул, реализующих элементарные функции математического анализа), нормальной формы не имеет. Доказательство последнего утверждения выходит за рамки этого учебника.

3.4.3. Эквивалентные преобразования

Используя уже доказанные равносильности, можно преобразовывать по правилу замены одни формулы в другие, равносильные им. Преобразование формулы в равносильную называется *эквивалентным преобразованием.*

Пример Используя равносильности из подраздела 3.2.2, покажем, что имеет место *правило склеивания/расщепления*: $(x \wedge y) \vee (x \wedge \overline{y}) = x$. Действительно:

$$(x \wedge y) \vee (x \wedge \overline{y}) \stackrel{5}{=} x \wedge (y \vee \overline{y}) \stackrel{10}{=} x \wedge 1 \stackrel{7}{=} x.$$

ЗАМЕЧАНИЕ

Если равносильность из предыдущего примера применяется для уменьшения числа операций в формуле, то говорят, что производится *склеивание*, а если наоборот, то *расщепление.*

ТЕОРЕМА *Для любых двух равносильных формул $\mathcal{F}_1$ и $\mathcal{F}_2$ существует последовательность эквивалентных преобразований из $\mathcal{F}_1$ в $\mathcal{F}_2$ посредством равносильностей, указанных в подразделе 3.2.2.*

Доказательство Любую формулу (кроме тех, которые реализуют 0) можно преобразовать в СДНФ с помощью равносильностей из подраздела 3.2.2 и правила расщепления из предыдущего примера по следующему алгоритму:

1. *Элиминация операций.* Любая булева функция реализуется формулой над базисом $\{\wedge, \vee, \neg\}$ (например, в виде СДНФ). Таким образом, любая присутствующая в формуле подформула с главной операцией, отличной от дизъюнкции, конъюнкции и отрицания, может быть заменена подформулой, содержащей только три базисные операции. Например, элиминация импликации выполняется с помощью равносильности $x_1 \to x_2 = \neg x_1 \vee x_2$. В результате первого шага в формуле остаются только базисные операции.
2. *Протаскивание отрицаний.* С помощью инволютивности отрицания и правил де Моргана операция отрицания «протаскивается» к переменным. В результате второго шага отрицания могут присутствовать в формуле только непосредственно перед переменными.
3. *Раскрытие скобок.* По дистрибутивности конъюнкции относительно дизъюнкции раскрываются все скобки, являющиеся операндами конъюнкции. В результате третьего шага формула приобретает вид *дизъюнктивной формы*:

$$\bigvee (A_i \wedge \cdots \wedge A_j),$$

   где A_k — это либо переменная, либо отрицание переменной.
4. *Удаление нулей.* Если в слагаемое дизъюнктивной формы входят переменная и её отрицание $(x \wedge \neg x)$, то такое слагаемое удаляется. Если при этом формула оказывается пустой, то процесс прерывается и считается завершённым (исходная формула реализует 0).

5. *Приведение подобных.* С помощью идемпотентности конъюнкции удаляются повторные вхождения переменных в каждую конъюнкцию, а затем с помощью идемпотентности дизъюнкции удаляются повторные вхождения одинаковых конъюнкций в дизъюнкцию. В результате пятого шага формула не содержит «лишних» переменных и «лишних» конъюнктивных слагаемых.
6. *Расщепление переменных.* По правилу расщепления в каждую конъюнкцию, которая содержит не все переменные, добавляются недостающие. В результате шестого шага формула становится «совершенной», то есть в каждой конъюнкции содержатся все переменные.
7. *Сортировка.* С помощью коммутативности переменные в каждой конъюнкции, а затем конъюнкции в дизъюнкции сортируются в установленном порядке (см. 3.1.1 и 3.6.2). В результате седьмого шага формула приобретает вид СДНФ.

Заметим, что все указанные преобразования обратимы.

Если теперь даны две формулы, преобразуем их в СДНФ указанным алгоритмом. Если результаты не совпали, значит, формулы не равносильны и эквивалентное преобразование одной в другую невозможно. Если же результаты совпали, то, применяя обратные преобразования в обратном порядке, преобразуем полученную СДНФ во вторую формулу. Объединяя последовательность преобразований первой формулы в СДНФ и обратных преобразований СДНФ во вторую формулу, имеем искомую последовательность преобразований. □

3.4.4. Минимальные дизъюнктивные формы

Булева функция может быть задана бесконечным числом различных, но равносильных формул. Возникает естественная задача: для данной булевой функции найти реализующую формулу, обладающую теми или иными свойствами.

Практически наиболее востребованной оказалась задача минимизации: найти минимальную формулу, реализующую функцию. Но и в этой постановке задача имеет множество вариантов:

1. Найти реализующую формулу, содержащую наименьшее количество переменных.
2. Найти реализующую формулу, содержащую наименьшее количество определённых операций.
3. Найти реализующую формулу, содержащую наименьшее количество подформул определённого вида.

Кроме того, могут быть наложены ограничения на синтаксический вид искомой формулы, набор операций, которые разрешается использовать в формуле, и т. д. Из всего этого разнообразия наиболее детально, по-видимому, изучена задача отыскания дизъюнктивных форм, минимальных по числу вхождений переменных. Эту задачу мы бегло рассматриваем в заключительных подразделах данного раздела.

Дизъюнктивной формой называется формула вида

$$\bigvee_{i=1}^{k} K_i, \quad \text{где } K_i = \bigwedge_{p=1}^{m_i} x_{j_p}^{\sigma_p}.$$

Формула K_i называется элементарной конъюнкцией, переменные в элементарную конъюнкцию входят в установленном порядке (хотя и не обязательно все n). Количество переменных в конъюнкции называется её *рангом* (обозначение $|K_i|$).

ТЕОРЕМА *Число различных дизъюнктивных форм n переменных равно 2^{3^n}.*

Доказательство Переменная либо не входит в элементарную конъюнкцию, либо входит с отрицанием, либо входит без отрицания. Таким образом, существует 3^n элементарных конъюнкций, а каждое подмножество множества элементарных конъюнкций даёт дизъюнктивную форму. □

Из этой теоремы можно извлечь два практических вывода, важных для рассматриваемой задачи минимизации дизъюнктивной формы:

1. Существует тривиальный алгоритм решения задачи: перебрать все формы, а их конечное число, и выбрать минимальную.
2. Количество дизъюнктивных форм с ростом n растёт очень быстро, и уже при сравнительно небольших n полный перебор практически неосуществим.

3.4.5. Геометрическая интерпретация

При обсуждении задач, связанных с булевыми функциями, очень полезна следующая геометрическая интерпретация. Двоичным наборам из n элементов можно взаимно-однозначно сопоставить вершины n-мерного единичного гиперкуба E^n.

Пример На рис. 3.1 представлен гиперкуб для $n = 3$. При этом булева функция $f(x_1, \ldots, x_n)$ задается подмножеством N вершин гиперкуба, на которых её значение равно 1: $N \stackrel{\text{Def}}{=} \{(a_1, \ldots, a_n) \mid f(a_1, \ldots, a_n) = 1\}$.

Пример На рис. 3.1 выделены вершины, соответствующие функции $g(x, y, z)$, имеющей следующую таблицу истинности:

x	y	z	$g(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Для функции g имеем $N = \{(0,0,0),(0,0,1),(0,1,0),(1,1,0)\}$.

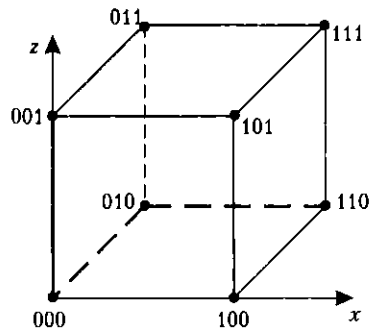

Рис. 3.1. Представление булевой функции в виде гиперкуба

Каждой элементарной конъюнкции $K = x_{i_1}^{\sigma_1} \wedge \ldots \wedge x_{i_k}^{\sigma_k}$ соответствует множество вершин гиперкуба, у которых $x_{i_1} = \sigma_1, \ldots, x_{i_k} = \sigma_k$, а значения остальных координат произвольны. Другими словами, элементарной конъюнкции K сопоставляется множество $\{(a_1,\ldots,a_n) \in E^n \mid K(a_1,\ldots,a_n) = 1\}$ вершин (кортежей), на которых конъюнкция имеет значение 1. Мы будем обозначать одной и той же буквой и элементарную конъюнкцию, и то множество вершин, на котором она принимает значение 1.

Легко видеть, что элементарная конъюнкция k переменных образует $(n-k)$-мерную грань гиперкуба E^n.

Примеры

1. Элементарной конъюнкции $\neg x \wedge \neg y$ соответствует ребро $((0,0,0),(0,0,1))$: на рис. 3.1 это ребро выделено.
2. Элементарной конъюнкции z соответствует верхняя грань куба на рис. 3.1.

Теперь ясен геометрический смысл *задачи минимизации дизъюнктивной формы.* Дано подмножество N вершин гиперкуба E^n. Требуется найти такой набор гиперграней R_i, чтобы они в совокупности образовывали покрытие N ($N = \bigcup_{i=1}^{k} K_i$) и сумма рангов $\sum_{i=1}^{k} |K_i|$ была минимальна.

3.4.6. Сокращённые дизъюнктивные формы

Известно несколько различных методов решения задачи минимизации дизъюнктивной формы. Все они используют одну и ту же идею: уменьшить по возможности множество рассматриваемых элементарных конъюнкций и затем найти минимальную дизъюнктивную форму, перебирая оставшиеся конъюнкции. Рассмотрим один из самых простых методов этого типа.

Пусть функция f задана множеством N вершин единичного гиперкуба E^n.

Если $K \subset N$, то конъюнкция K называется *допустимой* для функции f. Ясно, что в минимальную (да и любую другую) дизъюнктивную форму функции f могут входить только допустимые конъюнкции. Если $K \cap (E^n \setminus N) \neq \varnothing$, то K — недопустимая конъюнкция. Поэтому для каждого набора $(a_1, \ldots, a_n)$ из множества $E^n \setminus N$ следует удалить из множества всех 3^n конъюнкций те 2^n конъюнкций, которые можно построить из сомножителей $\{x_1^{a_1}, \ldots, x_n^{a_n}\}$.

Пример Для функции g имеем:

$f(x,y,z)=0$	Недопустимые конъюнкции							
$(0,1,1)$	1	$\neg x$	y	z	$\neg x \wedge y$	$\neg x \wedge z$	$y \wedge z$	$\neg x \wedge y \wedge z$
$(1,0,0)$	1	x	$\neg y$	$\neg z$	$x \wedge \neg y$	$x \wedge \neg z$	$\neg y \wedge \neg z$	$x \wedge \neg y \wedge \neg z$
$(1,0,1)$	1	x	$\neg y$	z	$x \wedge \neg y$	$x \wedge z$	$\neg y \wedge z$	$x \wedge \neg y \wedge z$
$(1,1,1)$	1	x	y	z	$x \wedge y$	$x \wedge z$	$y \wedge z$	$x \wedge y \wedge z$

Удаляя из множества всех 27 конъюнкций те, которые не являются допустимыми, получаем следующий список допустимых конъюнкций:

$$y \wedge \neg z, \quad \neg x \wedge \neg y, \quad \neg x \wedge \neg z, \quad x \wedge y \wedge \neg z, \quad \neg x \wedge y \wedge \neg z, \quad \neg x \wedge \neg y \wedge z, \quad \neg x \wedge \neg y \wedge \neg z.$$

Конъюнкция K называется *максимальной* для функции f, если

$$K \subset N \;\&\; (K \subset K' \subset N \Longrightarrow K' = K).$$

Очевидно, что всякая допустимая конъюнкция содержится в некоторой максимальной. Поэтому совокупность всех максимальных конъюнкций образует покрытие множества N, то есть дизъюнктивную форму, которая называется *сокращённой дизъюнктивной нормальной формой*. Сокращённая дизъюнктивная нормальная форма определена для функции f однозначно (с учётом установленного порядка переменных).

Пример Для функции g сокращённая дизъюнктивная нормальная форма имеет вид

$$(\neg x \wedge \neg y) \vee (\neg x \wedge \neg z) \vee (y \wedge \neg z),$$

что существенно короче, чем СДНФ этой функции:

$$(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge y \wedge \neg z).$$

На рис. 3.1 выделены рёбра (000–001, 000–010, 010–110), соответствующие сокращённой дизъюнктивной нормальной форме.

ТЕОРЕМА *Минимальная дизъюнктивная форма является подформулой сокращённой дизъюнктивной нормальной формы.*

Доказательство От противного. Пусть минимальная форма содержит не максимальную конъюнкцию. Тогда эту конъюнкцию можно заменить соответствующей максимальной, при этом покрытие сохранится, а сумма рангов уменьшится, что противоречит минимальности формы. □

Пример Для функции g минимальная дизъюнктивная форма имеет вид

$$\neg x \wedge \neg y \vee y \wedge \neg z.$$

Действительно, на рис. 3.1 видно, что рёбра 000–001 и 010–110 являются максимальными конъюнкциями и в совокупности покрывают множество N.

3.5. Полнота

В типичной современной цифровой вычислительной машине цифрами являются 0 и 1. Следовательно, команды, которые выполняет процессор, суть булевы функции. Выше показано, что любая булева функция реализуется через конъюнкцию, дизъюнкцию и отрицание. Следовательно, можно построить нужный процессор, имея в распоряжении элементы, реализующие конъюнкцию, дизъюнкцию и отрицание. Этот раздел посвящён ответу на вопрос: существуют ли (и если существуют, то какие) иные системы булевых функций, обладающих тем свойством, что с их помощью можно выразить все другие функции.

3.5.1. Замкнутые классы

Пусть $F = \{f_1, \ldots, f_m\}$, $\forall i \in 1..m \ \ (f_i \in P_n)$. *Замыканием* F (обозначается $[F]$) называется множество всех булевых функций, реализуемых формулами над F:

$$[F] \stackrel{\text{Def}}{=} \{f \in P_n \mid f = \operatorname{func} \mathcal{F}[F]\}.$$

Свойства замыкания (см. 2.1.2):

1. $F \subset [F]$.
2. $[[F]] = [F]$.
3. $F_1 \subset F_2 \Longrightarrow [F_1] \subset [F_2]$.
4. $([F_1] \cup [F_2]) \subset [F_1 \cup F_2]$.

Класс (множество) функций F называется *замкнутым*, если $[F] = F$.

Рассмотрим следующие классы функций.

1. Класс функций, сохраняющих 0: $T_0 \stackrel{\text{Def}}{=} \{f \mid f(0, \ldots, 0) = 0\}$.
2. Класс функций, сохраняющих 1: $T_1 \stackrel{\text{Def}}{=} \{f \mid f(1, \ldots, 1) = 1\}$.
3. Класс самодвойственных функций: $T_* \stackrel{\text{Def}}{=} \{f \mid f = f^*\}$.
4. Класс монотонных функций: $T_\leqslant \stackrel{\text{Def}}{=} \{f \mid \alpha \leqslant \beta \Longrightarrow f(\alpha) \leqslant f(\beta)\}$, где

$$\alpha = (a_1, \ldots, a_n), \quad \beta = (b_1, \ldots, b_n), a_i, b_i \in E_2, \quad \alpha \leqslant \beta \stackrel{\text{Def}}{=} \forall i \ \ (a_i \leqslant b_i).$$

5. Класс линейных функций: $T_L \stackrel{\text{Def}}{=} \{f \mid f = c_0 + c_1 x_1 + \cdots + c_n x_n\}$, где $+$ обозначает сложение по модулю 2, а знак конъюнкции опущен.

Примеры

1. Рассмотрим отрицание и введём обозначение $\varphi(x) := \overline{x}$. Имеем: $\varphi \notin T_0$, так как $\varphi(0) = 1$, $\varphi \notin T_1$, так как $\varphi(1) = 0$, $\varphi \notin T_{\leqslant}$, так как $0 < 1$, но $\varphi(0) > \varphi(1)$. С другой стороны, $\varphi \in T_*$, так как $\varphi^*(x) = \overline{\varphi}(\overline{x}) = \neg\varphi(\neg x) = \neg\neg\overline{x} = \overline{x} = \varphi(x)$, и $\varphi \in T_L$, так как $\varphi(x) = x + 1$.
2. Рассмотрим конъюнкцию и введём обозначение $\psi(x, y) := x \wedge y$. Имеем: $\psi \in T_0$, так как $0 \wedge 0 = 0$, $\psi \in T_1$, так как $1 \wedge 1 = 1$, $\psi \in T_{\leqslant}$, так как $\psi(1,1) = 1$, и $\forall\,(a,b) \neq (1,1)\ \ ((a,b) \leqslant (1,1)\ \&\ \psi(a,b) = 0)$. С другой стороны, $\psi \notin T_*$, так как $\psi^*(x,y) = x \vee y$, и $\psi \notin T_L$. Действительно, от противного, пусть $\psi(x,y) = ax + by + c$. Тогда имеем: если $x = 0$ и $y = 0$, то $a0 + b0 + c = 0$, и значит, $c = 0$; если $x = 0$ и $y = 1$, то $a0 + b1 + 0 = 0$, и значит, $b = 0$; если $x = 1$ и $y = 0$, то $a1 + 0 \cdot 0 + 0 = 0$, и значит, $a = 0$; если $x = 1$ и $y = 1$, то $0 \cdot 1 + 0 \cdot 1 + 0 = 1$, и значит, $0 = 1$ — противоречие.

ТЕОРЕМА *Классы T_0, T_1, T_*, $T_{\leqslant}$, T_L замкнуты.*

Доказательство Чтобы доказать, что некоторый класс F замкнут, достаточно показать, что если функция реализована в виде формулы над F, то она принадлежит F. Доказать, что произвольная формула обладает заданным свойством, можно с помощью индукции по структуре формулы (см. 3.3.3). База индукции очевидна: функции из F реализованы как тривиальные формулы над F. Таким образом, осталось обосновать индукционные переходы для пяти рассматриваемых классов.

[T_0] Пусть $f, f_1, \ldots, f_n \in T_0$ и $\Phi = f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$. Тогда

$$\Phi(0, \ldots, 0) = f(f_1(0, \ldots, 0), \ldots, f_n(0, \ldots, 0)) = f(0, \ldots, 0) = 0.$$

Следовательно, $\Phi \in T_0$.

[T_1] Пусть $f, f_1, \ldots, f_n \in T_1$ и $\Phi = f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$. Тогда

$$\Phi(1, \ldots, 1) = f(f_1(1, \ldots, 1), \ldots, f_n(1, \ldots, 1) = f(1, \ldots, 1) = 1.$$

Следовательно, $\Phi \in T_1$.

[T_*] Пусть $f, f_1, \ldots, f_n \in T_*$ и $\Phi = f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$. Тогда

$$\begin{aligned}\Phi^* =& f^*(f_1^*(x_1, \ldots, x_n), \ldots, f_n^*(x_1, \ldots, x_n)) = \\ =& f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) = \Phi.\end{aligned}$$

Следовательно, $\Phi \in T_*$.

[$T_{\leqslant}$] Пусть $f, f_1, \ldots, f_n \in T_{\leqslant}$ и $\Phi = f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$. Тогда

$$\begin{aligned}\alpha \leqslant \beta \Longrightarrow& (f_1(\alpha), \ldots, f_n(\alpha)) \leqslant (f_1(\beta), \ldots, f_n(\beta)) \Longrightarrow \\ \Longrightarrow& f(f_1(\alpha), \ldots, f_n(\alpha)) \leqslant f(f_1(\beta), \ldots, f_n(\beta)) \Longrightarrow \Phi(\alpha) \leqslant \Phi(\beta).\end{aligned}$$

Следовательно, $\Phi \in T_{\leqslant}$.

[T_L] Пусть $f, f_1, \ldots, f_n \in T_L$ и $\Phi = f(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$. Тогда

$$f = c_0 + c_1 x_1 + \ldots + c_n x_n,$$
$$f_1 = c_0^1 + c_1^1 x_1 + \ldots + c_n^1 x_n,$$
$$\vdots$$
$$f_n = c_0^n + c_1^n x_1 + \ldots + c_n^n x_n.$$

Подставим эти формулы в формулу для Φ. Имеем:

$$\Phi(x_1, \ldots, x_n) = c_0 + c_1 \left(c_0^1 + c_1^1 x_1 + \ldots + c_n^1 x_n\right) + \ldots + c_n \left(c_0^n + c_1^n x_1 + \ldots + c_n^n x^n\right) =$$
$$= d_0 + d_1 x_1 + \ldots + d_n x_n.$$

Следовательно, $\Phi \in T_L$. □

Пример Таблица принадлежности некоторых булевых функций рассмотренным замкнутым классам:

	T_0	T_1	T_*	$T_\leqslant$	T_L
0	+	−	−	+	+
1	−	+	−	+	+
$\overline{x}$	−	−	+	−	+
$x_1 \wedge x_2$	+	+	−	+	−

Таким образом, рассмотренные классы T_0, T_1, T_*, $T_\leqslant$, T_L попарно различны, не пусты и не совпадают с P_n.

3.5.2. Полные системы функций

Класс функций F называется *полным*, если его замыкание совпадает с P_n:

$$[F] = P_n.$$

Другими словами, множество функций F образует полную систему, если любая функция реализуема в виде формулы над F.

ТЕОРЕМА *Пусть заданы две системы функций:*

$$F = \{f_1, \ldots, f_m\} \quad \text{и} \quad G = \{g_1, \ldots, g_k\}.$$

Тогда, если система F полна и все функции из F реализуемы формулами над G, то система G также полна:

$$([F] = P_n \;\&\; \forall i \in 1..m \;\; (f_i = \text{func}\, \mathcal{G}_i[G])) \Longrightarrow [G] = P_n.$$

Доказательство Пусть h — произвольная функция, $h \in P_n$. Тогда $[F] = P_n \Longrightarrow$ $\Longrightarrow h = \text{func}\, \mathcal{F}[F] \Longrightarrow \mathcal{F}\{\mathcal{G}_i // f_i\}$ — формула над G. Следовательно, $h = \text{func}\, \mathcal{G}[G]$. □

Пример Система $\{\vee, \wedge, \neg\}$ — полная, как показано в подразделе 3.4.2. Следовательно:

1) система $\{\neg, \wedge\}$ полная, так как $x_1 \vee x_2 = \neg(\neg x_1 \wedge \neg x_2)$;
2) система $\{\neg, \vee\}$ полная, так как $x_1 \wedge x_2 = \neg(\neg x_1 \vee \neg x_2)$;
3) система $\{|\}$ полная, так как $\neg x = x \mid x$, $x_1 \wedge x_2 = \neg(x_1 \mid x_2) = (x_1 \mid x_2) \mid (x_1 \mid x_2)$;
4) система $\{0, 1, \wedge, +\}$ полная, так как $\neg x = x + 1$ (здесь $+$ означает сложение по модулю 2). Представление булевой функции над базисом $\{0, 1, \wedge, +\}$ называется *полиномом Жегалкина*[1]. Таким образом, всякая булева функция представима в виде

$$\sum_{(i_1,\ldots,i_s)} a_{i_1,\ldots,i_s} x_{i_1} \cdot \ldots \cdot x_{i_s},$$

где $\sum$ — сложение по модулю 2, знак $\cdot$ обозначает конъюнкцию и $a_{i_1,\ldots,i_s} \in E_2$.

ЗАМЕЧАНИЕ

Фактически, в полиноме Жегалкина $\sum a_{i_1,\ldots,i_s} x_{i_1} \cdot \ldots \cdot x_{i_s} = \sum x_{i_1} \cdots x_{i_s}$, поскольку если $a_{i_1,\ldots,i_s} = 1$, то этот коэффициент можно опустить, а если $a_{i_1,\ldots,i_s} = 0$, то можно опустить все слагаемое. Знак конъюнкции обычно также опускают.

3.5.3. Полнота двойственной системы

ТЕОРЕМА *Если система $F = \{f_1, \ldots, f_k\}$ полна, то система двойственных функций $F^* = \{f_1^*, \ldots, f_k^*\}$ также полна.*

Доказательство Пусть h — произвольная функция, $h \in P_n$. Рассмотрим двойственную функцию h^*. Система F полна, так что $h^* = \operatorname{func}\mathcal{H}[F]$. По принципу двойственности $h = \operatorname{func}\mathcal{H}^*[F^*]$. □

Пример Система $\{0, 1, \wedge, +\}$ полна, следовательно, система $\{1, 0, \vee, \equiv\}$ также полна.

3.5.4. Теорема Поста

Теорема Поста устанавливает необходимые и достаточные условия полноты системы булевых функций.

ТЕОРЕМА (Пост) *Система булевых функций F полна тогда и только тогда, когда она содержит хотя бы одну функцию, не сохраняющую нуль, хотя бы одну функцию, не сохраняющую единицу, хотя бы одну несамодвойственную функцию, хотя бы одну немонотонную функцию и хотя бы одну нелинейную функцию:*

$$[F] = P_n \Longleftrightarrow \neg(F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L).$$

[1] Иван Иванович Жегалкин (1869–1947).

Доказательство

[Необходимость] От противного. Пусть $[F] = P_n$ и

$$F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L.$$

Введем обозначение: i — один из индексов, 0, 1, $*$, $\leqslant$ или L.
Тогда $T_i = [T_i] \Longrightarrow [F] \subset T_i \Longrightarrow P_n \subset T_i \Longrightarrow P_n = T_i$, но $P_n \neq T_i$ по таблице из подраздела 3.5.1.

[Достаточность] Пусть $\neg(F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L)$. Тогда

$$\exists\, F' = \langle f_0, f_1, f_*, f_\leqslant, f_L\rangle \ \ (f_0 \notin T_0 \ \&\ f_1 \notin T_1 \ \&\ f_* \notin T_* \ \&\ f_\leqslant \notin T_\leqslant \ \&\ f_L \notin T_L)\,.$$

Функции $f_0, f_1, f_*, f_\leqslant, f_L$ не обязательно различны и не обязательно исчерпывают F. Покажем, что отрицание и конъюнкция реализуются в виде формул над F'. Тем самым теорема будет доказана (см. 3.5.2). Построение проводится в три этапа: на первом строятся формулы, реализующие константы 0 и 1, которые нужны на третьем этапе. На втором этапе строится формула, реализующая отрицание. На третьем этапе строится формула, реализующая конъюнкцию.

[Константы] Построим формулу, реализующую 1. Пусть $\varphi(x) := f_0(x, \ldots, x)$. Тогда

$$\varphi(0) = f_0(0, \ldots, 0) \neq 0 \Longrightarrow \varphi(0) = 1.$$

Возможны два случая: $\varphi(1) = 1$ или $\varphi(1) = 0$. Пусть $\varphi(1) = 1$. В этом случае формула φ реализует 1. Пусть $\varphi(1) = 0$. В этом случае формула φ реализует отрицание. Тогда рассмотрим функцию f_*. Имеем:

$$f_* \notin T_* \Longrightarrow \exists\, a_1, \ldots, a_n \ \ \big(f_*(a_1, \ldots, a_n) \neq \overline{f_*}(\overline{a}_1, \ldots, \overline{a}_n)\big)\,.$$

Следовательно, $f_*(a_1, \ldots, a_n) = f_*(\overline{a}_1, \ldots, \overline{a}_n)$. Пусть теперь $\psi(x) := f_*(x^{a_1}, \ldots, x^{a_n})$. Тогда

$$\begin{aligned}\psi(0) =& f_*(0^{a_1}, \ldots, 0^{a_n}) = f_*(\overline{a}_1, \ldots, \overline{a}_n) = \\ =& f_*(a_1, \ldots, a_n) = f_*(1^{a_1}, \ldots, 1^{a_n}) = \psi(1).\end{aligned}$$

Таким образом, $\psi(0) = \psi(1)$, откуда $\psi = 1$ или $\psi = 0$. Если $\psi = 1$, то требуемая константа 1 построена. В противном случае ψ реализует 0, и значит, $\varphi(\psi(x)) = \overline{\psi}(x)$ реализует 1.

Построение 0 аналогично, только вместо f_0 нужно использовать f_1.

[Отрицание] Построим формулу, реализующую отрицание.

Рассмотрим функцию $f_\leqslant$. Имеем:

$$f_\leqslant \notin T_\leqslant \Longrightarrow \exists\, \alpha = (a_1, \ldots, a_n), \beta = (b_1, \ldots, b_n) \ \ (\alpha \leqslant \beta \ \&\ f_\leqslant(\alpha) > f_\leqslant(\beta))\,.$$

Тогда $\alpha \leqslant \beta \Longrightarrow \forall\, i \ \ (a_i = b_i \vee a_i = 0 \ \&\ b_i = 1)$. Но

$$f_\leqslant(\alpha) \neq f_\leqslant(\beta) \Longrightarrow \alpha \neq \beta \Longrightarrow \exists\, J \subset 1..n \ \ (j \in J \Longrightarrow a_j = 0 \ \&\ b_j = 1)\,.$$

Другими словами, J — это множество индексов j, для которых $a_j \neq b_j$. Пусть $\varphi(x) := f_\leqslant(c_1, \ldots, c_n)$, где $c_j := x$, если $j \in J$, и $c_j := a_j (= b_j)$, если $j \notin J$. Тогда $\varphi(0) = f_\leqslant(c_1, \ldots, c_n)\{0//x\} = f_\leqslant(\alpha) > f_\leqslant(\beta) = f_\leqslant(c_1, \ldots, c_n)\{1//x\} = \varphi(1)$. Имеем: $\varphi(0) > \varphi(1) \Longrightarrow \varphi(0) = 1 \ \&\ \varphi(1) = 0 \Longrightarrow \varphi(x) = \overline{x}$.

[Конъюнкция] Построим формулу, реализующую конъюнкцию. Рассмотрим функцию f_L. Имеем:

$$f_L \in P_n \Longrightarrow f_L = \sum_{i_1,\ldots,i_s} a_{a_{i_1},\ldots,a_{i_s}} x_{i_1},\ldots,x_{i_s}.$$

Но $f_L \notin T_L$, следовательно, в полиноме Жегалкина существует нелинейное слагаемое, содержащее конъюнкцию по крайней мере двух переменных. Пусть, для определённости, это x_1 и x_2. Тогда

$$f_L = x_1 \cdot x_2 \cdot f_a(x_3,\ldots,x_n) + x_1 \cdot f_b(x_3,\ldots,x_n) + \\ + x_2 \cdot f_c(x_3,\ldots,x_n) + f_d(x_3,\ldots,x_n),$$

причём $f_a(x_3,\ldots,x_n) \neq 0$. Следовательно, $\exists a_3,\ldots,a_n$ $(f_a(a_3,\ldots,a_n) = 1)$. Пусть $b := f_b(a_3,\ldots,a_n)$, $c := f_c(a_3,\ldots,a_n)$, $d := f_d(a_3,\ldots,a_n)$ и

$$\varphi(x_1,x_2) := f_L(x_1,x_2,a_3,\ldots,a_n) = x_1 \cdot x_2 + b \cdot x_1 + c \cdot x_2 + d.$$

Пусть далее $\psi(x_1,x_2) := \varphi(x_1 + c, x_2 + b) + b \cdot c + d$. Тогда

$$\psi(x_1,x_2) = (x_1 + c) \cdot (x_2 + b) + b \cdot (x_1 + c) + c \cdot (x_2 + b) + d + b \cdot c + d = \\ = x_1 \cdot x_2 + c \cdot x_2 + b \cdot x_1 + b \cdot c + b \cdot x_1 + b \cdot c + c \cdot x_2 + b \cdot c + \\ + d + b \cdot c + d = x_1 \cdot x_2.$$

(Функции $x + a$ выразимы, так как $x + 1 = \overline{x}$, $x + 0 = x$, а константы 0, 1 и отрицание уже построены.) □

Пример В системе $\{\neg, \wedge\}$ отрицание не сохраняет констант и немонотонно, а конъюнкция несамодвойственна и нелинейна.

3.6. Представление булевых функций в программах

Для представления булевых функций можно использовать стандартные методы представления функций (см. 1.6.5), а также некоторые специальные приёмы, и эти идеи часто оказываются применимыми для представления других, более сложных объектов.

3.6.1. Табличные представления

Область определения и область значений у булевой функции конечна, поэтому булева функция может быть представлена массивом. Самое бесхитростное представление прямо воспроизводит таблицу истинности. Если условиться, что кортежи в таблице всегда идут в установленном порядке (см. 3.1.1), то для представления булевой функции $f(x_1,\ldots,x_n)$ достаточно хранить столбец значений:

F_1 : **array** $[0..(2^n - 1)]$ **of** $0..1$

Пример В этом и последующих примерах рассматривается булева функция $g(x, y, z)$, заданная следующей таблицей истинности:

x	y	z	$g(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$F_1(g)$: **array** [0..7] **of** 0..1 : =(1, 1, 1, 0, 0, 0, 1, 0)

Данное представление — не самое эффективное. В частности, если набор аргументов задан массивом x : **array** [1..n] **of** 0..1, то для вычисления значения функции f с помощью представления F_1 необходимо сначала вычислить индекс d (то есть номер кортежа в установленном порядке), чтобы затем получить значение $F_1[d]$. Индекс d нетрудно вычислить, например, с помощью следующего алгоритма.

Алгоритм 3.2 Вычисление номера кортежа в установленном порядке

Вход: кортеж x : **array** [1..n] **of** 0..1 значений переменных.
Выход: номер d кортежа x при перечислении кортежей в установленном порядке.

```
d := 0 { начальное значение индекса }
for i from 1 to n do
  d := d * 2 { сдвигаем код числа d влево на один разряд }
  if x[i] = 1 then
    d := d + 1 { добавляем 1, если нужно }
  end if
end for
```

Обоснование Если рассматривать кортеж булевых значений как двоичную запись числа, то это число — номер кортежа в установленном порядке. Значение числа d, заданного в позиционной двоичной системе счисления цифрами $x_1 \ldots x_n$, определяется следующим образом:

$$d = x_1 2^{n-1} + \ldots + x_n 2^0 = \sum_{i=1}^{n} x_i 2^{n-i} = 2(2(\ldots(2x_1 + x_2)\ldots) + x_{n-1}) + x_n.$$

Цикл в алгоритме непосредственно вычисляет последнюю формулу, которая является частным случаем *схемы Горнера*. □

ЗАМЕЧАНИЕ

По схеме Горнера можно определить значение числа d по записи $x_1 \dots x_n$ в *любой* позиционной системе счисления с основанием b: $d = b(b(\dots(bx_1 + x_2)\dots) + x_{n-1}) + x_n$.

Более эффективным представлением таблицы истинности является использование n-мерного массива:

F_2 : **array** $[0..1, \dots, 0..1]$ **of** $0..1$

В случае использования представления F_2 значение функции $f(x_1, \dots, x_n)$ задаётся выражением $F_2[x_1, \dots, x_n]$.

Пример Для функции g из предыдущего примера

$$F_2 : \textbf{array}\ [0..1, 0..1, 0..1]\ \textbf{of}\ 0..1 = (((1,1),(1,0)),((0,0),(1,0)))$$

Представления булевой функции n переменных в виде массива занимают память объёмом $O(2^n)$.

3.6.2. Строковые представления

Булеву функцию можно представить с помощью реализующей её формулы. Существует множество способов представления формул, но наиболее удобной для человека является запись в виде цепочки символов на некотором формальном языке.

Как уже указывалось, для любой булевой функции существует бесконечно много различных реализующих её формул. Однако булевы функции имеют нормальные формы, в частности СДНФ (см. 3.4.2), и при установленном порядке переменных СДНФ единственна.

СДНФ булевой функции может быть построена по заданной таблице истинности с помощью следующего алгоритма.

Алгоритм 3.3 Построение СДНФ

Вход: вектор x : **array** $[1..n]$ **of string** идентификаторов переменных,
вектор F_1 : **array** $[0..2^n - 1]$ **of** $0..1$ значений функции при установленном порядке кортежей.
Выход: последовательность символов, образующих запись формулы СДНФ для заданной функции.

```
f := false { признак присутствия левого операнда дизъюнкции }
for i from 0 to 2^n − 1 do
  if F1[i] = 1 then
    if f then
      yield ' ∨ ' { добавление в формулу знака дизъюнкции }
    else
      f := true { это первое слагаемое в дизъюнкции }
    end if
```

```
    g := false { признак присутствия левого операнда конъюнкции }
    for j from 1 to n do
      if g then
        yield '∧' { добавление в формулу знака конъюнкции }
      else
        g := true { это первый сомножитель в конъюнкции }
      end if
      v :=(i div 2^(j-1)) mod 2  { значение j-го разряда кортежа с номером i }
      if v = 0 then
        yield '¬' { добавление в формулу знака отрицания }
      end if
      yield x[j] { добавление в формулу идентификатора переменной }
    end for
  end if
end for
```

Обоснование Данный алгоритм буквально воспроизводит словесную запись следующего правила: для каждой строки таблицы истинности, для которой значение функции равно 1, построить дизъюнктивное слагаемое, включающее все переменные, причём те переменные, которые имеют значение 0 в этой строке, входят со знаком отрицания. Остальное в этом алгоритме — мелкие программистские «хитрости», которые полезно один раз посмотреть, но не стоит обсуждать. □

Пример Для функции g, используемой в примерах данного раздела, алгоритм построит строку

$$\neg x \wedge \neg y \wedge \neg z \vee \neg x \wedge \neg y \wedge z \vee \neg x \wedge y \wedge \neg z \vee x \wedge y \wedge \neg z.$$

Представление функции в виде формулы, выраженной как строка символов, совершенно необходимо при реализации интерфейса пользователя в системах компьютерной алгебры, но крайне неудобно при выполнении других манипуляций с формулами. В следующем подразделе рассматривается представление СДНФ, более удобное, например, для вычисления значения функции.

3.6.3. Алгоритм вычисления значения булевой функции

Некоторые классы формул допускают более эффективную интерпретацию по сравнению с алгоритмом Eval. Рассмотрим алгоритм вычисления значения булевой функции, заданной в виде СДНФ, для заданных значений переменных $x_1, \ldots, x_n$. В этом алгоритме используется следующее представление данных. СДНФ задана массивом F_3: **array** $[1..k, 1..n]$ **of** 0..1, где строка $F_3[i, *]$ содержит набор значений $\sigma_1, \ldots, \sigma_n$, для которого $f(\sigma_1, \ldots, \sigma_n) = 1$, $i \in 1..k$, $k \leqslant 2^n$.

Пример Для функции g

F_3 : **array** $[1..4, 1..3]$ **of** $0..1 = ((0,0,0),(0,0,1),(0,1,0),(1,1,0))$.

ОТСТУПЛЕНИЕ

Быстрое вычисление значения СДНФ имеет, помимо теоретического, большое практическое значение. Например, во многих современных программах с графическим интерфейсом для составления сложных логических условий используется наглядный бланк в виде таблицы: в клетках записываются условия, причём клетки одного столбца считаются соединёнными конъюнкцией, а столбцы — дизъюнкцией, то есть образуют ДНФ (или наоборот, в таком случае получается КНФ). В частности, так устроен графический интерфейс QBE (Query-by-Example), применяемый для формулировки логических условий при запросе к СУБД.

Алгоритм 3.4 Алгоритм вычисления значения СДНФ

Вход: массив, представляющий СДНФ: f : **array** $[1..k, 1..n]$ **of** 0..1;
множество значений переменных x : **array** $[1..n]$ **of** 0..1.
Выход: 0..1 — значение булевой функции.

```
for i from 1 to k do
  for j from 1 to n do
    if f[i,j] ≠ x[j] then
      next for i { x_j ≠ σ_j ⟹ x_j^{σ_j} = 0 ⟹ x_1^{σ_1} ∧ ... ∧ x_n^{σ_n} = 0 }
    end if
  end for
  return 1 { x_1^{σ_1} & ... & x_n^{σ_n} = 1 ⟹ ⋁_{(σ_1,...,σ_m)} x_1^{σ_1} ∧ ... ∧ x_n^{σ_n} = 1 }
end for
return 0 { все слагаемые в дизъюнкции равны нулю }
```

Обоснование Алгоритм основан на следующих (очевидно, верных) правилах. Можно прекратить вычисление конъюнкции, как только получен конъюнктивный сомножитель, равный 0 (вся конъюнкция имеет значение 0). Можно прекратить вычисление дизъюнкции, как только получено дизъюнктивное слагаемое, равное 1 (вся дизъюнкция имеет значение 1). □

Этот алгоритм в худшем случае выполняет $k \cdot n$ сравнений, а в среднем — гораздо меньше. Таким образом, он существенно эффективнее общего алгоритма интерпретации.

3.6.4. Деревья решений

Наиболее эффективными с точки зрения экономии памяти и времени оказываются представления, которые не имеют прямой связи с «естественными» представлениями функции в виде графика (массива) или формулы (выражения), но специально ориентированы на выполнение операций.

Начнём с простого наблюдения. Таблицу истинности булевой функции n переменных можно представить в виде полного бинарного дерева высоты $n + 1$.

ЗАМЕЧАНИЕ

Бинарные деревья, равно как и другие виды деревьев, а также способы их представления в программах и соответствующая терминология, подробно рассматриваются в главе 9.

Ярусы дерева соответствуют переменным, дуги дерева соответствуют значениям переменных, скажем, левая дуга — 0, а правая — 1. Листья дерева на последнем ярусе хранят значение функции на кортеже, соответствующем пути из корня в этот лист. Такое дерево называется *деревом решений* (или *семантическим деревом*).

Пример На рис. 3.2 слева представлено дерево решений для функции g.

Дерево решений можно сократить, если заменить корень каждого поддерева, все листья которого имеют одно и то же значение, этим значением. Иногда такое сокращение значительно уменьшает объём дерева.

Пример На рис. 3.2 справа представлено сокращённое дерево решений для функции g.

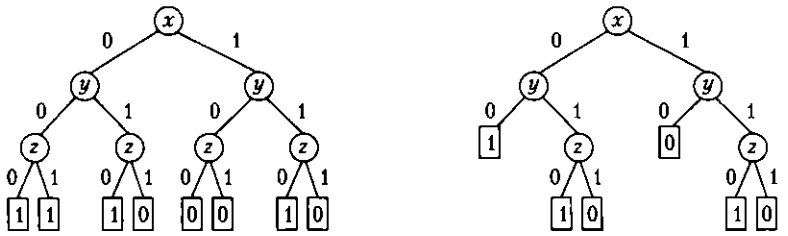

Рис. 3.2. Дерево решений и сокращённое дерево решений

Вычисление значения функции осуществляется проходом по дереву решений, как показано в алгоритме 3.5. При этом тип узла дерева N определён следующим образом: N = **record** i : 0..1; l, r : ↑ N **end record**.

Алгоритм 3.5 Вычисление значения функции по сокращённому дереву решений

Вход: указатель T : ↑ N на корень дерева решений; массив x : **array** $[1..n]$ **of** 0..1 значений переменных.
Выход: 0..1 — значение булевой функции.

```
for i from 1 to n do
  if T.l = nil & T.r = nil then
    return T.i { листовой узел — возвращаем значение }
  else
    if x[i] then
      T := T.r { 1 — переход вправо }
```

```
      else
        T := T.l { 0 — переход влево }
      end if
    end if
  end for
```

Дерево решений можно сделать ещё компактнее, если отказаться от древовидности связей, то есть допускать несколько дуг, входящих в узел. В таком случае мы получаем *бинарную диаграмму решений*. Бинарная диаграмма решений получается из бинарного дерева решений тремя последовательными преобразованиями:

1. Отождествляются листовые узлы, содержащие 0 и содержащие 1.

   Пример На рис. 3.3, *а* показано исходное полное дерево решений для функции g, а на рис. 3.3, *б* — результат первого преобразования.

2. В диаграмме выделяются изоморфные поддиаграммы и заменяются единственным их экземпляром.

   Пример На рис. 3.3, *б* видно, что два поддерева, корнями которых являются узлы z, находящиеся в середине, изоморфны. На рис. 3.3, *в* показан результат второго преобразования.

3. Исключаются узлы, обе исходящие дуги которых ведут в один узел.

   Пример На рис. 3.3, *в* видно, что крайние узлы z излишни — значение функции от значения z не зависит. Они удаляются, а входящие в них дуги продолжаются до тех узлов, в которые вели дуги из узлов z. На рис. 3.3, *г* показан результат третьего преобразования, который является построенной диаграммой решений.

Интерпретация бинарной диаграммы решений (вычисление значения функции) производится в точности так же, как и для дерева решений, то есть по алгоритму 3.5.

Результат преобразования дерева решений в диаграмму решений существенно зависит от того, в каком порядке рассматриваются переменные при построении исходного полного дерева решений.

Пример На рис. 3.4 показаны последовательность преобразований и окончательная диаграмма решений для функции g в том случае, когда переменные рассматриваются в следующем порядке: y, x, z. Мы видим, что диаграмма на рис. 3.4, *г* компактнее диаграммы на рис. 3.3, *г* и вычисление значения функции g требует всего двух операций. Фактически, диаграмма на рис. 3.4 показывает, что функция g может быть реализована следующим условным выражением:
if y **then** $\neg z$ **else** $\neg x$ **end if**.

ОТСТУПЛЕНИЕ

Последний разобранный пример свидетельствует, что иногда можно построить такое специальное представление функции, которое позволяет хранить меньше информации и при этом производить вычисления быстрее, чем это в принципе возможно при использовании универсальных математических методов представления функций с помощью графиков (массивов) и формул (выражений). Советуем читателям обдумать этот факт.

Комментарии

Прекрасным руководством по булевым функциям являются книги [11] и [10], в которых можно найти обширный дополнительный материал, в частности, опущенные за недостатком места более изощрённые алгоритмы построения, упрощения и минимизации дизъюнктивных нормальных форм. Все алгоритмы этой главы являются программистским фольклором, обработанным автором. Краткое описание бинарных деревьев решений и много другого полезного материала можно найти в книге [12].

Упражнения

3.1. Доказать, что число булевых функций от n переменных, среди которых k фиктивных, равно $2^{2^{n-k}}$.

3.2. Проверить равносильности из подраздела 3.2.2 путем построения таблиц истинности.

3.3. Какие функции являются двойственными для функций $+$, $\equiv$, $|$, $\downarrow$?

3.4. Построить СДНФ для функций $x_1 \mid x_2$, $x_1 \downarrow x_2$, $x_1 \to x_2$, $x_1 + x_2$.

3.5. Доказать, что если $f \notin T_0$, то $f \notin T_* \vee (f \notin T_1 \,\&\, f \notin T_\leqslant)$.

3.6. Построить СДНФ, сокращённую дизъюнктивную нормальную форму, минимальную дизъюнктивную форму, сокращённое дерево решений и бинарную диаграмму решений для функции, заданной формулой $(x \to y) \to z$.

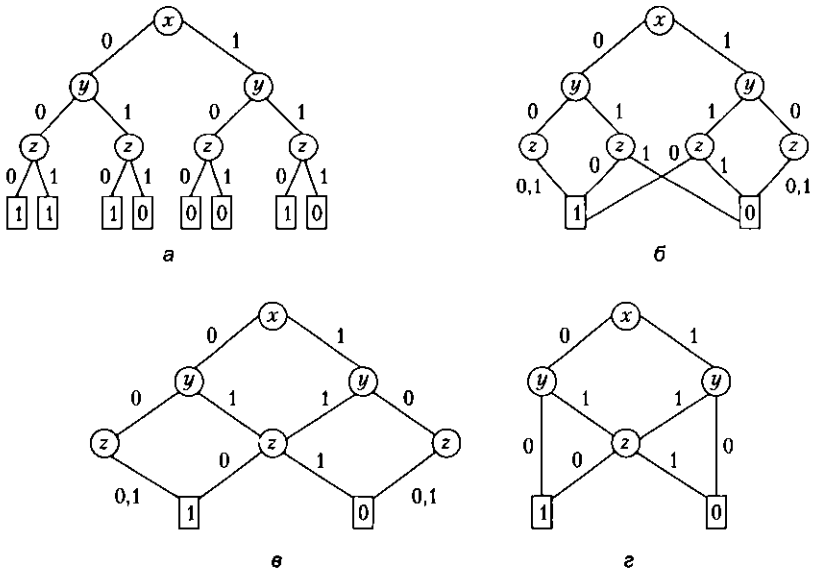

Рис. 3.3. Построение бинарной диаграммы решений

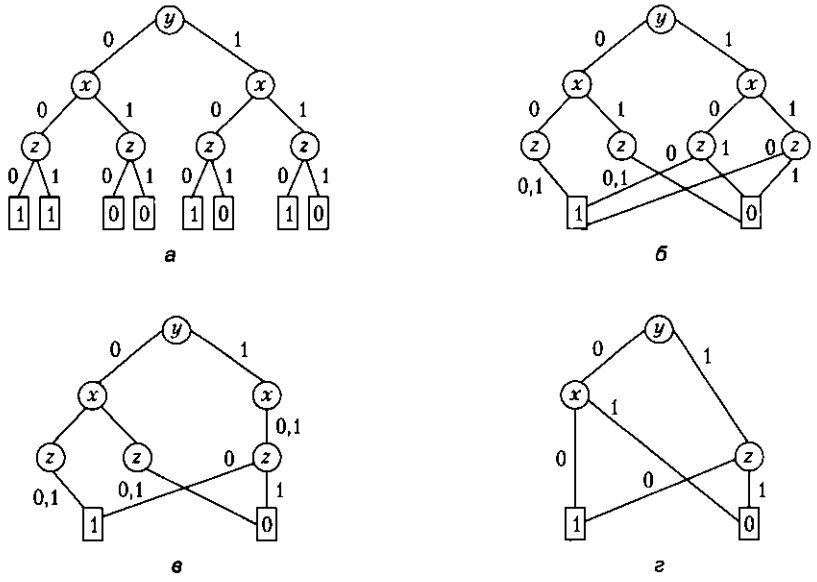

Рис. 3.4. Бинарная диаграмма решений при другом порядке переменных

Глава 4 Логические исчисления

С древнейших времён человечеству известна логика, или искусство правильно рассуждать. Вообще говоря, способность к рассуждениям — это именно искусство. Имея какие-то утверждения (посылки), истинность которых проверена, скажем, на опыте, логик путем умозрительных построений приходит к другому утверждению (заключению), которое также оказывается истинным (в некоторых случаях). Опыт древних (чисто наблюдательный) был систематизирован Аристотелем[1]. Он рассмотрел конкретные виды рассуждений, которые назвал *силлогизмами*. А именно, Аристотель рассмотрел так называемые *категорические утверждения* четырех видов (A, B — *категории*):

1) все A обладают свойством B (все A суть B);
2) некоторые A обладают свойством B (некоторые A суть B);
3) все A не обладают свойством B (все A суть не B);
4) некоторые A не обладают свойством B (некоторые A суть не B)

и зафиксировал все случаи, когда из посылок такого вида выводятся заключения одного из этих же видов.

Примеры

1. Все люди смертны. Сократ — человек. Следовательно, Сократ смертен. Это рассуждение правильно, потому что подходит под один из образцов силлогизмов Аристотеля.
2. Все дикари раскрашивают свои лица. Некоторые современные молодые люди раскрашивают свои лица. Следовательно, некоторые современные молодые люди — дикари. Это рассуждение неправильно, хотя, видимо, все входящие в него утверждения истинны.

Логика Аристотеля — это *классическая* логика, то есть наука, традиционно относящаяся к гуманитарному циклу и тем самым находящаяся вне рамок данной книги.

---

[1] Аристотель (384–322 до н. э.).

Предметом этой главы являются некоторые элементы логики *математической*, которая соотносится с логикой классической примерно так, как язык Паскаль соотносится с английским языком. Эта аналогия довольно точна и по степени формализованности, и по широте применимости в реальной жизни, и по значимости для практического программирования. План главы состоит в том, чтобы на основе небольшого предварительного рассмотрения ввести понятие «формальной теории», или «исчисления», в наиболее общем виде, а затем конкретизировать это понятие примерами двух наиболее часто используемых исчислений: исчисления высказываний и исчисления предикатов.

4.1. Логические связки

Цель данного раздела — неформально ввести специфическую «логическую» терминологию и указать на её связь с материалом предшествующих глав. В последующих разделах главы определения этого раздела уточняются и конкретизируются.

4.1.1. Высказывания

Элементами логических рассуждений являются утверждения, которые либо истинны, либо ложны, но не то и другое вместе. Такие утверждения называются *(простыми) высказываниями*. Простые высказывания обозначаются *пропозициональными переменными*. Пропозициональная переменная может принимать одно из двух *истинностных значений*, обозначаемых символами «T» и «F».

ЗАМЕЧАНИЕ

Истинностные значения обозначают различными способами: латинскими буквами T и F, как в этом учебнике, русскими буквами И и Л (от слов «Истина» и «Ложь»), цифрами 1 и 0, специальными значками $\top$ и $\bot$. Каждый из способов обозначения истинностных значений имеет свои достоинства и недостатки. Пара символов T и F выбрана нами, во-первых, для удобства чтения, во-вторых, эти символы не используются в учебнике в другом смысле, и, в-третьих, ради ассоциации с английскими словами «true» и «false», которые используются во многих языках программирования для обозначения истинностных значений.

Из простых высказываний с помощью *логических связок* могут быть построены составные высказывания. Обычно рассматривают следующие логические связки.

Название	Прочтение	Обозначение
Отрицание	не	$\neg$
Конъюнкция	и	&
Дизъюнкция	или	$\vee$
Импликация	если... то	$\rightarrow$

4.1.2. Формулы

Правильно построенные составные высказывания называются *(пропозициональными) формулами*. Формулы имеют следующий синтаксис:

⟨формула⟩ ::= T | F |
⟨пропозициональная переменная⟩ |
(¬⟨формула⟩) |
(⟨формула⟩ & ⟨формула⟩) |
(⟨формула⟩ ∨ ⟨формула⟩) |
(⟨формула⟩ → ⟨формула⟩)

ЗАМЕЧАНИЕ

Для описания синтаксиса языка мы используем один из стандартных приёмов: контекстно-свободную порождающую грамматику в форме Бэкуса—Наура. Здесь названия синтаксических конструкций заключаются в угловые скобки, метасимвол «::=» означает «это», метасимвол «|» означает «или», все остальные символы означают сами себя. Исчерпывающее изложение теории формальных грамматик применительно к программированию можно найти в [1].

Для упрощения записи вводится старшинство связок (¬, &, ∨, →) и лишние скобки часто опускаются.

Истинностное значение формулы определяется через истинностные значения её составляющих в соответствии со следующей таблицей:

A	B	$\neg A$	$A \,\&\, B$	$A \vee B$	$A \to B$
F	F	T	F	F	T
F	T	T	F	T	T
T	F	F	F	T	F
T	T	F	T	T	T

4.1.3. Интерпретация

Пусть $A(x_1, \ldots, x_n)$ — пропозициональная формула, где $x_1, \ldots, x_n$ — входящие в неё пропозициональные переменные. Конкретный набор истинностных значений, приписанных переменным $x_1, \ldots, x_n$, называется *интерпретацией* формулы A. Формула может быть истинной (иметь значение T) при одной интерпретации и ложной (иметь значение F) при другой интерпретации. Значение формулы A в интерпретации I будем обозначать $I(A)$. Формула, истинная при некоторой интерпретации, называется *выполнимой*. Формула, истинная при всех возможных интерпретациях, называется *общезначимой* (или *тавтологией*). Формула, ложная при всех возможных интерпретациях, называется *невыполнимой* (или *противоречием*).

Примеры

$A \vee \neg A$ — тавтология, $A \,\&\, \neg A$ — противоречие, $A \to \neg A$ — выполнимая формула, она истинна при $I(A) = \mathrm{F}$.

ТЕОРЕМА 1 *Пусть A — некоторая формула. Тогда:*

1) *если A — тавтология, то $\neg A$ — противоречие, и наоборот;*
2) *если A — противоречие, то $\neg A$ — тавтология, и наоборот;*
3) *если A — тавтология, то неверно, что A — противоречие, но не наоборот;*
4) *если A — противоречие, то неверно, что A — тавтология, но не наоборот;*
5) *если $\neg A$ выполнима, то неверно, что A — тавтология;*
6) *если A выполнима, то неверно, что A — противоречие.*

Доказательство Очевидно из определений. □

ТЕОРЕМА 2 *Если формулы A и $A \to B$ — тавтологии, то формула B — тавтология.*

Доказательство От противного. Пусть $I(B) = \mathrm{F}$. Но $I(A) = \mathrm{T}$, так как A — тавтология. Значит, $I(A \to B) = \mathrm{F}$, что противоречит предположению о том, что $A \to B$ — тавтология. □

4.1.4. Логическое следование и логическая эквивалентность

Говорят, что формула B *логически следует* из формулы A (обозначение $A \Longrightarrow B$), если формула B имеет значение T при всех интерпретациях, при которых формула A имеет значение T. Говорят, что формулы A и B *логически эквивалентны* (обозначается $A \Longleftrightarrow B$ или просто $A = B$), если они являются логическими следствиями друг друга. Очевидно, что логически эквивалентные формулы имеют одинаковые значения при любой интерпретации.

ТЕОРЕМА 1 $P \to Q = \neg P \vee Q$.

Доказательство Для доказательства достаточно проверить, что формулы действительно имеют одинаковые истинностные значения при всех интерпретациях.

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

□

ТЕОРЕМА 2 *Если A, B, C — любые формулы, то имеют место следующие логические эквивалентности:*

1. $A \vee A = A$, $A \,\&\, A = A$.
2. $A \vee B = B \vee A$, $A \,\&\, B = B \,\&\, A$.
3. $A \vee (B \vee C) = (A \vee B) \vee C$, $A \,\&\, (B \,\&\, C) = (A \,\&\, B) \,\&\, C$.
4. $A \vee (B \,\&\, C) = (A \vee B) \,\&\, (A \vee C)$, $A \,\&\, (B \vee C) = (A \,\&\, B) \vee (A \,\&\, C)$.
5. $(A \,\&\, B) \vee A = A$, $(A \vee B) \,\&\, A = A$.
6. $A \vee \mathrm{F} = A$, $A \,\&\, \mathrm{F} = \mathrm{F}$.
7. $A \vee \mathrm{T} = \mathrm{T}$, $A \,\&\, \mathrm{T} = A$.
8. $\neg(\neg A) = A$.
9. $\neg(A \,\&\, B) = \neg A \vee \neg B$, $\neg(A \vee B) = \neg A \,\&\, \neg B$.
10. $A \vee \neg A = \mathrm{T}$, $A \,\&\, \neg A = \mathrm{F}$.

Доказательство Непосредственно проверяется построением таблиц истинности. □

ЗАМЕЧАНИЕ

Таким образом, алгебра $(\{\mathrm{T},\mathrm{F}\}; \vee, \&, \neg)$ является булевой алгеброй, которая называется *алгеброй высказываний*.

ТЕОРЕМА 3 *$P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ тогда и только тогда, когда $(P_1 \,\&\, \ldots \,\&\, P_n) \to Q$ — тавтология.*

Доказательство

[$\Longrightarrow$] Пусть $I(P_1 \,\&\, \ldots \,\&\, P_n) = \mathrm{T}$. Тогда $I(Q) = \mathrm{T}$ и $I(P_1 \,\&\, \ldots \,\&\, P_n \to Q) = \mathrm{T}$. Пусть $I(P_1 \,\&\, \ldots \,\&\, P_n) = \mathrm{F}$. Тогда $I(P_1 \,\&\, \ldots \,\&\, P_n \to Q) = \mathrm{T}$ при любой интерпретации I. Таким образом, формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ общезначима.

[$\Longleftarrow$] Пусть $I(P_1 \,\&\, \ldots \,\&\, P_n) = \mathrm{T}$. Тогда $I(Q) = \mathrm{T}$, иначе бы формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ не была бы тавтологией. Таким образом, формула Q — логическое следствие формулы $P_1 \,\&\, \ldots \,\&\, P_n$. □

ТЕОРЕМА 4 *$P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ тогда и только тогда, когда $P_1 \,\&\, \ldots \,\&\, P_n \,\&\, \neg Q$ — противоречие.*

Доказательство По предыдущей теореме $P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ тогда и только тогда, когда формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ — тавтология. По первой теореме подраздела 4.1.3 формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ является тавтологией тогда и только тогда, когда формула $\neg(P_1 \,\&\, \ldots \,\&\, P_n \to Q)$ является противоречием. Имеем:

$$\neg(P_1 \,\&\, \ldots \,\&\, P_n \to Q) = \neg(\neg(P_1 \,\&\, \ldots \,\&\, P_n) \vee Q) =$$
$$= \neg\neg(P_1 \,\&\, \ldots \,\&\, P_n) \,\&\, \neg Q = P_1 \,\&\, \ldots \,\&\, P_n \,\&\, \neg Q. \quad \square$$

4.1.5. Подстановка и замена

Пусть A — некоторая формула, в которую входит переменная x (обозначается $A(\dots x\dots)$) или входит некоторая подформула B (обозначается $A(\dots B\dots)$), и пусть C — некоторая формула. Тогда $A(\dots x\dots)\{C//x\}$ обозначает формулу, полученную из формулы A подстановкой формулы C вместо *всех* вхождений переменной x, а $A(\dots B\dots)\{C/B\}$ обозначает любую формулу, полученную из формулы A подстановкой формулы C вместо *некоторых* (в частности, вместо одного) вхождений подформулы B.

ТЕОРЕМА 1 *Если формула $A(\dots x\dots)$ — тавтология, а B — любая формула, то $A(\dots x\dots)\{B//x\}$ — тавтология.*

Доказательство Пусть $C := A(\dots x\dots)\{B//x\}$. Пусть I — интерпретация формулы C (формула уже не содержит переменной x). Построим интерпретацию I' формулы A, положив $x := I(B)$, а значения остальных переменных взяв такими же, как в интерпретации I. Тогда $I'(A) = I(C)$, но $I'(A) = \mathrm{T}$, следовательно, $I(C) = \mathrm{T}$. □

ТЕОРЕМА 2 *Если $A(\dots B\dots)$ и $B = C$, а $D := A(\dots B\dots)\{C/B\}$, то $A = D$.*

Доказательство Пусть I — любая интерпретация, содержащая значения для всех переменных в формулах A, B и C. Тогда $I(B) = I(C)$, значит, $I(A) = I(D)$. □

Теоремы 1 и 2 аналогичны правилам подстановки и замены, изложенным в разделе 3.2.3.

4.2. Формальные теории

Исторически понятие формальной теории было разработано в период интенсивных исследований в области оснований математики для формализации собственно логики и теории доказательства. Сейчас этот аппарат широко используется при создании специальных исчислений для решения конкретных прикладных задач. Рамки книги не позволяют привести развёрнутые примеры подобных специальных исчислений (они довольно велики), но, тем не менее, такие примеры существуют.

4.2.1. Определение формальной теории

Формальная теория $\mathcal{T}$ имеет следующие составляющие:

1) множество $\mathcal{A}$ символов, образующих *алфавит*;
2) множество $\mathcal{F}$ слов в алфавите $\mathcal{A}$, $\mathcal{F} \subset \mathcal{A}^*$, которые называются *формулами*;
3) подмножество $\mathcal{B}$ формул, $\mathcal{B} \subset \mathcal{F}$, которые называются *аксиомами*;
4) множество $\mathcal{R}$ отношений R на множестве формул, $R \in \mathcal{R}$, $R \subset \mathcal{F}^{n+1}$, которые называются *правилами вывода*.

Множество символов $\mathcal{A}$ может быть конечным или бесконечным. Обычно для образования символов используют конечное множество букв, к которым, если нужно, приписываются в качестве индексов натуральные числа.

Множество формул $\mathcal{F}$ обычно задается индуктивным определением, например, с помощью порождающей формальной грамматики. Как правило, это множество бесконечно. Множества $\mathcal{A}$ и $\mathcal{F}$ в совокупности определяют *язык*, или *сигнатуру*, формальной теории.

Множество аксиом $\mathcal{B}$ может быть конечным или бесконечным. Если множество аксиом бесконечно, то обычно оно задаётся с помощью конечного множества *схем* аксиом и правил порождения конкретных аксиом из схемы аксиом. Часто аксиомы делятся на два вида: *логические* аксиомы (общие для целого класса формальных теорий) и *нелогические* (или *собственные*) аксиомы (определяющие специфику и содержание конкретной теории).

Множество правил вывода $\mathcal{R}$ чаще всего конечно.

4.2.2. Выводимость

Пусть $F_1, \ldots, F_n, G$ — формулы теории $\mathcal{T}$, то есть $F_1, \ldots, F_n, G \in \mathcal{F}$. Если существует такое правило вывода R, $R \in \mathcal{R}$, что $(F_1, \ldots, F_n, G) \in R$, то говорят, что формула G *непосредственно выводима* из формул $F_1, \ldots, F_n$ по правилу вывода R. Обычно этот факт записывают следующим образом:

$$\frac{F_1, \ldots, F_n}{G}\,(R).$$

Здесь формулы $F_1, \ldots, F_n$ называются *посылками*, формула G — *заключением*, а R является обозначением правила вывода.

ЗАМЕЧАНИЕ

Обозначение правила вывода справа от черты, разделяющей посылки и заключение, часто опускают, если оно ясно из контекста.

Выводом формулы G из формул $F_1, \ldots, F_n$ в формальной теории $\mathcal{T}$ называется такая последовательность формул $E_1, \ldots, E_k$, что $E_k = G$, а любая формула E_i $(i < k)$ является либо аксиомой $(E_i \in \mathcal{B})$, либо исходной формулой F_j $(E_i = F_j)$, либо непосредственно выводима из ранее полученных формул $E_{j_1}, \ldots, E_{j_n}$ $(j_1, \ldots, j_n < i)$. Если в теории $\mathcal{T}$ существует вывод формулы G из формул $F_1, \ldots, F_n$, то это записывают следующим образом:

$$F_1, \ldots, F_n \vdash_{\mathcal{T}} G.$$

Формулы $F_1, \ldots, F_n$ называются *гипотезами* вывода. Если теория $\mathcal{T}$ подразумевается, то её обозначение обычно опускают.

Если $\vdash_{\mathcal{T}} G$, то формула G называется *теоремой* теории $\mathcal{T}$ (то есть теорема — это формула, выводимая только из аксиом, без гипотез).

Если $\Gamma \vdash_{\mathcal{T}} G$, то очевидно, что $\Gamma, \Delta \vdash_{\mathcal{T}} G$, где Γ и Δ — любые множества формул (то есть при добавлении лишних гипотез выводимость сохраняется).

ЗАМЕЧАНИЕ

При изучении формальных теорий нужно различать теоремы *самой* формальной теории и теоремы *об этой* формальной теории, или *метатеоремы*. Это различие иногда не формализуется явно, но всегда является существенным. В этой главе теоремы конкретной формальной теории, как правило, записываются в виде формул, составленных из специальных знаков, а метатеоремы формулируются на естественном языке, чтобы их легче было отличать от теорем самой формальной теории.

4.2.3. Интерпретация

В этом подразделе вводится более общее и строгое определение понятия интерпретации по сравнению с подразделом 4.1.3.

Интерпретацией формальной теории $\mathcal{T}$ в область интерпретации M называется функция $I\colon \mathcal{F} \to M$, которая каждой формуле формальной теории $\mathcal{T}$ однозначно сопоставляет некоторое содержательное высказывание относительно объектов множества (алгебраической системы) M. Это высказывание может быть истинным или ложным (или не иметь истинностного значения). Если соответствующее высказывание является истинным, то говорят, что формула *выполняется* в данной интерпретации.

ОТСТУПЛЕНИЕ

В конечном счёте, нас интересуют такие формальные теории, которые описывают какие-то реальные объекты и связи между ними. Речь идёт прежде всего о математических объектах и математических теориях, которые выбираются в качестве области интерпретации.

Интерпретация I называется *моделью* множества формул Γ, если все формулы этого множества выполняются в интерпретации I. Интерпретация I называется *моделью* формальной теории $\mathcal{T}$, если все теоремы этой теории выполняются в интерпретации I (то есть все выводимые формулы оказываются истинными в данной интерпретации).

4.2.4. Общезначимость и непротиворечивость

Формула называется *общезначимой* (или *тавтологией*), если она истинна в любой интерпретации. Формула называется *противоречивой*, если она ложна в любой интерпретации.

Формула G называется *логическим следствием* множества формул Γ, если G выполняется в любой модели Γ.

Формальная теория $\mathcal{T}$ называется *семантически непротиворечивой*, если ни одна её теорема не является противоречием. Таким образом, формальная теория пригодна для описания тех множеств (алгебраических систем), которые являются её моделями. Модель для формальной теории $\mathcal{T}$ существует тогда и только тогда, когда $\mathcal{T}$ семантически непротиворечива.

Формальная теория $\mathcal{T}$ называется *формально непротиворечивой*, если в ней не являются выводимыми одновременно формулы F и $\neg F$. Теория $\mathcal{T}$ формально непротиворечива тогда и только тогда, когда она семантически непротиворечива.

ЗАМЕЧАНИЕ

Последние утверждения являются доказуемыми метатеоремами, доказательства которых опускаются из-за технических сложностей.

4.2.5. Полнота, независимость и разрешимость

Пусть интерпретация $I: \mathcal{T} \to M$ является моделью формальной теории $\mathcal{T}$. Тогда формальная теория $\mathcal{T}$ называется *полной* (или *адекватной*), если каждому истинному высказыванию области интерпретации M соответствует теорема теории $\mathcal{T}$.

Если для множества (алгебраической системы) M существует формальная полная непротиворечивая теория $\mathcal{T}$, то система M называется *аксиоматизируемой* (или *формализуемой*).

Система аксиом (или аксиоматизация) формально непротиворечивой теории $\mathcal{T}$ называется *независимой*, если никакая из аксиом не выводима из остальных по правилам вывода теории $\mathcal{T}$.

Формальная теория $\mathcal{T}$ называется *разрешимой*, если существует алгоритм, который для любой формулы теории определяет, является ли эта формула теоремой теории. Формальная теория $\mathcal{T}$ называется *полуразрешимой*, если существует алгоритм, который для любой формулы F теории выдает ответ «ДА», если F является теоремой теории, и, может быть, не выдает никакого ответа, если F не является теоремой (то есть алгоритм применим не ко всем формулам).

4.3. Исчисление высказываний

В этом разделе дано описание формальной теории исчисления высказываний. Поскольку исчисление высказываний уже знакомо читателю по материалам разделов 3.1 и 4.1, здесь сделан сильный акцент именно на *формальной* технике. Знакомство с чисто механическим характером формальных выводов подготавливает рассмотрение методов автоматического доказательства теорем в разделе 4.5.

4.3.1. Классическое определение исчисления высказываний

Исчисление высказываний — это формальная теория $\mathcal{L}$, в которой:

1. Алфавит: $\neg$ и $\to$ — *связки*;
   (,) — служебные символы;
   $a, b, \ldots, a_1, b_1, \ldots$ — *пропозициональные переменные*.
2. Формулы: 1) переменные суть формулы;
   2) если A, B — формулы, то $(\neg A)$ и $(A \to B)$ — формулы.
3. Аксиомы: $A_1 : (A \to (B \to A))$;
   $A_2 : ((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$;
   $A_3 : ((\neg B \to \neg A) \to ((\neg B \to A) \to B))$.
4. Правило: $\dfrac{A,\ A \to B}{B}$ (Modus ponens).

Здесь A, B и C — *любые* формулы. Таким образом, множество аксиом теории $\mathfrak{L}$ бесконечно, хотя задано тремя *схемами* аксиом. Множество правил вывода также бесконечно, хотя оно задано только одной *схемой*.

ЗАМЕЧАНИЕ

Латинское словосочетание *Modus ponens* обычно переводят на русский как *правило отделения* и обозначают MP.

При записи формул лишние скобки опускаются, если это не вызывает недоразумений. Другие связки вводятся определениями:

$$A \,\&\, B \stackrel{\text{Def}}{=} \neg(A \to \neg B), \qquad A \vee B \stackrel{\text{Def}}{=} \neg A \to B.$$

Любая формула, содержащая эти связки, рассматривается как синтаксическое сокращение собственной формулы теории $\mathfrak{L}$.

ОТСТУПЛЕНИЕ

Классическое задание теории $\mathfrak{L}$ с помощью схем аксиом над любыми формулами вполне соответствует математической традиции (мы пишем $(a+b)^2 = a^2+2ab+b^2$, подразумевая под a и b не только любые числа, но и любые выражения!). В то же время это не очень удобно для представления формальных теорий в программах. В следующих трёх подразделах вводится определение исчисления высказываний, более удобное для программной реализации.

4.3.2. Частный случай формулы

Если в формулу (исчисления высказываний) A вместо переменных $x_1, \ldots, x_n$ подставить, соответственно, формулы $B_1, \ldots, B_n$, то получится формула B, которая называется *частным случаем* формулы A:

$$B \stackrel{\text{Def}}{=} A(\ldots, x_i, \ldots)\,\{B_i//x_i\}_{i=1}^{n}.$$

Каждая формула B_i подставляется вместо *всех* вхождений переменной x_i. Набор подстановок $\{B_i//x_i\}_{i=1}^{n}$ называется *унификатором*.

Формула C называется *совместным* частным случаем формул A и B, если C является частным случаем формулы A и одновременно частным случаем формулы B при одном и том же наборе подстановок, то есть

$$C = A(\ldots, x_i, \ldots)\,\{X_i//x_i\}_{i=1}^{n} \quad \text{и} \quad C = B(\ldots, x_i, \ldots)\,\{X_i//x_i\}_{i=1}^{n}.$$

Формулы, которые имеют совместный частный случай, называются *унифицируемыми*, а набор подстановок $\{X_i//x_i\}_{i=1}^{n}$, с помощью которого получается совместный частный случай унифицируемых формул, называется *общим* унификатором. Наименьший возможный унификатор называется *наиболее общим унификатором*.

Набор формул $B_1, \ldots, B_n$ называется *частным случаем набора формул* $A_1, \ldots, A_n$, если каждая формула B_i является частным случаем формулы A_i при одном и том же наборе подстановок. Набор формул $C_1, \ldots, C_n$ называется *совместным* частным случаем наборов формул $A_1, \ldots, A_n$ и $B_1, \ldots, B_n$, если каждая формула C_i является частным случаем формул A_i и B_i при одном и том же наборе подстановок.

4.3.3. Алгоритм унификации

Если язык формул удовлетворяет определённым условиям, то можно проверить, являются ли две заданные формулы унифицируемыми, и если это так, то найти наиболее общий унификатор. В частности, это возможно для типичного языка формул, описанного в подразделе 4.3.1. В следующем алгоритме предполагается, что функция f осуществляет синтаксический разбор формулы и возвращает знак главной операции (то есть $\rightarrow$, $\neg$ или признак того, что формула является переменной), а функции l и r возвращают левый и правый операнды главной операции, то есть подформулы (считаем, что отрицание имеет только правый операнд). Набор подстановок (унификатор) представлен в виде глобального массива S, значениями индекса которого являются имена переменных, а значениями элементов — формулы, которые подставляются вместо соответствующих переменных.

Алгоритм 4.1 Рекурсивная функция Unify

Вход: формулы A и B.
Выход: **false**, если формулы не унифицируемы; **true** и наиболее общий унификатор S в противном случае.

```
if f(A) – переменная then
  v := f(A) { переменная }
  if S[v] = ∅ then
    S[v] := B; return true { то есть добавляем подстановку {B//v} }
  else
    return (S[v] = B) { либо эта подстановка уже есть, либо унификация
    невозможна }
  end if
end if
if f(A) ≠ f(B) then
  return false { главные операции различны – унификация невозможна }
end if
if f(A) =' ¬' then
  return Unify(r(A), r(B)) { пытаемся унифицировать операнды отрицаний }
end if
if f(A) ='→' then
  return Unify(l(A), l(B)) & Unify(r(A), r(B))
  { пытаемся унифицировать операнды импликаций }
end if
```

Обоснование При любых подстановках формул вместо переменных главная операция (связка) формулы остаётся неизменной. Поэтому если главные операции формул различны, то формулы заведомо не унифицируемы. В противном случае, то есть если главные операции совпадают, формулы унифицируемы тогда и только тогда, когда унифицируемы подформулы, являющиеся операндами главной операции. Эта рекурсия заканчивается, когда сопоставление доходит до переменных. Переменная унифицируется с любой формулой (в частности, с другой переменной) простой подстановкой этой формулы вместо переменной. Но подстановки для всех вхождений одной переменной должны совпадать. □

4.3.4. Конструктивное определение исчисления высказываний

Алфавит и множество формул — те же (см. 4.3.1). Аксиомы — три конкретные формулы:

A_1: $(a \to (b \to a))$;
A_2: $((a \to (b \to c)) \to ((a \to b) \to (a \to c)))$;
A_3: $((\neg b \to \neg a) \to ((\neg b \to a) \to b))$.

Правила вывода:

1. Правило подстановки: если формула B является частным случаем формулы A, то B непосредственно выводима из A.
2. Правило Modus ponens: если набор формул A, B, C является частным случаем набора формул a, $(a \to b)$, b, то формула C является непосредственно выводимой из формул A и B.

ЗАМЕЧАНИЕ

Здесь a, $(a \to b)$, b — это три конкретные формулы, построенные с помощью переменных a, b и связки $\to$.

4.3.5. Производные правила вывода

Исчисление высказываний $\mathfrak{L}$ — это достаточно богатая формальная теория, в которой выводимы многие важные теоремы.

ЗАМЕЧАНИЕ

Выводимость формул в теории $\mathfrak{L}$ доказывается путём предъявления конкретного вывода, то есть последовательности формул, удовлетворяющих определению, данному в разделе 4.2.2. Для удобства чтения формулы последовательности вывода выписываются друг под другом в столбик, слева указываются их номера в последовательности, а справа указывается, на каком основании формула включена в вывод (то есть является ли она гипотезой, или получена из схемы аксиом указанной подстановкой, или получена из предшествующих формул по указанному правилу вывода и т. д.).

ТЕОРЕМА 1 $\vdash_{\mathcal{L}} A \to A$.

Доказательство Вывод:

1. $(A \to ((A \to A) \to A))$	$A_1; \{A \to A/B\}$.
2. $((A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A)))$	$A_2; \{A \to A/B, A/C\}$.
3. $((A \to (A \to A)) \to (A \to A))$	MP; 1, 2.
4. $A \to (A \to A)$	$A_1; A/B$.
5. $A \to A$	MP; 4, 3. □

ТЕОРЕМА 2 $A \vdash_{\mathcal{L}} B \to A$.

Доказательство Вывод:

1. A	гипотеза.
2. $A \to (B \to A)$	A_1.
3. $B \to A$	MP; 1,2. □

Всякую доказанную выводимость можно использовать как новое (*производное*) правило вывода. Например, последняя доказанная выводимость называется *правилом введения импликации*:

$$\frac{A}{B \to A}\ (\to^{+}).$$

4.3.6. Дедукция

В теории $\mathcal{L}$ импликация очень тесно связана с выводимостью.

ТЕОРЕМА (дедукции) $\Gamma, A \vdash_{\mathcal{L}} B$ *тогда и только тогда, когда* $\Gamma \vdash_{\mathcal{L}} A \to B$.

Доказательство

[Необходимость] Пусть $E_1, \ldots, E_n$ — вывод B из Γ, A. $E_n = B$. Индукцией по i $(1 \leqslant i \leqslant n)$ покажем, что $\Gamma \vdash_{\mathcal{L}} A \to E_i$. Тем самым теорема будет доказана. База: $i = 1$. Возможны три случая.

1. Пусть E_1 — аксиома. Тогда рассмотрим вывод E_1, $E_1 \to (A \to E_1)$, $A \to E_1$. Имеем $\vdash_{\mathcal{L}} A \to E_1$.
2. Пусть $E_1 \in \Gamma$. Тогда рассмотрим вывод E_1, $E_1 \to (A \to E_1)$, $A \to E_1$. Имеем $\Gamma \vdash_{\mathcal{L}} A \to E_1$.
3. Пусть $E_1 = A$. Тогда по первой теореме предыдущего подраздела $\vdash_{\mathcal{L}} E_1 \to E_1$, а значит, $\vdash_{\mathcal{L}} A \to E_1$.

В любом случае $\Gamma \vdash_{\mathcal{L}} A \to E_1$. Таким образом, база индукции доказана. Пусть теперь $\Gamma \vdash_{\mathcal{L}} A \to E_i$ для всех $i < k$. Рассмотрим E_k. Возможны четыре случая: либо E_k — аксиома, либо $E_k \in \Gamma$, либо $E_k = A$, либо формула E_k получена по правилу Modus ponens из формул E_i и E_j, причём $i, j < k$ и $E_j = E_i \to E_k$. Для первых трёх случаев имеем $\Gamma \vdash_{\mathcal{L}} A \to E_k$ аналогичным образом. Для четвёртого случая по индукционному предположению имеется вывод $\Gamma \vdash_{\mathcal{L}} A \to E_i$ и вывод $\Gamma \vdash_{\mathcal{L}} A \to (E_i \to E_k)$. Объединим эти выводы и достроим следующий вывод:

$n.$	$(A \to (E_i \to E_k)) \to ((A \to E_i) \to (A \to E_k))$	$A_2; \{E_i/B, E_k/C\}$.
$n+1.$	$(A \to E_i) \to (A \to E_k)$	MP; j, n.
$n+2.$	$A \to E_k$	MP; i, $n+1$.

Таким образом, $\Gamma \vdash_{\mathcal{L}} A \to E_k$ для любого k, в частности, для $k = n$. Но $E_n = B$, то есть $\Gamma \vdash_{\mathcal{L}} A \to B$.

[Достаточность] Имеем вывод $\Gamma \vdash_{\mathcal{L}} A \to B$, состоящий из n формул. Достроим его следующим образом:

$n.$	$A \to B.$	
$n+1.$	A	гипотеза.
$n+2.$	B	MP; $n+1$, n.

Таким образом, имеем $\Gamma, A \vdash_{\mathcal{L}} B$. □

ЗАМЕЧАНИЕ

Схема аксиом A_3 теории $\mathcal{L}$ не использовалась в доказательстве, поэтому теорема дедукции имеет место не только для теории $\mathcal{L}$, но и для любых теорий, в которых выполнены аксиомы A_1 и A_2 и действует правило отделения.

СЛЕДСТВИЕ 1 *$A \vdash_{\mathcal{L}} B$ тогда и только тогда, когда $\vdash_{\mathcal{L}} A \to B$.*

Доказательство $\Gamma := \varnothing$. □

СЛЕДСТВИЕ 2 $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$.

Доказательство Вывод:

1. $A \to B$	гипотеза.
2. $B \to C$	гипотеза.
3. A	гипотеза.
4. B	MP; 3, 1.
5. C	MP; 4, 2.
6. $A \to B, B \to C, A \vdash_{\mathcal{L}} C$	1–5.
7. $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$	по теореме дедукции.

□

ЗАМЕЧАНИЕ

Это производное правило называется *правилом транзитивности.*

СЛЕДСТВИЕ 3 $A \to (B \to C), B \vdash_{\mathfrak{L}} A \to C.$

Доказательство Вывод:

1. $A \to (B \to C)$	гипотеза.
2. A	гипотеза.
3. $B \to C$	MP; 2, 1.
4. B	гипотеза.
5. C	MP; 4, 3.
6. $A \to (B \to C), B, A \vdash_{\mathfrak{L}} C$	1–5.
7. $A \to (B \to C), B \vdash_{\mathfrak{L}} A \to C$	по теореме дедукции.

□

ЗАМЕЧАНИЕ

Это производное правило называется *правилом сечения.*

4.3.7. Некоторые теоремы исчисления высказываний

Множество теорем теории $\mathfrak{L}$ бесконечно. Здесь приведены некоторые теоремы, которые используются в дальнейших построениях.

ЗАМЕЧАНИЕ

Каждая доказанная теорема (то есть формула теории, для которой построен вывод) может использоваться в дальнейших выводах на правах аксиомы.

ТЕОРЕМА *В теории $\mathfrak{L}$ выводимы следующие теоремы:*

а) $\vdash_{\mathfrak{L}} \neg\neg A \to A.$
б) $\vdash_{\mathfrak{L}} A \to \neg\neg A.$
в) $\vdash_{\mathfrak{L}} \neg A \to (A \to B).$
г) $\vdash_{\mathfrak{L}} (\neg B \to \neg A) \to (A \to B).$
д) $\vdash_{\mathfrak{L}} (A \to B) \to (\neg B \to \neg A).$
е) $\vdash_{\mathfrak{L}} A \to (\neg B \to \neg(A \to B)).$
ж) $\vdash_{\mathfrak{L}} (A \to B) \to ((\neg A \to B) \to B).$

Доказательство

[*а*) $\vdash_{\mathcal{L}} \neg\neg A \to A$]

1.	$(\neg A \to \neg\neg\neg A) \to ((\neg A \to \neg A) \to A)$	A_3; $\{A/B, \neg A/A\}$.
2.	$\neg A \to \neg A$	первая теорема 4.3.5; $\{\neg A/A\}$.
3.	$(\neg A \to \neg\neg A) \to A$	Сл. 3; $\{\neg A \to \neg\neg A/A, \neg A \to A/B, A/C\}$.
4.	$\neg\neg A \to (\neg A \to \neg\neg A)$	A_1; $\{\neg\neg A/A, \neg A/B\}$.
5.	$\neg\neg A \to A$	Сл. 2; $\{\neg\neg A/A, \neg A \to \neg\neg A/B, A/C\}$.

[*б*) $\vdash_{\mathcal{L}} A \to \neg\neg A$]

1.	$(\neg\neg\neg A \to \neg A) \to .$ $\to ((\neg\neg\neg A \to A) \to \neg\neg A)$	A_3; $\{\neg\neg A/B\}$.
2.	$\neg\neg\neg A \to \neg A$	*а*; $\{\neg A/A\}$.
3.	$(\neg\neg\neg A \to A) \to \neg\neg A$	MP; 2, 1.
4.	$A \to (\neg\neg\neg A \to A)$	A_1; $\{\neg\neg\neg A/B\}$.
5.	$A \to \neg\neg A$	Сл. 2; $\{\neg\neg\neg A/B, A/A, \neg\neg A/C\}$.

[*в*) $\vdash_{\mathcal{L}} \neg A \to (A \to B)$]

1.	$\neg A$	гипотеза.
2.	A	гипотеза.
3.	$A \to (\neg B \to A)$	A_1; $\{\neg B/B\}$.
4.	$\neg A \to (\neg B \to \neg A)$	A_1; $\{\neg A/A, \neg B/B\}$.
5.	$\neg B \to A$	MP; 2, 3.
6.	$\neg B \to \neg A$	MP; 1, 4.
7.	$(\neg B \to \neg A) \to ((\neg B \to A) \to B)$	A_3.
8.	$(\neg B \to A) \to B$	MP; 6, 7.
9.	B	MP; 5, 8.
10.	$\neg A, A \vdash_{\mathcal{L}} B$	1–9.
11.	$\neg A \vdash_{\mathcal{L}} A \to B$	теорема дедукции.
12.	$\vdash_{\mathcal{L}} \neg A \to (A \to B)$	теорема дедукции.

[*г*) $\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$]

1.	$\neg B \to \neg A$	гипотеза.
2.	A	гипотеза.
3.	$(\neg B \to \neg A) \to ((\neg B \to A) \to B)$	A_3.
4.	$A \to (\neg B \to A)$	A_1; $\{\neg B/B\}$.
5.	$(\neg B \to A) \to B$	MP; 1, 3.
6.	$A \to B$	Сл. 2; $\{\neg B \to A/B, B/C\}$.
7.	B	MP; 2, 6.
8.	$\neg B \to \neg A, A \vdash_{\mathcal{L}} B$	1–7.
9.	$\neg B \to \neg A \vdash_{\mathcal{L}} A \to B$	теорема дедукции.
10.	$\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$	теорема дедукции.

[*д)* $\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$]

1.	$A \to B$	гипотеза.
2.	$\neg\neg A \to A$	*а*.
3.	$\neg\neg A \to B$	Сл. 2; $\{A/B, \neg\neg A/A, B/C\}$.
4.	$B \to \neg\neg B$	*б*; $\{B/A\}$.
5.	$\neg\neg A \to \neg\neg B$	Сл. 2; $\{\neg\neg A/A, \neg\neg B/C\}$.
6.	$(\neg\neg A \to \neg\neg B) \to (\neg B \to \neg A)$	*г*; $\{\neg A/B, \neg B/A\}$.
7.	$\neg B \to \neg A$	MP; 5, 6.
8.	$A \to B \vdash_{\mathcal{L}} \neg B \to \neg A$	1–7.
9.	$\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$	теорема дедукции.

[*е)* $\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$]

1.	A	гипотеза.
2.	$A \to B$	гипотеза.
3.	B	MP; 1, 2.
4.	$A, A \to B \vdash_{\mathcal{L}} B$	1–3.
5.	$A \vdash_{\mathcal{L}} (A \to B) \to B$	теорема дедукции.
6.	$\vdash_{\mathcal{L}} A \to ((A \to B) \to B)$	теорема дедукции.
7.	$\vdash_{\mathcal{L}} ((A \to B) \to B) \to$ $\to (\neg B \to \neg(A \to B))$	*д*; $\{A \to B/A\}$.
8.	$\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$	Сл. 2; $\{((A \to B) \to B)/B;$ $(\neg B \to \neg(A \to B))/C\}$.

[*ж)* $\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$]

1.	$A \to B$	гипотеза.
2.	$\neg A \to B$	гипотеза.
3.	$(A \to B) \to (\neg B \to \neg A)$	*д*.
4.	$\neg B \to \neg A$	MP; 1, 3.
5.	$(\neg A \to B) \to (\neg B \to \neg\neg A)$	*д*; $\{\neg A/A\}$.
6.	$\neg B \to \neg\neg A$	MP; 2, 5.
7.	$(\neg B \to \neg\neg A) \to ((\neg B \to \neg A) \to B)$	A_3; $\{\neg A/A\}$.
8.	$(\neg B \to \neg A) \to B$	MP; 6, 7.
9.	B	MP; 4, 8.
10.	$A \to B, \neg A \to B \vdash_{\mathcal{L}} B$	1–9.
11.	$A \to B \vdash_{\mathcal{L}} (\neg A \to B) \to B$	теорема дедукции.
12.	$\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$	теорема дедукции. □

4.3.8. Множество теорем исчисления высказываний

Пусть формула A содержит переменные $a_1, \ldots, a_n$, и пусть задана некоторая интерпретация I формулы A, то есть переменным $a_1, \ldots, a_n$ приписаны истинностные значения. Обозначим

$$A'_i \stackrel{\text{Def}}{=} \begin{cases} a_i, & \text{если } a_i = \text{T}, \\ \neg a_i, & \text{если } a_i = \text{F}; \end{cases} \qquad A' \stackrel{\text{Def}}{=} \begin{cases} A, & \text{если } I(A) = \text{T}, \\ \neg A, & \text{если } I(A) = \text{F} \end{cases}$$

в данной интерпретации.

ЛЕММА $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A'$.

Доказательство Индукция по структуре формулы A.

[Переменная] Пусть $A = a$. Тогда $a \vdash_{\mathcal{L}} a$ и $\neg a \vdash_{\mathcal{L}} \neg a$.

[Отрицание] Пусть $A = \neg B$ и I — произвольная интерпретация формулы A. Возможны два случая: $I(B) = \text{T}$ или $I(B) = \text{F}$. Пусть $I(B) = \text{T}$. Тогда $I(A) = \text{F}$ и $A' = \neg A = \neg\neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B$. Но $\vdash_{\mathcal{L}} B \to \neg\neg B$ по теореме 4.3.7, *б*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg\neg B = A'$. Пусть теперь $I(B) = \text{F}$. Тогда $I(A) = \text{T}$ и $A' = A = \neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg B = A'$.

[Импликация] Пусть $A = (B \to C)$ и I — произвольная интерпретация формулы A. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B'$ и $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} C'$. Возможны три случая: $I(B) = \text{F}$ или $I(B) = \text{T}$ и $I(C) = \text{T}$ или же $I(B) = \text{T}$ и $I(C) = \text{F}$.

Пусть $I(B) = \text{F}$. Тогда независимо от значения $I(C)$ имеем: $I(A) = \text{T}$ и $B' = \neg B$, $A' = A$. Но $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg B$, $\vdash_{\mathcal{L}} \neg B \to (B \to C)$ по теореме 4.3.7, *в*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B \to C = A'$.

Пусть $I(B) = \text{T}$ и $I(C) = \text{T}$. Тогда $I(A) = \text{T}$ и $C' = C$, $A' = A = B \to C$. Имеем: $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} C$, $\vdash_{\mathcal{L}} C \to (B \to C)$ (аксиома A_1 с подстановкой $\{C/A\}$), следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B \to C = A'$.

Пусть теперь $I(B) = \text{T}$ и $I(C) = \text{F}$. Тогда $A' = \neg A = \neg(B \to C)$, $B' = B$ и $C' = \neg C$. Имеем: $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B$, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg C$, $\vdash_{\mathcal{L}} B \to (\neg C \to \neg(B \to C))$ по теореме 4.3.7, *е*. Следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg(B \to C) = A'$. □

ТЕОРЕМА *Теоремами теории* $\mathcal{L}$ *являются общезначимые формулы и только они:*

$$\vdash_{\mathcal{L}} A \iff A \text{ — тавтология.}$$

Доказательство

[$\Longleftarrow$] Пусть A — тавтология. Тогда $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A$ в любой интерпретации. Таким образом, имеется 2^n различных выводимостей $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A$. Среди них есть две, которые различаются в A'_n: $A'_1, \ldots, A'_{n-1}, a_n \vdash_{\mathcal{L}} A$ и $A'_1, \ldots, A'_{n-1}, \neg a_n \vdash_{\mathcal{L}} A$. По теореме дедукции $A'_1, \ldots, A'_{n-1} \vdash_{\mathcal{L}} a_n \to A$ и $A'_1, \ldots, A'_{n-1} \vdash_{\mathcal{L}} \neg a_n \to A$. Но $\vdash_{\mathcal{L}} (a_n \to A) \to ((\neg a_n \to A) \to A)$ по теореме 4.3.7, *ж*, с подстановкой $a_n//A$,

$A//B$, следовательно, $A'_1, \ldots, A'_{n-1} \vdash_{\mathfrak{L}} A$. Повторив этот процесс ещё $n-1$ раз, имеем $\vdash_{\mathfrak{L}} A$.

[$\Longrightarrow$] Аксиомы A_1, A_2, A_3 суть тавтологии. Правило Modus ponens сохраняет тавтологичпость (см. 4.1.3), следовательно, теоремы теории $\mathfrak{L}$ суть тавтологии. □

СЛЕДСТВИЕ *Теория $\mathfrak{L}$ формально непротиворечива.*

Доказательство Все теоремы $\mathfrak{L}$ суть тавтологии. Отрицание тавтологии пе есть тавтология. Следовательно, в $\mathfrak{L}$ пет теоремы и её отрицания. □

4.3.9. Другие аксиоматизации исчисления высказываний

Теория $\mathfrak{L}$ пе является единственной возможной аксиоматизацией исчисления высказываний. Её основное достоинство — лаконичность при сохранении определённой наглядности. Действительно, в теории $\mathfrak{L}$ всего две связки, три схемы аксиом и одно правило. Известны и многие другие аксиоматизации исчисления высказываний, предложенные различными авторами.

1. Гильберт[1] и Аккерман[2], 1938.

   Связки: $\vee, \neg$; $(A \to B \overset{\text{Def}}{=} \neg A \vee B)$.
   Аксиомы:
   $A \vee A \to A$,
   $A \to A \vee B$,
   $A \vee B \to B \vee A$,
   $(B \to C) \to (A \vee B \to A \vee C)$.
   Правило: Modus ponens.

2. Россер[3], 1953.

   Связки: $\&, \neg$; $(A \to B \overset{\text{Def}}{=} \neg(A \,\&\, \neg B))$.
   Аксиомы:
   $A \to A \,\&\, A$,
   $A \,\&\, B \to A$,
   $(A \to B) \to (\neg(B \,\&\, C) \to \neg(C \,\&\, A))$.
   Правило: Modus ponens.

3. Клини[4], 1952.

   Связки: $\neg$, $\&$, $\vee$, $\to$.
   Аксиомы:
   $A \to (B \to A)$,
   $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$,
   $A \,\&\, B \to A$,
   $A \,\&\, B \to B$,
   $A \to (B \to (A \,\&\, B))$,
   $A \to (A \vee B)$,

[1] Давид Гильберт (1862–1943).
[2] Вильгельм Аккерман (1896–1962).
[3] Джон Беркли Россер (1907–1989).
[4] Стефан Клини (1909–1994).

$B \to (A \vee B)$,
$(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$,
$(A \to B) \to ((A \to \neg B) \to \neg A)$,
$\neg\neg A \to A$.

Правило: Modus ponens.

4. Никод[1], 1917.

Связка: $|; \quad (A \mid B = \neg A \vee \neg B)$.

Аксиома: $(A \mid (B \mid C)) \mid ((D \mid (D \mid D)) \mid$
$\mid ((E \mid B) \mid ((A \mid E) \mid (A \mid E))))$.

Правило: $\dfrac{A, A \mid (B \mid C)}{C}$.

4.4. Исчисление предикатов

Описание формальной теории исчисления предикатов в этом разделе носит конспективный характер, в частности, многие технически сложные доказательства опущены. В то же время уделено внимание прагматическим аспектам теории, то есть ответу на вопрос, что выразимо и что невыразимо в исчислении предикатов.

4.4.1. Определения

(Чистое) исчисление предикатов (первого порядка) – это формальная теория $\mathcal{K}$, в которой определены следующие компоненты: алфавит, формулы, аксиомы и правила вывода.

1. Алфавит:

связки	основные	$\neg$, $\to$
	дополнительные	$\&$, $\vee$
служебные символы		(,)
кванторы	*всеобщности*	$\forall$
	существования	$\exists$
предметные	*константы*	$a, b, \ldots, a_1, b_1, \ldots$
	переменные	$x, y, \ldots, x_1, y_1, \ldots$
предметные	*предикаты*	$P, Q, \ldots$
	функциональные символы	$f, g, \ldots$

С каждым предикатом и функциональным символом связано некоторое натуральное число n, которое называется *арностью*, *местностью* или *вместимостью*.

2. Формулы имеют следующий синтаксис:

⟨формула⟩ ::= ⟨атом⟩ |
$\neg$⟨формула⟩ |
(⟨формула⟩ $\to$ ⟨формула⟩) |
$\forall$ ⟨переменная⟩ (⟨формула⟩) |

[1] Жан Никод (1893–1924).

		$\exists$ ⟨переменная⟩ (⟨формула⟩)
⟨атом⟩	::=	⟨предикат⟩(⟨список термов⟩)
⟨список термов⟩	::=	⟨терм⟩ | ⟨терм⟩, ⟨список термов⟩
⟨терм⟩	::=	⟨константа⟩ | ⟨переменная⟩ | ⟨функциональный символ⟩(⟨список термов⟩)

При этом должны быть выполнены следующие контекстные условия: в *терме* $f(t_1, \ldots, t_n)$ функциональный символ f должен быть n-местным. В *атоме* (или *атомарной* формуле) $P(t_1, \ldots, t_n)$ предикат P должен быть n-местным.

Вхождения переменных в атомарную формулу называются *свободными*. Свободные вхождения переменных в формулах A и B по определению считаются свободными в формулах $\neg A$ и $A \to B$. В формулах $\forall x\ (A)$ и $\exists x\ (A)$ формула A, как правило, имеет свободные вхождения переменной x. Вхождения переменной x в формулы $\forall x\ (A)$ и $\exists x\ (A)$ называются *связанными*. Вхождения других переменных (отличных от x), которые были свободными в формуле A, по определению считаются свободными и в формулах $\forall x\ (A)$ и $\exists x\ (A)$. Одна и та же переменная может иметь в одной и той же формуле как свободные, так и связанные вхождения. Если все вхождения переменной в формулу связаны, то переменная называется *связанной* (в данной формуле). Формула, не содержащая свободных вхождений переменных, называется *замкнутой*. В замкнутой формуле все переменные связаны.

Пример Рассмотрим формулу $\forall x\ ((P(x) \to \exists y\ (Q(x, y))))$ и её подформулы. В подформулу $\exists y\ (Q(x, y))$ переменная x входит свободно, а все вхождения переменной y связаны (квантором существования). Таким образом, эта подформула не замкнута. С другой стороны, то же самое вхождение переменной x в подформулу $Q(x, y)$ является связанным в формуле $\forall x\ (P(x) \to \exists y\ (Q(x, y)))$. В этой формуле все вхождения всех переменных связаны, а потому формула замкнута.

ЗАМЕЧАНИЕ

Язык теории $\mathfrak{L}$ не содержит кванторов, поэтому понятия свободного и связанного вхождения переменных не применимы непосредственно. Можно расширить исчисление высказываний, допуская замкнутые формулы исчисления предикатов наравне с пропозициональными переменными. В таком случае для удобства обычно полагают, что все формулы теории $\mathfrak{L}$ замкнуты.

Формулы вида A и $\neg A$, где A — атом, называются *литеральными* формулами (или *литералами*).

В формулах $\forall x\ (A)$ и $\exists x\ (A)$ подформула A называется *областью действия* квантора по x.

Обычно связки и кванторы упорядочивают по приоритету следующим образом: $\neg, \forall, \exists, \&, \vee, \to$. Лишние скобки при этом иногда опускают.

Терм t называется *свободным* для переменной x в формуле A, если никакое свободное вхождение переменной x в формулу A не лежит в области действия никакого квантора по переменной y, входящей в терм t. В частности, терм t свободен для любой переменной в формуле A, если никакая переменная терма не является связанной переменной формулы A.

Пример Терм y свободен для переменной x в формуле $P(x)$, но тот же терм y не свободен для переменной x в формуле $\forall y\ (P(x))$. Терм $f(x,z)$ свободен для переменной x в формуле $\forall y\ (P(x,y) \to Q(x))$, но тот же терм $f(x,z)$ не свободен для переменной x в формуле $\exists z\ (\forall y\ (P(x,y) \to Q(x)))$.

3. Аксиомы (логические): любая система аксиом исчисления высказываний плюс

$$P_1\colon\quad \forall x\ (A(x) \to A(t)),$$
$$P_2\colon\quad A(t) \to \exists x\ (A(x)),$$

где терм t свободен для переменной x в формуле A.

4. Правила вывода:

$$\frac{A,\ A \to B}{B}\ \text{(Modus ponens)},\qquad \frac{B \to A(x)}{B \to \forall x\ (A(x))}\ (\forall^+),\qquad \frac{A(x) \to B}{\exists x\ (A(x)) \to B}\ (\exists^+),$$

где формула A содержит свободные вхождения переменной x, а формула B их не содержит.

Исчисление предикатов, которое не содержит предметных констант, функциональных символов, предикатов и собственных аксиом, называется *чистым*. Исчисление предикатов, которое содержит предметные константы и/или функциональные символы и/или предикаты и связывающие их собственные аксиомы, называется *прикладным*.

Исчисление предикатов, в котором кванторы могут связывать только предметные переменные, но не могут связывать функциональные символы или предикаты, называется исчислением *первого порядка*. Исчисления, в которых кванторы могут связывать не только предметные переменные, но и функциональные символы, предикаты или иные множества объектов, называются исчислениями *высших порядков*.

Практика показывает, что прикладного исчисления предикатов первого порядка оказывается достаточно для формализации содержательных теорий во всех разумных случаях.

4.4.2. Интерпретация

Интерпретация I (прикладного) исчисления предикатов $\mathcal{K}$ с областью интерпретации (или *носителем*) M — это набор функций, которые сопоставляют:

1) каждой предметной константе a элемент носителя $I(a)$, $I(a) \in M$;

2) каждому n-местному функциональному символу f операцию $I(f)$ на носителе, $I(f)\colon M^n \to M$;

3) каждому n-местному предикату P отношение $I(P)$ на носителе, $I(P) \subset M^n$.

Пусть $x = (x_1, \dots)$ — набор (последовательность) переменных (входящих в формулу), а $s = (s_1, \dots)$ — набор значений из M. Тогда всякий терм $f(t_1, \dots, t_n)$ имеет значение на s (из области M), то есть существует функция $s^*\colon \{t\} \to M$, определяемая следующим образом:

$$s^*(a) \stackrel{\text{Def}}{=} I(a), \quad s^*(x_i) \stackrel{\text{Def}}{=} s_i, \quad s^*(f(t_1, \dots, t_n)) \stackrel{\text{Def}}{=} I(f)(s^*(t_1), \dots, s^*(t_n)).$$

Всякий атом $P(t_1, \dots, t_n)$ имеет на s истинностное значение $s^*(P)$, определяемое следующим образом:

$$s^*(P(t_1, \dots, t_n)) \stackrel{\text{Def}}{=} (s^*(t_1), \dots, s^*(t_n)) \in I(P).$$

Если $s^*(P) = \mathrm{T}$, то говорят, что формула P *выполнена* на s.

Формула $\neg A$ выполнена на s тогда и только тогда, когда формула A не выполнена на s.

Формула $A \to B$ выполнена на s тогда и только тогда, когда формула A не выполнена на s или формула B выполнена на s.

Формула $\forall x_i\ (A)$ выполнена на s тогда и только тогда, когда формула A выполнена на любом наборе s', отличающемся от s, возможно, только i-м компонентом.

Формула $\exists x_i\ (A)$ выполнена на s тогда и только тогда, когда формула A выполнена на каком-либо наборе s', отличающемся от s, возможно, только i-м компонентом.

Формула называется *истинной* в данной интерпретации I, если она выполнена на любом наборе s элементов M. Формула называется *ложной* в данной интерпретации I, если она не выполнена ни на одном наборе s элементов M.

Интерпретация называется *моделью* множества формул Γ, если все формулы из Γ истинны в данной интерпретации.

Всякая замкнутая формула истинна или ложна в данной интерпретации. *Открытая* (то есть *не* замкнутая) формула $A(x, y, z, \dots)$ истинна в данной интерпретации тогда и только тогда, когда её *замыкание* $\forall x\ (\forall y\ (\forall z \cdots A(x, y, z, \dots)))$ истинно в данной интерпретации.

ЗАМЕЧАНИЕ

Это обстоятельство объясняет, почему собственные аксиомы прикладных теорий обычно пишутся в открытой форме.

4.4.3. Общезначимость

Формула (исчисления предикатов) *общезначима*, если она истинна в любой интерпретации.

ТЕОРЕМА *Формула* $\forall x\ (A(x)) \to A(t)$, *где терм* t *свободен для переменной* x *в формуле* A, *общезначима.*

Доказательство Рассмотрим произвольную интерпретацию I, произвольную последовательность значений из области интерпретации s и соответствующую функцию s^*. Заметим следующее. Пусть t_1 — терм и $s^*(t_1) = a_1$. Пусть $t(x)$ — некоторый другой терм, а $t' := t(\ldots x \ldots)\{t_1//x\}$. Тогда $s^*(t) = s_1^*(t')$, где s_1 имеет значение a_1 на месте x. Пусть теперь $A(x)$ — формула, а терм t свободен для x в A. Положим $A(t) := A(\ldots x \ldots)\{t//x\}$. Имеем: $s^*(A(t)) = \mathrm{T} \iff s_1^*(A(x)) = \mathrm{T}$, где s_1 имеет значение $s^*(t)$ на месте x. Если $s^*(\forall x\ (A(x))) = \mathrm{T}$ и терм t свободен для x в A, то $s^*(A(t)) = \mathrm{T}$. Следовательно, формула $\forall x\ (A(x)) \to A(t)$ выполнена на всех последовательностях в произвольной интерпретации. □

ЗАМЕЧАНИЕ

Можно показать, что формула $A(t) \to \exists x\ (A)(x)$, где терм t свободен для переменной x в формуле A, общезначима.

4.4.4. Непротиворечивость и полнота чистого исчисления предикатов

Нам потребуется следующее преобразование h формулы исчисления предикатов в формулу исчисления высказываний: в формуле опускаются все кванторы и термы вместе с соответствующими скобками и запятыми, остаются предикатные символы (которые рассматриваются как пропозициональные переменные) и связки.

Пример $h(\forall x\ (P(x) \to \exists y\ (Q(x,y)))) = P \to Q$.

Такое преобразование как бы «забывает» всё, что связано с исчислением предикатов, оставляя только *пропозициональную структуру*, которая является формулой исчисления высказываний.

Нетрудно видеть, что $h(\neg A) = \neg h(A)$ и $h(A \to B) = h(A) \to h(B)$.

ТЕОРЕМА 1 *Формальная теория $\mathcal{K}$ непротиворечива.*

Доказательство Рассмотрим пропозициональную структуру аксиом исчисления предикатов $\mathcal{K}$, то есть применим к аксиомам преобразование h. Ясно, что все полученные формулы суть тавтологии, поскольку аксиомы A_1, A_2, A_3 сами по себе совпадают со своими пропозициональными структурами, которые являются тавтологиями, а для аксиом исчисления предикатов имеем $h(P_1) = h(\forall x\ (A(x)) \to A(t)) = A \to A$ и $h(P_2) = h(A(t) \to \exists x\ (A(x))) = A \to A$. Далее, правило Modus ponens сохраняет тавтологичность пропозициональной структуры, то есть если $h(A)$ и $h(A \to B)$ — тавтологии, то $h(B)$ — тавтология. Правила $\exists^+$ и $\forall^+$ также сохраняют тавтологичность пропозициональной структуры, поскольку в этих правилах пропозициональная структура заключе-

ния совпадает с пропозициональной структурой посылки. Таким образом, если формула A является теоремой теории $\mathcal{K}$, то формула $h(A)$ является тавтологией. Далее от противного. Если $\vdash_{\mathcal{K}} A$ и $\vdash_{\mathcal{K}} \neg A$, то $h(A)$ и $\neg h(A)$ — тавтологии, что невозможно. □

Следующие две метатеоремы устанавливают свойства полноты исчисления предикатов, аналогичные тем, которые установлены для исчисления высказываний в подразделе 4.3.8. Теоремы приводятся без доказательств, которые технически довольно громоздки.

ТЕОРЕМА 2 *Всякая теорема чистого исчисления предикатов первого порядка общезначима.*

ТЕОРЕМА 3 *Всякая общезначимая формула является теоремой чистого исчисления предикатов первого порядка.*

4.4.5. Логическое следование и логическая эквивалентность

Формула B является *логическим следствием* формулы A (обозначение $A \Longrightarrow B$), если формула B выполнена на любом наборе в любой интерпретации, на котором выполнена формула A. Формулы A и B *логически эквивалентны* (обозначение $A = B$), если они являются логическим следствием друг друга. Можно показать, что имеют место следующие логические следования и эквивалентности:

1. $\neg\forall x\ (A(x)) = \exists x\ (\neg A(x))$.
2. $\neg\exists x\ (A(x)) = \forall x\ (\neg A(x))$.
3. $\forall x\ ((A(x)\ \&\ B(x))) = \forall x\ (A(x))\ \&\ \forall x\ (B(x))$.
4. $\exists x\ (A(x) \vee B(x)) \Longrightarrow \exists x\ (A(x)) \vee \exists x\ (B(x))$.
5. $\exists x\ (A(x)\ \&\ B(x)) \Longrightarrow \exists x\ (A(x))\ \&\ \exists x\ (B(x))$.
6. $\forall x\ (A(x)) \vee \forall x\ (B(x)) \Longrightarrow \forall x\ (A(x) \vee B(x))$.
7. $\forall x\ (\forall y\ (A(x,y))) = \forall y\ (\forall x\ (A(x,y)))$.
8. $\exists x\ (\exists y\ (A(x,y))) = \exists y\ (\exists x\ (A(x,y)))$.
9. $\forall x\ (A(x)\ \&\ C) = \forall x\ (A(x))\ \&\ C$.
10. $\forall x\ (A(x) \vee C) = \forall x\ (A(x)) \vee C$.
11. $\exists x\ (A(x)\ \&\ C) = \exists x\ (A(x))\ \&\ C$.
12. $\exists x\ (A(x) \vee C) = \exists x\ (A(x)) \vee C$.
13. $C \to \forall x\ (A(x)) = \forall x\ (C \to A(x))$.
14. $C \to \exists x\ (A(x)) = \exists x\ (C \to A(x))$.
15. $\exists x\ (\forall y\ (A(x,y))) \Longrightarrow \forall y\ (\exists x\ (A(x,y)))$.
16. $\forall x\ (A(x) \to B(x)) \Longrightarrow \forall x\ (A(x)) \to \forall x\ (B(x))$,

где формула C не содержит никаких вхождений переменной x.

Для всякой формулы A существует логически эквивалентная ей формула A', имеющая *предварённую форму*:

$$A' = Q_1 x_1 \dots Q_n x_n \ \bar{A}(x_1, \dots, x_n),$$

где $Q_1, \dots, Q_n$ — некоторые кванторы, а $\bar{A}$ — *бесквантор*ная формула.

4.4.6. Теория равенства

Этот подраздел начинает серию примеров прикладных исчислений предикатов.

ЗАМЕЧАНИЕ

Напомним, что всякое прикладное исчисление предикатов содержит в себе чистое исчисление предикатов, поэтому при описании прикладного исчисления аксиомы и все теоремы чистого исчисления подразумеваются, указываются только собственные функциональные символы, предикаты и аксиомы.

Теория равенства $\mathcal{E}$ — это прикладное исчисление предикатов, в котором имеются:

1. Собственный двухместный предикат = (инфиксная запись $x = y$).
2. Собственные аксиомы (схемы аксиом):

E_1: $\forall x \ (x = x)$,
E_2: $(x = y) \to (A(x) \to A(x)\{y/x\})$.

ТЕОРЕМА *В теории* $\mathcal{E}$ *выводимы следующие теоремы:*

1. $\vdash_{\mathcal{E}} t = t$ *для любого терма* t.
2. $\vdash_{\mathcal{E}} x = y \to y = x$.
3. $\vdash_{\mathcal{E}} x = y \to (y = z \to x = z)$.

Доказательство

[1] Из E_1 и P_1 по Modus ponens.

[2] Имеем: $(x = y) \to (x = x \to y = x)$ из E_2 при подстановке $\{x = x/A(x)\}$. Значит, $x = y, x = x \vdash_{\mathcal{E}} y = x$. Но $\vdash_{\mathcal{E}} x = x$, следовательно, $x = y \vdash_{\mathcal{E}} y = x$. По теореме дедукции $\vdash_{\mathcal{E}} x = y \to y = x$.

[3] Имеем: $y = x \to (y = z \to x = z)$ из E_2 при подстановке $\{x = z/A(x), y/x, x/y\}$, значит, $y = x \vdash_{\mathcal{E}} (y = z \to x = z)$. Но $x = y \vdash_{\mathcal{E}} y = x$, по транзитивности $x = y \vdash_{\mathcal{E}} (y = z \to x = z)$, по теореме дедукции $\vdash_{\mathcal{E}} (x = y) \to (y = z \to x = z)$. □

Таким образом, равенство является эквивалентностью. Но равенство — это более сильное свойство, чем эквивалентность. Аксиома E_2 выражает *неотличимость* равных элементов.

4.4.7. Формальная арифметика

Формальная арифметика A — это прикладное исчисление предикатов, в котором имеются:

1. Предметная константа 0.
2. Двухместные функциональные символы $+$ и $\cdot$, одноместный функциональный символ $'$.
3. Двухместный предикат $=$.
4. Собственные аксиомы (схемы аксиом):

A_1: $(P(0)\ \&\ \forall x\ (P(x) \to P(x'))) \to \forall x\ (P(x))$,
A_2: $t_1' = t_2' \to t_1 = t_2$,
A_3: $\neg(t' = 0)$,
A_4: $t_1 = t_2 \to (t_1 = t_3 \to t_2 = t_3)$,
A_5: $t_1 = t_2 \to t_1' = t_2'$,
A_6: $t + 0 = t$,
A_7: $t_1 + t_2' = (t_1 + t_2)'$,
A_8: $t \cdot 0 = 0$,
A_9: $t_1 \cdot t_2' = t_1 \cdot t_2 + t_1$,

где P — любая формула, а t, t_1, t_2 — любые термы теории A.

ЗАМЕЧАНИЕ

Впервые данная система аксиом для арифметики была предложена Пеано[1], а A_1 — это схема аксиомы математической индукции.

4.4.8. Неаксиоматизируемые теории

В этом подразделе на примере некоторых понятий теории (абелевых) групп неформально устанавливаются границы применимости исчисления предикатов первого порядка.

Теория групп $\mathcal{G}$ — это прикладное исчисление предикатов *с равенством*, в котором имеются:

1. Предметная константа 0.
2. Двухместный функциональный символ $+$.
3. Собственные аксиомы (схемы аксиом):

G_1: $\forall x.y,z\ (x + (y + z) = (x + y) + z)$,
G_2: $\forall x\ (0 + x = x)$,
G_3: $\forall x\ (\exists y\ (x + y = 0))$.

[1] Джузеппе Пеано (1858–1932).

ЗАМЕЧАНИЕ

Выражение «теория первого порядка с равенством» означает, что подразумевается наличие предиката $=$, аксиом E_1 и E_2 и всех их следствий.

Группа называется *абелевой*, если имеет место собственная аксиома

$$G_4\colon \quad \forall x, y\ (x + y = y + x)\,.$$

Говорят, что в абелевой группе все элементы имеют *конечный порядок* n, если выполнена собственная аксиома

$$G_5\colon \quad \forall x\ (\exists k \leqslant n\ (kx = 0))\,,$$

где kx — это сокращение для $\underbrace{x + x + \cdots + x}_{k\text{раз}}$.

Эта формула не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные предикаты и переменные. Однако для любого конкретного конечного n собственная аксиома может быть записана в виде допустимой формулы теории $\mathcal{G}$:

$$G'_5\colon \quad \forall x\ ((x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0))\,.$$

Абелева группа называется *делимой*, если выполнена собственная аксиома

$$G_6\colon \quad \forall n \geqslant 1\ (\forall x\ (\exists y\ (ny = x)))\,.$$

Эта формула не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные константы и переменные. Однако собственная аксиома может быть записана в виде бесконечного множества допустимых формул теории $\mathcal{G}$:

$$G'_6\colon \quad \forall x\ (\exists y\ (2y = x))\,,$$
$$\forall x\ (\exists y\ (3y = x))\,,$$

Но можно показать, что любое *конечное* множество формул, истинное во всех делимых абелевых группах, истинно и в некоторой неделимой абелевой группе, то есть теория делимых абелевых групп не является *конечно* аксиоматизируемой.

Абелева группа называется *периодической*, если выполнена собственная аксиома

$$G_7\colon \quad \forall x\ (\exists n \geqslant 1\ (nx = 0))\,.$$

Эта формула также не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные константы и переменные. Если попытаться преобразовать формулу G_7 по образцу формулы G_5, то получится бесконечная «формула»

$$G'_7\colon \quad \forall x\ ((x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0 \vee \ldots))\,,$$

которая не является допустимой формулой исчисления предикатов и тем более теории $\mathcal{G}$. Таким образом, теория периодических абелевых групп не является *аксиоматизируемой* (если не включать в теорию групп и всю формальную арифметику).

ОТСТУПЛЕНИЕ

Наличие неаксиоматизируемых и конечно неаксиоматизируемых формальных теорий не означает практической неприменимости аксиоматического метода на основе использования языка исчисления предикатов первого порядка. Это означает, что аксиоматический метод не применим «в чистом виде». На практике формальные теории, описывающие содержательные объекты, задаются с помощью собственных аксиом, которые наряду с собственными предикатами и функциональными символами содержат «внелогические» предикаты и функциональные символы, свойства которых аксиомами не описываются, а считаются известными (в данной теории). В рассмотренных примерах подраздела 4.4.8 внелогическими являются натуральные числа и операции над ними. Аналогичное обстоятельство имеет место и в системах логического программирования типа Пролог. Реализация такой системы всегда снабжается обширной библиотекой внелогических (или *встроенных*) предикатов и функций, которые и обеспечивают практическую применимость системы логического программирования.

4.4.9. Теоремы Гёделя о неполноте

Давно известны некоторые важные свойства формальных теорий, в частности, прикладных исчислений предикатов первого порядка, которые существенным образом влияют на практическую применимость формальных теорий (аксиоматического метода). Полное доказательство этих фактов выходит далеко за рамки данного учебника. Однако понимание границ применимости формальных методов является, по нашему мнению, необходимым для использования таких методов. Поэтому ниже приводятся для сведения (без доказательства и в упрощённой формулировке) два важнейших факта, известные как теоремы Гёделя[1] о неполноте.

Аксиоматический метод обладает множеством достоинств и с успехом применяется на практике для формализации самых разнообразных предметных областей в математике, физике и других науках. Однако этому методу присуще следующее принципиальное ограничение.

ТЕОРЕМА (Первая теорема Гёделя о неполноте) *Во всякой достаточно богатой теории первого порядка (в частности, во всякой теории, включающей формальную арифметику), существует такая истинная формула F, что ни F, ни $\neg F$ не являются выводимыми в этой теории.*

Утверждения о теории первого порядка также могут быть сформулированы в виде формул теории первого порядка. В частности, утверждения о свойствах формальной арифметики (например, утверждение о непротиворечивости формальной арифметики), могут быть сформулированы как арифметические утверждения.

[1] Курт Гёдель (1906–1978).

ТЕОРЕМА (Вторая теорема Гёделя о неполноте) *Во всякой достаточно богатой теории первого порядка (в частности, во всякой теории, включающей формальную арифметику), формула F, утверждающая непротиворечивость этой теории, не является выводимой в ней.*

ОТСТУПЛЕНИЕ

Вторая теорема Гёделя не утверждает, что арифметика противоречива. (Формальная арифметика из примера 4.4.7 как раз непротиворечива.) Эта теорема утверждает, что непротиворечивость достаточно богатой формальной теории не может быть установлена средствами самой этой теории (см. ещё раз следствие к теореме 4.3.8). При этом вполне может статься, что непротиворечивость одной конкретной теории может быть установлена средствами другой, более мощной формальной теории. Но тогда возникает вопрос о непротиворечивости этой второй теории и т. д.

4.5. Автоматическое доказательство теорем

Автоматическое доказательство теорем — это краеугольный камень логического программирования, искусственного интеллекта и других современных направлений в программировании. Здесь излагаются основы *метода резолюций* — классического (и в то же время до сих пор популярного) метода автоматического доказательства теорем, предложенного Робинсоном[1] в 1965 году.

4.5.1. Постановка задачи

Алгоритм, который проверяет отношение

$$\Gamma \vdash_{\mathcal{T}} S$$

для формулы S, множества формул Γ и теории $\mathcal{T}$, называется алгоритмом *автоматического доказательства теорем*. В общем случае такой алгоритм невозможен, то есть не существует алгоритма, который для любых S, Γ и $\mathcal{T}$ выдавал бы ответ «Да», если $\Gamma \vdash_{\mathcal{T}} S$, и ответ «Нет», если неверно, что $\Gamma \vdash_{\mathcal{T}} S$. Более того, известно, что нельзя построить алгоритм автоматического доказательства теорем даже для большинства конкретных достаточно богатых формальных теорий $\mathcal{T}$. В некоторых случаях удаётся построить алгоритм автоматического доказательства теорем, который применим не ко всем формулам теории (то есть частичный алгоритм, см. 4.2.5).

Для некоторых простых формальных теорий (например, для исчисления высказываний) и некоторых простых классов формул (например, для прикладного исчисления предикатов с одним одноместным предикатом) алгоритмы автоматического доказательства теорем известны.

[1] Джон А. Робинсон (р. 1930).

Пример Поскольку для исчисления высказываний известно, что теоремами являются общезначимые формулы, можно воспользоваться простым методом проверки общезначимости формулы с помощью таблиц истинности. А именно, достаточно вычислить истинностное значение формулы при всех возможных интерпретациях (их конечное число). Если во всех случаях получится значение T, то проверяемая формула — тавтология и, следовательно, является теоремой теории $\mathfrak{L}$. Если же хотя бы в одном случае получится значение F, то проверяемая формула не является тавтологией и, следовательно, не является теоремой теории $\mathfrak{L}$.

ЗАМЕЧАНИЕ

Приведённый выше пример является алгоритмом автоматического доказательства теорем в теории $\mathfrak{L}$, хотя и не является алгоритмом автоматического поиска вывода теорем из аксиом теории $\mathfrak{L}$.

Наиболее известный классический алгоритм автоматического доказательства теорем называется *методом резолюций*. Для любого прикладного исчисления предикатов первого порядка $\mathfrak{T}$, любой формулы S и множества формул Γ теории $\mathfrak{T}$ метод резолюций выдаёт ответ «Да», если $\Gamma \vdash_{\mathfrak{T}} S$, и выдаёт ответ «Нет» или не выдаёт никакого ответа (то есть «зацикливается»), если неверно, что $\Gamma \vdash_{\mathfrak{T}} S$.

4.5.2. Доказательство от противного

В основе метода резолюций лежит идея «доказательства от противного».

ТЕОРЕМА *Если* $\Gamma, \neg S \vdash F$, *где* F *— любое противоречие (тождественно ложная формула), то* $\Gamma \vdash S$.

Доказательство (для случая $\mathfrak{L}$) $\Gamma, \neg S \vdash F \iff \Gamma \,\&\, \neg S \to F$ — тавтология. Но

$$\Gamma \,\&\, \neg S \to F = \neg(\Gamma \,\&\, \neg S) \vee F = \neg(\Gamma \,\&\, \neg S) = \neg\Gamma \vee S = \Gamma \to S.$$

Имеем: $\Gamma \to S$ — тавтология $\iff \Gamma \vdash S$. □

Пустая формула не имеет никакого значения ни в какой интерпретации, в частности, не является истиной ни в какой интерпретации и, по определению, является противоречием. В качестве формулы F при доказательстве от противного по методу резолюций принято использовать пустую формулу. Полезно сравнить использование пустой формулы в методе резолюций с подразделом 3.4.2.

ЗАМЕЧАНИЕ

При изложении метода резолюций пустая формула традиционно обозначается □. Однако значок □ занят в книге для обозначения конца доказательства, поэтому для обозначения пустой формулы в этом разделе мы используем нетрадиционный знак $\boxed{\square}$.

4.5.3. Сведéние к предложениям

Метод резолюций работает с особой стандартной формой формул, которые называются *предложениями*. Предложение — это бескванторная дизъюнкция литералов. Любая формула исчисления предикатов может быть преобразована в множество предложений следующим образом (здесь знак $\mapsto$ используется для обозначения способа преобразования формул):

1. *Элиминация импликации.* Преобразование:
   $A \to B \mapsto \neg A \vee B$.
   После первого этапа формула содержит только $\neg$, $\vee$, $\&$, $\forall$, $\exists$.

2. *Протаскивание отрицаний.* Преобразования:
   $\neg\forall x\ (A) \mapsto \exists x\ (\neg A)$, $\neg\exists x\ (A) \mapsto \forall x\ (\neg A)$, $\neg\neg A \mapsto A$,
   $\neg(A \vee B) \mapsto \neg A\ \&\ \neg B$, $\neg(A\ \&\ B) \mapsto \neg A \vee \neg B$.
   После второго этапа формула содержит отрицания только перед атомами.

3. *Разделение связанных переменных.* Преобразование:
   $Q_1 x\ A(\ldots Q_2 x\ B(\ldots x \ldots)\ldots) \mapsto Q_1 x\ A(\ldots Q_2 y\ B(\ldots y \ldots)\ldots)$,
   где Q_1 и Q_2 — любые кванторы. После третьего этапа формула не содержит случайно совпадающих связанных переменных.

4. *Приведение к предварённой форме.* Преобразования:
   $Q x\ A \vee B \mapsto Q x\ (A \vee B)$, $Q x\ A\ \&\ B \mapsto Q x\ (A\ \&\ B)$,
   где Q — любой квантор. После четвёртого этапа формула находится в предварённой форме.

5. *Элиминация кванторов существования (сколемизация).* Преобразования:
   $\exists x_1\ (Q_2 x_2\ \ldots Q_n x_n\ A(x_1, x_2, \ldots, x_n)) \mapsto Q_2 x_2 \ldots Q_n x_n\ A(a, x_2, \ldots, x_n)$,
   $\forall x_1\ (\ldots \forall x_{i-1}\ (\exists x_i\ (Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, x_i, \ldots, x_n)))\ldots) \mapsto$
   $\mapsto \forall x_1\ (\ldots \forall x_{i-1}\ (Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, f(x_1, \ldots, x_{i-1}), \ldots, x_n))\ldots)$,
   где a — новая предметная константа, f — новый функциональный символ, а $Q_1, Q_2, \ldots, Q_n$ — любые кванторы. После пятого этапа формула содержит только кванторы всеобщности.

6. *Элиминация кванторов всеобщности.* Преобразование:
   $\forall x\ (A(x)) \mapsto A(x)$.
   После шестого этапа формула не содержит кванторов.

7. *Приведение к конъюнктивной нормальной форме.* Преобразования:
   $A \vee (B\ \&\ C) \mapsto (A \vee B)\ \&\ (A \vee C)$, $(A\ \&\ B) \vee C \mapsto (A \vee C)\ \&\ (B \vee C)$.
   После седьмого этапа формула находится в конъюнктивной нормальной форме.

8. *Элиминация конъюнкции.* Преобразование:
   $A\ \&\ B \mapsto A, B$.
   После восьмого этапа формула распадается на множество предложений.

Не все преобразования на этапах 1–8 являются логически эквивалентными.

ТЕОРЕМА *Если* Γ *— множество предложений, полученных из формулы* S*, то* S *является противоречием тогда и только тогда, когда множество* Γ *невыполнимо.*

Доказательство В доказательстве нуждается шаг 5 — сколемизация. Пусть $F := \forall x_1\ (\ldots \forall x_{i-1}\ (\exists x_i Q_{i+1} x_{i+1} \ldots Q_n x_n\ (A(x_1, \ldots, x_n))) \ldots)$. Положим $F' := \forall x_1\ (\ldots \forall x_{i-1}\ (Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, x_{i-1}, f(x_1, \ldots, x_{i-1}), x_{i+1}, \ldots, x_n) \ldots))$. Пусть F — противоречие, а F' — не противоречие. Тогда существуют интерпретация I и набор значений $s = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, такие, что $s^*(F') = \mathrm{T}$. Положим $a_i := f(a_1, \ldots, a_{i-1})$, $s_1 := (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$. Тогда $s_1^*(F) = \mathrm{T}$, и F — выполнимая формула. □

ЗАМЕЧАНИЕ

Невыполнимость множества формул Γ означает, что множество Γ не имеет модели, то есть не существует интерпретации, в которой все формулы Γ имели бы значение T.

4.5.4. Правило резолюции для исчисления высказываний

Метод резолюций основан на использовании специального правила вывода.

Пусть C_1 и C_2 — два предложения в исчислении высказываний и пусть $C_1 = P \vee C_1'$, а $C_2 = \neg P \vee C_2'$, где P — пропозициональная переменная, а C_1', C_2' — любые предложения (в частности, может быть, пустые или состоящие только из одного литерала). Правило вывода

$$\frac{C_1,\ C_2}{C_1' \vee C_2'}\ (R)$$

называется *правилом резолюции*. Предложения C_1 и C_2 называются *резольвируемыми* (или *родительскими*), предложение $C_1' \vee C_2'$ — *резольвентой*, а формулы P и $\neg P$ — *контрарными* литералами.

ЗАМЕЧАНИЕ

Резольвента является предложением, то есть множество предложений замкнуто относительно правила резолюции.

ТЕОРЕМА 1 *Правило резолюции логично, то есть резольвента является логическим следствием резольвируемых предложений.*

Доказательство Пусть $I(C_1) = \mathrm{T}$ и $I(C_2) = \mathrm{T}$. Тогда если $I(P) = \mathrm{T}$, то $C_2' \neq \varnothing$ и $I(C_2') = \mathrm{T}$, а значит, $I(C_1' \vee C_2') = \mathrm{T}$. Если же $I(P) = \mathrm{F}$, то $C_1' \neq \varnothing$ и $I(C_1') = \mathrm{T}$, а значит, $I(C_1' \vee C_2') = \mathrm{T}$. □

Правило резолюции — это очень мощное правило вывода. Многие ранее рассмотренные правила являются его частными случаями:

$$\frac{A, A \to B}{B}\ \text{Modus ponens} \qquad \frac{A, \neg A \vee B}{B}\ R,$$

$$\frac{A \to B, B \to C}{A \to C}\ \text{Транзитивность} \qquad \frac{\neg A \vee B, \neg B \vee C}{\neg A \vee C}\ R,$$

$$\frac{A \vee B, A \to B}{B}\ \text{Слияние} \qquad \frac{A \vee B, \neg A \vee B}{B}\ R.$$

ТЕОРЕМА 2 *Правило резолюции полно, то есть всякая тавтология может быть выведена с использованием только правила резолюции.*

Доказательство Без доказательства. □

4.5.5. Правило резолюции для исчисления предикатов

Для применения правила резолюции нужны контрарные литералы в резольвируемых предложениях. В случае исчисления высказываний контрарные литералы определяются очень просто: это пропозициональная переменная и её отрицание. Для исчисления предикатов определение чуть сложнее.

Пусть C_1 и C_2 — два предложения в исчислении предикатов. Правило вывода

$$\frac{C_1,\ C_2}{(C_1' \vee C_2')\sigma}\ (R)$$

называется правилом резолюции в исчислении предикатов, если в предложениях C_1 и C_2 существуют унифицируемые контрарные литералы P_1 и P_2, то есть $C_1 = P_1 \vee C_1'$ и $C_2 = \neg P_2 \vee C_2'$, причём атомарные формулы P_1 и P_2 унифицируются наиболее общим унификатором σ. В этом случае резольвентой предложений C_1 и C_2 является предложение $(C_1' \vee C_2')\sigma$, полученное из предложения $C_1' \vee C_2'$ применением унификатора σ.

4.5.6. Опровержение методом резолюций

Опровержение методом резолюций — это алгоритм автоматического доказательства теорем в прикладном исчислении предикатов, который сводится к следующему. Пусть нужно установить выводимость

$$S \vdash G.$$

Каждая формула множества S и формула $\neg G$ (отрицание целевой теоремы) независимо преобразуются в множества предложений. В полученном совокупном множестве предложений C отыскиваются резольвируемые предложения, к ним применяется правило резолюции и резольвента добавляется в множество предложений до тех пор, пока не будет получено пустое предложение. При этом возможны три случая:

1. Среди текущего множества предложений нет резольвируемых или же все резольвенты уже присутствуют в текущем множестве предложений. Это означает, что теорема опровергнута, то есть формула G не выводима из множества формул S.
2. В результате очередного применения правила резолюции получено пустое предложение. Это означает, что теорема доказана, то есть $S \vdash G$.
3. Процесс не заканчивается, то есть множество предложений пополняется всё новыми резольвентами, среди которых нет пустых. Это ничего не означает.

ЗАМЕЧАНИЕ

Таким образом, исчисление предикатов является *полуразрешимой* теорией, а метод резолюций является *частичным* алгоритмом автоматического доказательства теорем.

Пример Докажем методом резолюций теорему $\vdash_{\mathcal{L}} (((A \to B) \to A) \to A)$. Сначала нужно преобразовать в предложения отрицание целевой формулы

$$\neg(((A \to B) \to A) \to A).$$

1. $\neg(\neg(\neg(\neg A \vee B) \vee A) \vee A)$.
2. $(((A \,\&\, \neg B) \vee A) \,\&\, \neg A)$.

3–6. Формула без изменений.

7. $(A \vee A) \,\&\, (\neg B \vee A) \,\&\, \neg A$.
8. $A \vee A, \neg B \vee A, \neg A$.

После этого проводится резольвирование имеющихся предложений 1–3.

1. $A \vee A$.
2. $\neg B \vee A$.
3. $\neg A$.
4. A из 1 и 3 по правилу резолюции.
5. $\boxed{\square}$ из 3 и 4 по правилу резолюции.

Таким образом, теорема доказана.

4.5.7. Алгоритм метода резолюций

Алгоритм поиска опровержения методом резолюций проверяет выводимость формулы G из множества формул S.

Алгоритм 4.2 Метод резолюций

Вход: множество предложений C, полученных из множества формул S и формулы $\neg G$.

Выход: **True** — если G выводимо из S, **false** — в противном случае.

$M := C$ { M — текущее множество предложений }

while $\boxed{\square} \notin M$ **do**
  Choose$(M, c_1, c_2, p_1, p_2, \sigma)$ { выбор родительских предложений }
  if $c_1, c_2 = \varnothing$ **then**
    return false { нечего резольвировать — теорема опровергнута }
  end if
  $c := R(c_1, c_2, p_1, p_2, \sigma)$ { вычисление резольвенты }
  $M := M + c$ { пополнение текущего множества }
end while
return true { теорема доказана }

Обоснование Если алгоритм заканчивает свою работу, то правильность результата очевидным образом следует из теорем предшествующих разделов. Вообще говоря, завершаемость этого алгоритма не гарантирована. □

В этом алгоритме использованы две вспомогательные функции. Функция R вычисляет резольвенту двух предложений c_1 и c_2, содержащих унифицируемые контрарные литералы p_1 и p_2 соответственно; при этом σ — наиболее общий унификатор (см. 4.3.3). Результатом работы функции является резольвента.

Процедура Choose выбирает в текущем множестве предложений M два резольвируемых предложения, то есть два предложения, которые содержат унифицируемые контрарные литералы. Если таковые есть, то процедура их возвращает, в противном случае возвращается пустое множество. Конкретные реализации процедуры Choose называются *стратегиями* метода резолюций.

ЗАМЕЧАНИЕ

В настоящее время предложено множество различных стратегий метода резолюций. Среди них различаются *полные* и *неполные* стратегии. Полные стратегии — это такие, которые гарантируют нахождение доказательства теоремы, если оно вообще существует. Неполные стратегии могут в некоторых случаях не находить доказательства, зато они работают быстрее. Следует иметь в виду, что автоматическое доказательство теорем методом резолюций имеет по существу переборный характер, и этот перебор столь велик, что может быть практически неосуществим за приемлемое время.

ОТСТУПЛЕНИЕ

При автоматическом доказательстве теорем методом резолюций львиная доля вычислений приходится на поиск унифицируемых предложений. Таким образом, эффективная реализация алгоритма унификации критически важна для практической применимости метода резолюций.

Комментарии

В этой главе изложены элементарные сведения из математической логики, причём технически сложные доказательства опущены. Более подробное изложение можно найти в многочисленных классических учебниках, таких как [30]. Классическое описание автоматического доказательства теорем методом резолюций име-

ется в [29]. Эта книга содержит описание различных стратегий и усовершенствований метода резолюций, а также многочисленные примеры применений. Алгоритмы 4.1 и 4.2 являются упрощенными модификациями алгоритмов, описанных в [29].

Упражнения

4.1. Пусть $x \equiv y \stackrel{\text{Def}}{=} (x \to y)\ \&\ (y \to x)$. Доказать, что формула $\mathcal{L}$, содержащая только связку $\equiv$, является тавтологией тогда и только тогда, когда каждая переменная входит в неё чётное число раз.

4.2. Описать процедуру, которая перечисляет множество термов для заданных конечных множеств переменных, констант и функциональных символов.

4.3. Пусть некоторая формула A теории $\mathcal{L}$ — не тавтология. Обозначим через $\tilde{A}$ множество всех частных случаев формулы A. Определим теорию $\mathcal{L}^+$, добавив к исчислению высказываний все частные случаи формулы A в качестве аксиом: $\mathcal{L}^+ \stackrel{\text{Def}}{=} \mathcal{L} \cup \tilde{A}$. Доказать, что теория $\mathcal{L}^+$ противоречива, то есть существует такая формула B, что $\vdash_{\mathcal{L}^+} B$ и $\vdash_{\mathcal{L}^+} \neg B$.

4.4. Доказать, что формула $(\forall x\ (A)(x) \to \forall x\ (B(x))) \to (\forall x\ (A(x) \to B(x)))$ общезначима.

4.5. Доказать методом резолюций: $\vdash_{\mathcal{L}} A \to (B \to A\ \&\ B)$.

Глава 5 Комбинаторика

Предмету комбинаторики не так просто дать краткое исчерпывающее определение. В некотором смысле слово «комбинаторика» можно понимать как синоним термина «дискретная математика», то есть исследование дискретных конечных математических структур. На школьном уровне с термином «комбинаторика» связывают просто набор известных формул, служащих для вычисления так называемых *комбинаторных чисел*, о которых идёт речь в первых разделах этой главы. Может показаться, что эти формулы полезны только для решения олимпиадных задач и не имеют отношения к практическому программированию. На самом деле это далеко не так. Вычисления на дискретных конечных математических структурах, которые часто называют *комбинаторными вычислениями*, требуют комбинаторного анализа для установления свойств и выявления оценки применимости используемых алгоритмов. Рассмотрим элементарный жизненный пример.

Пример Пусть некоторое агентство недвижимости располагает базой данных из n записей, причём каждая запись содержит одно предложение (что́ имеется) и один запрос (что́ требуется) относительно объектов недвижимости. Требуется найти все такие пары записей, в которых предложение первой записи совпадает с запросом второй и одновременно предложение второй записи совпадает с запросом первой. (На бытовом языке это называется подбором вариантов обмена.) Допустим, что используемая СУБД позволяет проверить вариант за одну миллисекунду. Нетрудно сообразить, что при «лобовом» алгоритме поиска вариантов (каждая запись сравнивается с каждой) потребуется $n(n-1)/2$ сравнений. Если $n = 100$, то все в порядке — ответ будет получен за 4,95 секунды. Но если $n = 100\,000$ (более реальный случай), то ответ будет получен за 4 999 950 секунд, что составляет без малого 1389 часов и вряд ли может считаться приемлемым. Обратите внимание на то, что мы оценили только трудоёмкость подбора *прямых* вариантов, а существуют ещё варианты, когда число участников сделки больше двух...

Этот пример показывает, что практические задачи и алгоритмы требуют предварительного анализа и количественной оценки. Задачи обычно оцениваются с точки зрения *размера*, то есть общего количества различных вариантов, среди которых нужно найти решение, а алгоритмы оцениваются с точки зрения *сложности*. При этом различают *сложность по времени* (или *временну́ю сложность*), то есть количество необходимых шагов алгоритма, и *сложность по памяти* (или *емкостну́ю сложность*), то есть объём памяти, необходимый для работы алгоритма.

Во всех случаях основным инструментом такого анализа оказываются формулы и методы, рассматриваемые в этой главе.

5.1. Комбинаторные задачи

Во многих практических случаях возникает необходимость подсчитать количество возможных комбинаций объектов, удовлетворяющих определённым условиям. Такие задачи называются *комбинаторными*. Разнообразие комбинаторных задач не поддаётся исчерпывающему описанию, но среди них есть целый ряд особенно часто встречающихся, для которых известны способы подсчёта.

5.1.1. Комбинаторные конфигурации

Для формулировки и решения комбинаторных задач используются различные модели *комбинаторных конфигураций*. Рассмотрим следующие две наиболее популярные:

1. Дано n предметов. Их нужно разместить по m ящикам так, чтобы выполнялись заданные ограничения. Сколькими способами это можно сделать?
2. Рассмотрим множество функций

$$F\colon X \to Y, \quad \text{где } |X| = n,\ |Y| = m.$$

Не ограничивая общности, можно считать, что

$$X = \{1,\ldots,n\},\ Y = \{1,\ldots,m\},\ F = \langle F(1),\ldots,F(n)\rangle,\ 1 \leqslant F(i) \leqslant m.$$

Сколько существует функций F, удовлетворяющих заданным ограничениям?

ЗАМЕЧАНИЕ

Большей частью соответствие конфигураций, описанных на «языке ящиков» и на «языке функций», очевидно, поэтому доказательство правильности способа подсчёта (вывод формулы) можно провести на любом языке. Если сведе́ние одной модели к другой не очевидно, то оно включается в доказательство.

5.1.2. Размещения

Число всех функций (при отсутствии ограничений), или число всех возможных способов разместить n предметов по m ящикам, называется *числом размещений* и обозначается $U(m,n)$.

ТЕОРЕМА $U(m,n) = m^n$.

Доказательство Поскольку ограничений нет, предметы размещаются независимо друг от друга и каждый из n предметов можно разместить m способами. □

ЗАМЕЧАНИЕ

В комбинаторных задачах часто используются два правила, которые называются *правилом суммы* и *правилом произведения*. Неформально эти правила можно сформулировать следующим образом. Пусть существуют некоторые возможности построения комбинаторной конфигурации. Если эти возможности взаимно исключают друг друга, то их количества следует складывать, а если возможности независимы, то их количества следует перемножать.

Пример При игре в кости бросаются две кости и выпавшие на верхних гранях очки складываются. Какова вероятность выбросить 12 очков? Каждый возможный исход соответствует функции $F\colon \{1,2\} \to \{1,2,3,4,5,6\}$ (аргумент — номер кости, результат — очки на верхней грани). Таким образом, всего возможно $U(6,2) = 6^2 = 36$ различных исходов. Из них только один $(6+6)$ даёт двенадцать очков. Вероятность $1/36$.

5.1.3. Размещения без повторений

Число инъективных функций, или число способов разместить n предметов по m ящикам, не более чем по одному в ящик, называется *числом размещений без повторений* и обозначается $A(m,n)$, или $[m]_n$, или $(m)_n$.

ТЕОРЕМА $A(m,n) = \dfrac{m!}{(m-n)!}$.

Доказательство Ящик для первого предмета можно выбрать m способами, для второго — $m-1$ способами и т. д. Таким образом,

$$A(m,n) = m \cdot (m-1) \cdot \ldots \cdot (m-n+1) = \frac{m!}{(m-n)!}. \qquad \square$$

По определению считают, что $A(m,n) \stackrel{\text{Def}}{=} 0$ при $n > m$ и $A(m,0) \stackrel{\text{Def}}{=} 1$.

ОТСТУПЛЕНИЕ

Простые формулы, выведенные для числа размещений без повторений, дают повод поговорить об элементарных, но весьма важных вещах. Рассмотрим две формулы:

$$A(m,n) = m \cdot (m-1) \cdot \ldots \cdot (m-n+1) \quad \text{и} \quad A(m,n) = \frac{m!}{(m-n)!}.$$

Формула слева выглядит сложной и незавершённой, формула справа — изящной и «математичной». Но формула — это частный случай алгоритма. При практическом вычислении левая формула оказывается намного предпочтительнее правой. Во-первых, для вычисления по левой формуле потребуется $m-1$ умножение, а по правой — $2m-n-2$ умножений и одно деление. Поскольку $n < m$, левая формула вычисляется существенно быстрее. Во-

вторых, число $A(m,n)$ растёт довольно быстро и при больших m и n может не поместиться в разрядную сетку. Левая формула работает правильно, если результат помещается в разрядную сетку. При вычислении же по правой формуле переполнение может наступить «раньше времени» (то есть промежуточные результаты не помещаются в разрядную сетку, в то время как окончательный результат мог бы поместиться), поскольку факториал — очень быстро растущая функция.

Пример В некоторых видах спортивных соревнований исходом является определение участников, занявших первое, второе и третье места. Сколько возможно различных исходов, если в соревновании участвуют n участников? Каждый возможный исход соответствует функции $F\colon \{1,2,3\} \to \{1..n\}$ (аргумент — номер призового места, результат — номер участника). Таким образом, всего возможно $A(n,3) = n(n-1)(n-2)$ различных исходов.

5.1.4. Перестановки

Если $|X| = |Y| = n$, то существуют взаимно-однозначные функции $f\colon X \to Y$. Число взаимно-однозначных функций, или *число перестановок* n предметов, обозначается $P(n)$.

ТЕОРЕМА $P(n) = n!$.

Доказательство $P(n) = A(n,n) = n\cdot(n-1)\cdot\ldots\cdot(n-n+1) = n\cdot(n-1)\cdot\ldots\cdot 1 = n!$. □

Пример Последовательность $\mathcal{E} = (E_1,\ldots,E_m)$ непустых подмножеств множества E ($\mathcal{E} \subset 2^E$, $E_i \subset E$, $E_i \neq \varnothing$) называется *цепочкой* в E, если

$$\forall i \in 1..(m-1)\ \ (E_i \subset E_{i+1}\ \&\ E_i \neq E_{i+1})\,.$$

Цепочка $\mathcal{E}$ называется *полной* цепочкой в E, если $|\mathcal{E}| = |E|$. Сколько существует полных цепочек? Очевидно, что в полной цепочке каждое следующее подмножество E_{i+1} получено из предыдущего подмножества E_i добавлением ровно одного элемента из E и таким образом, $|E_1| = 1$, $|E_2| = 2$, ..., $|E_m| = |E| = m$. Следовательно, полная цепочка определяется порядком, в котором элементы E добавляются для образования очередного элемента полной цепочки. Отсюда количество полных цепочек — это количество перестановок элементов множества E, равное $m!$.

5.1.5. Сочетания

Число строго монотонно возрастающих функций, или число размещений n неразличимых предметов по m ящикам не более чем по одному в ящик, то есть число способов выбрать из m ящиков n ящиков с предметами, называется *числом сочетаний* и обозначается $C(m,n)$, или C_m^n, или $\binom{n}{m}$.

ТЕОРЕМА $C(m,n) = \dfrac{m!}{n!(m-n)!}$.

Доказательство

[Обоснование формулы] Сочетание является размещением без повторений неразличимых предметов. Следовательно, число сочетаний определяется тем, какие ящики заняты предметами, поскольку перестановка предметов при тех же занятых ящиках считается одним сочетанием. Таким образом,

$$C(m,n) = \frac{A(m,n)}{P(n)} = \frac{m!}{n!(m-n)!}.$$

[Сведéние моделей] Число сочетаний является числом строго монотонно возрастающих функций, потому что любая строго монотонно возрастающая функция $F\colon 1..n \to 1..m$ определяется набором своих значений, причём $1 \leqslant F(1) < \ldots < F(n) \leqslant m$. Другими словами, каждая строго монотонно возрастающая функция определяется выбором n чисел из диапазона $1..m$. Таким образом, число строго монотонно возрастающих функций равно числу n-элементных подмножеств m-элементного множества, которое, в свою очередь, равно числу способов выбрать n ящиков с предметами из m ящиков. □

По определению $C(m,n) \stackrel{\text{Def}}{=} 0$ при $n > m$.

Пример В начале игры в домино каждому играющему выдаётся 7 костей из имеющихся 28 различных костей. Сколько существует различных комбинаций костей, которые игрок может получить в начале игры? Очевидно, что искомое число равно числу 7-элементных подмножеств 28-элементного множества. Имеем:

$$C(28,7) = \frac{28!}{7!(28-7)!} = \frac{28 \cdot 27 \cdot 26 \cdot 25 \cdot 24 \cdot 23 \cdot 22}{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 1\,184\,040.$$

5.1.6. Сочетания с повторениями

Число монотонно возрастающих функций, или число размещений n неразличимых предметов по m ящикам, называется *числом сочетаний с повторениями* и обозначается $V(m,n)$.

ТЕОРЕМА $V(m,n) = C(n+m-1,n)$.

Доказательство Монотонно возрастающей функции $f\colon 1..n \to 1..m$ однозначно соответствует строго монотонно возрастающая функция $f'\colon 1..n \to 1..(n+m-1)$. Это соответствие устанавливается следующей формулой: $f'(k) = f(k)+k-1$. □

Пример Сколькими способами можно рассадить n вновь прибывших гостей среди m гостей, уже сидящих за круглым столом? Очевидно, что между m сидящими за круглым столом гостями имеется m промежутков, в которые можно рассаживать вновь прибывших. Таким образом, число способов равно

$$V(m,n) = C(m+n-1,n) = \frac{(m+n-1)!}{n!(m-1)!}.$$

5.2. Перестановки

Для вычисления количества перестановок в 5.1.4 установлена очень простая формула: $P(n) = n!$. Применяя эту формулу при решении практических задач, не следует забывать, что факториал — это *очень* быстро растущая функция, в частности, факториал растёт быстрее экспоненты. Действительно, используя известную из математического анализа *формулу Стирлинга*

$$n^n\sqrt{2\pi n}\,e^{-n} < n! < n^n\sqrt{2\pi n}\,e^{-n+\frac{1}{12n}},$$

нетрудно показать, что

$$\lim_{n\to+\infty}\frac{n!}{2^n} = +\infty.$$

5.2.1. Графическое представление перестановок

В 2.2.5 рассматриваются взаимно-однозначные функции (перестановки), которые удобно задавать таблицами подстановки.

В таблице подстановки нижняя строка (значения функции) является перестановкой элементов верхней строки (значения аргумента). Если принять соглашение, что элементы верхней строки (аргументы) всегда располагаются в определённом порядке (например, по возрастанию), то верхнюю строку можно не указывать — подстановка определяется одной нижней строкой. Таким образом, подстановки (таблицы) взаимно-однозначно соответствуют перестановкам (функциям). Напомним (см. 2.2.5), что множество перестановок образует группу относительно суперпозиции.

Перестановку f (и соответствующую ей подстановку) элементов $1,\ldots,n$ будем обозначать $\langle a_1,\ldots,a_n\rangle$, где a_i — все различные числа из диапазона $1..n$.

Иногда перестановки удобно представлять в графической форме, проводя стрелки от каждого элемента x к элементу $f(x)$.

Пример Графическое представление перестановки (см. 2.2.5)

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 4 & 5 \end{vmatrix}$$

показано на рис. 5.1.

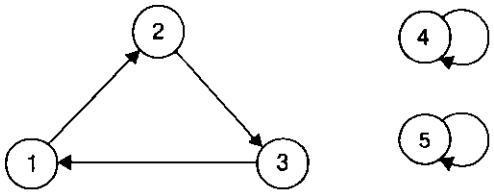

Рис. 5.1. Графическое представление перестановки

Если задана перестановка f, то *циклом* называется последовательность элементов $x_0, \ldots, x_k$, такая, что

$$f(x_i) = \begin{cases} x_{i+1}, & 0 \leqslant i < k, \\ x_0, & i = k. \end{cases}$$

Цикл длины 2 называется *транспозицией*.

ЗАМЕЧАНИЕ

Из графического представления перестановки наглядно видно происхождение термина «цикл».

5.2.2. Инверсии

Если в перестановке $f = \langle a_1, \ldots, a_n \rangle$ для элементов a_i и a_j имеет место неравенство $a_i > a_j$ при $i < j$, то пара (a_i, a_j) называется *инверсией*. Обозначим $I(f)$ — *число инверсий* в перестановке f.

ТЕОРЕМА *Произвольную перестановку f можно представить в виде суперпозиции $I(f)$ транспозиций соседних элементов.*

Доказательство Пусть $f = \langle a_1, \ldots, 1, \ldots, a_n \rangle$. Переставим 1 на первое место, меняя её местами с соседними слева элементами. Обозначим последовательность этих транспозиций через t_1. При этом все инверсии (и только они), в которых участвовала 1, пропадут. Затем переставим 2 на второе место и т. д. Таким образом, $f \circ t_1 \circ \ldots \circ t_n = e$ и по свойству группы $f = t_n^{-1} \circ \ldots \circ t_1^{-1}$, причём $|t_1| + |t_2| + \ldots + |t_n| = I(f)$. □

СЛЕДСТВИЕ *Всякая сортировка может быть выполнена перестановкой соседних элементов.*

ОТСТУПЛЕНИЕ

Доказанная теорема утверждает, что произвольную перестановку можно представить в виде композиции определённого числа транспозиций, но не утверждает, что такое представление является эффективным. Метод *сортировки*, основанный на предшествующей теореме, известен как *метод пузырька*. Заметим, что при перемещении элемента на своё

место транспозициями соседних элементов все элементы остаются на своих местах, кроме двух соседних элементов, которые, возможно, меняются местами. Таким образом, метод пузырька может быть выражен в форме алгоритма 5.1. Этот алгоритм прост, но является далеко не самым эффективным алгоритмом сортировки.

Алгоритм 5.1 Сортировка методом пузырька

Вход: массив A : **array** $[1..n]$ **of** B, где значения элементов массива расположены в произвольном порядке и для значений типа B задано отношение $<$.
Выход: массив A : **array** $[1..n]$ **of** B, в котором значения расположены в порядке возрастания.

```
for i from 2 to n do
  for j from n downto i do
    if A[j] < A[j − 1] then
      A[j] ↔ A[j − 1] { транспозиция соседних элементов }
    end if
  end for
end for
```

5.2.3. Генерация перестановок

На множестве перестановок естественным образом можно определить порядок на основе упорядоченности элементов. А именно, говорят, что перестановка $\langle a_1, \ldots, a_n \rangle$ *лексикографически* предшествует перестановке $\langle b_1, \ldots, b_n \rangle$, если $\exists k \leqslant n \ (a_k < b_k \ \& \ \forall i < k \ (a_i = b_i))$ (см. также упражнение 1.8). Аналогично, говорят, что перестановка $\langle a_1, \ldots, a_n \rangle$ *антилексикографически* предшествует перестановке $\langle b_1, \ldots, b_n \rangle$, если $\exists k \leqslant n \ (a_k > b_k \ \& \ \forall i > k \ (a_i = b_i))$.

Следующий алгоритм генерирует все перестановки элементов $1, \ldots, n$ в антилексикографическом порядке. Массив P : **array** $[1..n]$ **of** $1..n$ является глобальным и предназначен для хранения перестановок.

Алгоритм 5.2 Генерация перестановок в антилексикографическом порядке

Вход: n — количество элементов
Выход: последовательность перестановок элементов $1, \ldots, n$ в антилексикографическом порядке.

```
for i from 1 to n do
  P[i] := i { инициализация }
end for
Antilex(n) { вызов рекурсивной процедуры Antilex }
```

Основная работа по генерации перестановок выполняется рекурсивной процедурой Antilex.

Вход: m — параметр процедуры — количество первых элементов массива P, для которых генерируются перестановки.
Выход: последовательность перестановок $1, \ldots, m$ в антилексикографическом порядке.

```
if m = 1 then
  yield P { очередная перестановка }
else
  for i from 1 to m do
    Antilex(m − 1) { рекурсивный вызов }
    if i < m then
      P[i] ↔ P[m] { следующий элемент }
      Reverse(m − 1) { изменение порядка элементов }
    end if
  end for
end if
```

Вспомогательная процедура Reverse переставляет элементы заданного отрезка массива P в обратном порядке.

Вход: k — номер элемента, задающий отрезок массива P, подлежащий перестановке в обратном порядке.
Выход: первые k элементов массива P переставлены в обратном порядке

```
j := 1 { нижняя граница обращаемого диапазона }
while j < k do
  P[j] ↔ P[k] { меняем местами элементы }
  j := j + 1 { увеличиваем нижнюю границу }
  k := k − 1 { уменьшаем верхнюю границу }
end while
```

Обоснование Заметим, что искомую последовательность перестановок n элементов можно получить из последовательности перестановок $n-1$ элементов следующим образом. Нужно выписать n блоков по $(n-1)!$ перестановок в каждом, соответствующих последовательности перестановок $n-1$ элементов в антилексикографическом порядке. Затем ко всем перестановкам в первом блоке нужно приписать справа n, во втором — $n-1$ и т. д. в убывающем порядке. Затем в каждом из блоков (кроме первого), к перестановкам которого справа приписан элемент i, нужно в перестановках блока заменить все вхождения элемента i на элемент n. В полученной последовательности все перестановки различны, и их $n(n-1)! = n!$, то есть перечислены все перестановки. При этом антилексикографический порядок соблюден для последовательностей внутри одного блока, потому что этот порядок был соблюден в исходной последовательности, а для последовательностей на границах двух блоков — потому что происходит уменьшение самого правого элемента. Обратимся к процедуре Antilex — легко видеть, что в ней реализовано указанное построение. В основном цикле сначала строится очередной блок — последовательность перестановок первых $m-1$ элементов массива P (при этом элементы $P[m], \ldots, P[n]$ остаются неизменными). Затем элемент $P[m]$ меняется местами с очередным элементом $P[i]$. Вызов вспомогательной процедуры Reverse необходим, поскольку последняя перестановка в блоке является обращением первой, а для генерации следующего блока на очередном шаге цикла нужно восстановить исходный порядок. □

Пример Последовательность перестановок в антилексикографическом порядке для $n = 3$: $(1,2,3)$, $(2,1,3)$, $(1,3,2)$, $(3,1,2)$, $(2,3,1)$, $(3,2,1)$.

5.2.4. Двойные факториалы

Содержание этого подраздела не имеет отношения к основной теме раздела — перестановкам и включено в раздел по сходству обозначений и с целью привести две формулы, которые иногда используются в практических комбинаторных расчётах и требуются в последующих разделах.

Двойным факториалом натурального числа n (обозначается $n!!$) называется произведение числа n и всех меньших натуральных чисел той же чётности.

Пример $10!! = 10 \cdot 8 \cdot 6 \cdot 4 \cdot 2 = 3840$.

ТЕОРЕМА 1. $(2k)!! = 2^k \cdot k!$.

2. $(2k+1)!! = \dfrac{(2k+1)!}{2^k \cdot k!}$.

Доказательство

[1] $(2k)!! = 2 \cdot 4 \cdot \cdots \cdot (2k-2) \cdot 2k = (2 \cdot 1) \cdot (2 \cdot 2) \cdot \cdots \cdot (2 \cdot (k-1)) \cdot (2 \cdot k) =$
$= \underbrace{(2 \cdot 2 \cdot \cdots \cdot 2 \cdot 2)}_{k \text{ раз}} \cdot (1 \cdot 2 \cdot \cdots \cdot (k-1) \cdot k) = 2^k \cdot k!$.

[2] Достаточно заметить, что $(2k+1)! = (2k+1)!!(2k)!!$. □

5.3. Биномиальные коэффициенты

Число сочетаний $C(m,n)$ — это число различных n-элементных подмножеств m-элементного множества (см. 5.1.5). Числа $C(m,n)$ обладают целым рядом свойств, рассматриваемых в этом разделе, которые оказываются очень полезными при решении различных комбинаторных задач.

5.3.1. Элементарные тождества

Основная формула для числа сочетаний

$$C(m,n) = \frac{m!}{n!(m-n)!}$$

позволяет получить следующие простые тождества.

ТЕОРЕМА 1. $C(m,n) = C(m,m-n)$.

2. $C(m,n) = C(m-1,n) + C(m-1,n-1)$.

3. $C(n,i)C(i,m) = C(n,m)C(n-m,i-m)$.

Доказательство

[1] $$C(m,m-n)=\frac{m!}{(m-n)!\,(m-(m-n))!}=\frac{m!}{(m-n)!\,n!}=C(m,n).$$

[2] $$\begin{aligned}C(m-1,n)+C(m-1,n-1)&=\\&=\frac{(m-1)!}{n!\,(m-n-1)!}+\frac{(m-1)!}{(n-1)!\,(m-1-(n-1))!}=\\&=\frac{(m-1)!}{n(n-1)!\,(m-n-1)!}+\frac{(m-1)!}{(n-1)!\,(m-n)(m-n-1)!}=\\&=\frac{(m-n)(m-1)!+n(m-1)!}{n(n-1)!\,(m-n)(m-n-1)!}=\\&=\frac{(m-n+n)(m-1)!}{n!\,(m-n)!}=\frac{m!}{n!\,(m-n)!}=C(m,n).\end{aligned}$$

[3] $$\begin{aligned}C(n,i)C(i,m)&=\frac{n!}{i!(n-i)!}\frac{i!}{m!(i-m)!}=\frac{n!}{m!(i-m)!(n-i)!}=\\&=\frac{n!(n-m)!}{m!(i-m)!(n-i)!(n-m)!}=\\&=\frac{n!}{m!(n-m)!}\frac{(n-m)!}{(i-m)!(n-i)!}=\\&=C(n,m)C(n-m,i-m).\end{aligned}$$

□

5.3.2. Бином Ньютона

Числа сочетаний $C(m,n)$ называются также *биномиальными коэффициентами*. Смысл этого названия устанавливается следующей теоремой, известной также как формула *бинома Ньютона*[1].

ТЕОРЕМА $(x+y)^m=\sum_{n=0}^{m}C(m,n)x^n y^{m-n}$.

Доказательство По индукции. База, $m=1$:

$$(x+y)^1=x+y=1x^1y^0+1x^0y^1=C(1,0)x^1y^0+C(1,1)x^0y^1=\sum_{n=0}^{1}C(1,n)x^n y^{1-n}.$$

Индукционный переход:

[1] Исаак Ньютон (1643–1727).

$$(x+y)^m = (x+y)(x+y)^{m-1} = (x+y)\sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n-1} =$$
$$= \sum_{n=0}^{m-1} xC(m-1,n)x^n y^{m-n-1} + \sum_{n=0}^{m-1} yC(m-1,n)x^n y^{m-n-1} =$$
$$= \sum_{n=0}^{m-1} C(m-1,n)x^{n+1} y^{m-n-1} + \sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n} =$$
$$= \sum_{n=1}^{m} C(m-1,n-1)x^n y^{m-n} + \sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n} =$$
$$= C(m-1,0)x^0 y^m + \sum_{n=1}^{m-1} (C(m-1,n-1)+C(m-1,n))x^n y^{m-n} +$$
$$+ C(m-1,m-1)x^m y^0 = \sum_{n=0}^{m} C(m,n)x^n y^{m-n}. \quad \square$$

СЛЕДСТВИЕ 1 $\sum_{n=0}^{m} C(m,n) = 2^m$.

Доказательство $2^m = (1+1)^m = \sum_{n=0}^{n} C(m,n)1^n 1^{m-n} = \sum_{n=0}^{m} C(m,n)$. $\square$

СЛЕДСТВИЕ 2 $\sum_{n=0}^{m} (-1)^n C(m,n) = 0$.

Доказательство $0 = (-1+1)^m = \sum_{n=0}^{m} C(m,n)(-1)^n 1^{m-n} = \sum_{n=0}^{m} (-1)^n C(m,n)$. $\square$

5.3.3. Свойства биномиальных коэффициентов

Биномиальные коэффициенты обладают целым рядом замечательных свойств.

ТЕОРЕМА 1 $\sum_{n=0}^{m} nC(m,n) = m2^{m-1}$.

Доказательство Рассмотрим следующую последовательность, составленную из чисел $1,\ldots,m$. Сначала выписаны все подмножества длины 0, потом все подмножества длины 1 и т. д. Имеется $C(m,n)$ подмножеств мощности n, и каждое из них имеет длину n, таким образом, всего в этой последовательности $\sum_{n=0}^{m} nC(m,n)$ чисел. С другой стороны, каждое число x входит в эту последовательность $2^{|\{1,\ldots,m\}\setminus\{x\}|} = 2^{m-1}$ раз, а всего чисел m. $\square$

ТЕОРЕМА 2 $C(m+n,k) = \sum_{i=0}^{k} C(m,i)C(n,k-i)$.

Доказательство $C(n+m,k)$ — это число способов выбрать k предметов из $m+n$ предметов. Предметы можно выбирать в два приема: сначала выбрать i предметов из первых m предметов, а затем выбрать недостающие $k-i$ предметов из оставшихся n предметов. Отсюда общее число способов выбрать k предметов составляет $\sum_{i=0}^{k} C(m,i)C(n,k-i)$. □

ЗАМЕЧАНИЕ

Последнее свойство известно как *тождество Коши*[1].

ОТСТУПЛЕНИЕ

Данные свойства биномиальных коэффициентов нетрудно доказать, проводя алгебраические выкладки обычным образом. Например, первое свойство можно получить так:

$$\sum_{n=0}^{m} nC(m,n) = \sum_{n=1}^{m} \frac{nm!}{(m-n)!n!} = \sum_{n=1}^{m} \frac{m(m-1)!}{(m-n)!(n-1)!} =$$
$$= m\sum_{n=0}^{m-1} \frac{(m-1)!}{(m-n-1)!n!} = m\sum_{n=0}^{m-1} C(m-1,n) = m2^{m-1}.$$

В доказательстве использованы рассуждения «в комбинаторном духе», чтобы проиллюстрировать «неалгебраические» способы решения комбинаторных задач.

5.3.4. Треугольник Паскаля

Из второй формулы теоремы 5.3.1 вытекает эффективный способ рекуррентного вычисления значений биномиальных коэффициентов, который можно представить в графической форме, известной как *треугольник Паскаля*[2].

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
```

В этом равнобедренном треугольнике каждое число (кроме единиц на боковых сторонах) является суммой двух чисел, стоящих над ним. Число сочетаний $C(m,n)$ находится в $(m+1)$-м ряду на $(n+1)$-м месте.

[1] Огюстен Луи Коши (1789–1857).

[2] Блез Паскаль (1623–1662).

5.3.5. Генерация подмножеств

Элементы множества $\{1,\ldots,m\}$ естественным образом упорядочены. Поэтому каждое n-элементное подмножество этого множества также можно рассматривать как упорядоченную последовательность. На множестве таких последовательностей определяется лексикографический порядок (см. упражнение 1.8). Следующий простой алгоритм генерирует все n-элементные подмножества m-элементного множества в лексикографическом порядке.

Алгоритм 5.3 Генерация n-элементных подмножеств m-элементного множества

Вход: n — мощность подмножества, m — мощность множества, $m \geqslant n > 0$.
Выход: последовательность всех n-элементных подмножеств m-элементного множества в лексикографическом порядке.

```
for i from 1 to m do
  A[i] := i { инициализация исходного множества }
end for
if m = n then
  return A[1..n] { единственное подмножество }
end if
p := n { p — номер первого изменяемого элемента }
while p ⩾ 1 do
  yield A[1..n] { очередное подмножество в первых n элементах массива A }
  if A[n] = m then
    p := p − 1 { нельзя увеличить последний элемент }
  else
    p := n { можно увеличить последний элемент }
  end if
  if p ⩾ 1 then
    for i from n downto p do
      A[i] := A[p] + i − p + 1 { увеличение элементов }
    end for
  end if
end while
```

Обоснование Заметим, что в искомой последовательности n-элементных подмножеств (каждое из которых является возрастающей последовательностью n чисел из диапазона $1..m$) вслед за последовательностью $\langle a_1,\ldots,a_n\rangle$ следует последовательность $\langle b_1,\ldots,b_n\rangle = \langle a_1,\ldots,a_{p-1},a_p+1,a_p+2,\ldots,a_p+n-p+1\rangle$, где p — максимальный индекс, для которого $b_n = a_p+n-p+1 \leqslant m$. Другими словами, следующая последовательность получается из предыдущей заменой некоторого количества элементов в хвосте последовательности идущими подряд натуральными числами, но так, чтобы последний элемент не превосходил m, а первый изменяемый элемент был на 1 больше, чем соответствующий элемент в предыдущей последовательности. Таким образом, индекс p, начиная с которого следует изменить «хвост последовательности», определяется по значению элемента $A[n]$. Если $A[n] < m$, то следует изменять только $A[n]$, и при этом $p := n$. Если же уже $A[n] = m$, то нужно уменьшать индекс $p := p-1$, увеличивая длину изменяемого хвоста. □

Пример Последовательность n-элементных подмножеств m-элементного множества в лексикографическом порядке для $n = 3$ и $m = 4$: $(1,2,3)$, $(1,2,4)$, $(1,3,4)$, $(2,3,4)$.

ОТСТУПЛЕНИЕ

Довольно часто встречаются задачи, в которых требуется найти в некотором множестве элемент, обладающий определёнными свойствами. При этом исследуемое множество может быть велико, а его элементы устроены достаточно сложно. Если неизвестно заранее, как именно следует искать элемент, то остаётся перебирать все элементы, пока не попадётся нужный (поэтому такие задачи называют *переборными*).

При решении переборных задач совсем не обязательно и, как правило, нецелесообразно строить всё исследуемое множество (*пространство поиска*) в явном виде. Достаточно иметь итератор (см. 1.3.8), перебирающий элементы множества в подходящем порядке. Фактически, алгоритмы 1.2, 5.2 и 5.3 являются примерами таких итераторов для некоторых типичных пространств поиска.

Чтобы использовать эти алгоритмы в качестве итераторов при переборе, достаточно вставить вместо оператора **yield** проверку того, что очередной элемент является искомым.

5.3.6. Мультимножества и последовательности

В 5.1.5 показано, что число n-элементных подмножеств m-элементного множества равно $C(m,n)$. Поскольку n-элементные подмножества — это n-элементные индикаторы (см. 1.1.4), ясно, что число n-элементных индикаторов над m-элементным множеством также равно $C(m,n)$.

Общее число n-элементных мультимножеств $[x_1^{n_1},\ldots,x_m^{n_m}]$, в которых $n_1+\ldots+n_m=n$, над m-элементным множеством $X=\{x_1,\ldots,x_m\}$ нетрудно определить, если заметить, что это число способов разложить n неразличимых предметов по m ящикам, то есть число сочетаний с повторениями $V(m,n)=C(n+m-1,n)$.

Пусть задано множество $X=\{x_1,\ldots,x_m\}$. Рассмотрим последовательность $Y=\langle y_1,\ldots,y_n\rangle$, где $\forall i\ (y_i\in X)$ и в последовательности Y элемент x_i встречается n_i раз. Тогда мультимножество $\widehat{X}=[x_1^{n_1},\ldots,x_m^{n_m}]$ называется *составом* последовательности Y. Ясно, что состав любой последовательности определяется однозначно, но разные последовательности могут иметь один и тот же состав.

Пример Пусть $X=\{1,2,3\}$, $Y_1=\langle 1,2,3,2,1\rangle$, $Y_2=\langle 1,1,2,2,3\rangle$. Тогда последовательности Y_1 и Y_2 имеют один и тот же состав $\widehat{X}=[1^2,2^2,3^1]$.

ОТСТУПЛЕНИЕ

Из органической химии известно, что существуют различные вещества, имеющие один и тот же химический состав. Они называются изомерами.

Очевидно, что все последовательности над множеством $X = \{x_1, \ldots, x_m\}$, имеющие один и тот же состав $\widehat{X} = [x_1^{n_1}, \ldots, x_m^{n_m}]$, имеют одну и ту же длину $n = n_1 + \ldots + n_m$. Обозначим число последовательностей одного состава $C(n; n_1, \ldots, n_m)$.

ТЕОРЕМА $C(n; n_1, \ldots, n_m) = \dfrac{n!}{n_1! \ldots n_m!}$.

Доказательство Рассмотрим мультимножество $\widehat{X} = [x_1^{n_1}, \ldots, x_m^{n_m}]$, которое можно записать в форме последовательности $\langle x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_m, \ldots, x_m \rangle$, где элемент x_i встречается n_i раз. Ясно, что все последовательности одного состава $\widehat{X}$ получаются из указанной последовательности перестановкой элементов. При этом, если переставляются одинаковые элементы, например, x_i, то получается та же самая последовательность. Таких перестановок $n_i!$. Таким образом, общее число перестановок $n_1! \ldots n_m! C(n; n_1, \ldots, n_m)$. С другой стороны, число перестановок n элементов равно $n!$. □

5.3.7. Мультиномиальные коэффициенты

Числа $C(n; n_1, \ldots, n_m)$ встречаются в практических задачах довольно часто, поскольку обобщают случай индикаторов, которые описываются биномиальными коэффициентами, на случай произвольных мультимножеств. В частности, справедлива формула, обобщающая формулу бинома Ньютона, в которой участвуют числа $C(n; n_1, \ldots, n_m)$, поэтому их иногда называют *мультиномиальными коэффициентами*.

ТЕОРЕМА $(x_1 + \ldots + x_m)^n = \sum\limits_{n_1 + \ldots + n_m = n} C(n; n_1, \ldots, n_m) x_1^{n_1} \cdots x_m^{n_m}$.

Доказательство Индукция по m. База: при $m = 2$ по формуле бинома Ньютона имеем

$$(x_1 + x_2)^n = \sum_{i=0}^{n} C(n, i) x_1^i x_2^{n-i} = \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} x_1^i x_2^{n-i} = \sum_{n_1 + n_2 = n} \frac{n!}{n_1! n_2!} x_1^{n_1} x_2^{n_2}.$$

Пусть теперь $(x_1 + \ldots + x_{m-1})^n = \sum\limits_{n_1 + \ldots + n_{m-1} = n} C(n; n_1, \ldots, n_{m-1}) x_1^{n_1} \cdots x_{m-1}^{n_{m-1}}$.

Рассмотрим $(x_1 + \ldots + x_m)^n$. Имеем:

$$(x_1 + \ldots + x_m)^n = ((x_1 + \ldots + x_{m-1}) + x_m)^n = \sum_{i=0}^{n} C(n, i)(x_1 + \ldots + x_{m-1})^i x_m^{n-i} =$$

$$= \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} \left(\sum_{n_1 + \ldots + n_{m-1} = i} \frac{i!}{n_1! \cdots n_{m-1}!} x_1^{n_1} \cdots x_{m-1}^{n_{m-1}} \right) x_m^{n-i} =$$

$$= \sum_{i=0}^{n} \sum_{n_1 + \ldots + n_{m-1} = i} \frac{n! i!}{i!(n-i)! n_1! \cdots n_{m-1}!} x_1^{n_1} \cdots x_{m-1}^{n_{m-1}} x_m^{n-i} =$$

$$= \sum_{n_1+\ldots+n_{m-1}+n_m=n} \frac{n!}{n_1!\cdots n_{m-1}!n_m!} x_1^{n_1}\cdots x_{m-1}^{n_{m-1}} x_m^{n_m},$$

где $n_m := n - i$. □

СЛЕДСТВИЕ $\displaystyle\sum_{n_1+\ldots+n_m=n} \frac{n!}{n_1!\cdots!n_m!} = m^n.$

Доказательство $m^n = \underbrace{(1+\ldots+1)^n}_{m \text{ раз}} = \displaystyle\sum_{n_1+\ldots+n_m=n} \frac{n!}{n_1!\cdots!n_m!} 1^{n_1}1^{n_m} =$

$$= \sum_{n_1+\cdots+n_m=n} \frac{n!}{n_1!\ldots!n_m!}.$$ □

Пример При игре в преферанс трём играющим сдаётся по 10 карт из 32 карт и 2 карты остаются в «прикупе». Сколько существует различных раскладов?

$$C(32;10,10,10,2) = \frac{32!}{10!10!10!2!} = 2753294408504640.$$

5.4. Разбиения

Разбиения не рассматривались среди типовых комбинаторных конфигураций в разделе 5.1, потому что получить для них явную формулу не так просто, как для остальных. В этом разделе отмечаются основные свойства разбиений, а окончательные формулы приведены в 5.6.3.

5.4.1. Определения

Пусть $\mathcal{B} = \{B_1, \ldots, B_n\}$ — разбиение множества X из m элементов на n подмножеств:

$$B_i \subset X, \qquad \bigcup_{i=1}^{n} B_i = X, \qquad B_i \neq \varnothing, \qquad B_i \cap B_j = \varnothing \quad \text{при } i \neq j.$$

Подмножества B_i называются *блоками* разбиения.

Между разбиениями с непустыми блоками и отношениями эквивалентности существует взаимно-однозначное соответствие (см. 1.7.1). Если E_1 и E_2 — два разбиения X, то говорят, что разбиение E_1 есть *измельчение* разбиения E_2, если каждый блок E_2 есть объединение блоков E_1. Измельчение является частичным порядком на множестве разбиений.

5.4.2. Числа Стирлинга второго рода

Число разбиений m-элементного множества на n блоков называется *числом Стирлинга*[1] *второго рода* и обозначается $S(m,n)$. По определению положим

$$S(m,m) \stackrel{\text{Def}}{=} 1, \quad S(m,0) \stackrel{\text{Def}}{=} 0 \quad \text{при } m > 0,$$
$$S(0,0) \stackrel{\text{Def}}{=} 1, \quad S(m,n) \stackrel{\text{Def}}{=} 0 \quad \text{при } n > m.$$

ТЕОРЕМА 1 $S(m,n) = S(m-1,n-1) + nS(m-1,n)$.

Доказательство Пусть $\mathcal{B}$ — множество всех разбиений множества $M := \{1,\ldots,m\}$ на n блоков. Положим

$$\mathcal{B}_1 := \{X \in \mathcal{B} \mid \exists B \in X \ (B = \{m\})\}, \ \mathcal{B}_2 := \{X \in \mathcal{B} \mid \neg\exists B \in X \ (B = \{m\})\},$$

то есть в $\mathcal{B}_1$ входят разбиения, в которых элемент m образует отдельный блок, а в $\mathcal{B}_2$ — все остальные разбиения. Заметим, что $\mathcal{B}_2 = \{X \in \mathcal{B} \mid m \in X \Longrightarrow |X| > 1\}$. Тогда $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$, $\mathcal{B}_1 \cap \mathcal{B}_2 = \varnothing$. Имеем $|\mathcal{B}_1| = S(m-1,n-1)$, $|\mathcal{B}_2| = n\,S(m-1,n)$, так как все разбиения $\mathcal{B}_2$ получаются следующим образом: берём все разбиения множества $\{1,\ldots,m-1\}$ на n блоков (их $S(m-1,n)$) и в каждый блок по очереди помещаем элемент m. Следовательно,

$$S(m,n) = |\mathcal{B}| = |\mathcal{B}_1| + |\mathcal{B}_2| = S(m-1,n-1) + nS(m-1,n). \qquad \square$$

ТЕОРЕМА 2 $S(m,n) = \sum\limits_{i=n-1}^{m-1} C(m-1,i)S(i,n-1)$.

Доказательство Пусть $\mathcal{B}$ — множество всех разбиений множества $M := \{1,\ldots,m\}$ на n блоков. Рассмотрим семейство $\overline{B} := \{B \subset 2^M \mid m \in B\}$. Тогда $\mathcal{B} = \bigcup_{B\in\overline{B}} \mathcal{B}_B$, где $\mathcal{B}_B := \{X \mid X \in \mathcal{B} \ \&\ B \in X\}$, причём $\mathcal{B}_{B'} \cap \mathcal{B}_{B''} = \varnothing$, если $B' \neq B''$. Пусть $B \in \overline{B}$ и $b := |B|$. Тогда $|\mathcal{B}_B| = S(m-b,n-1)$. Заметим, что $\left|\{B \in \overline{B} \mid b = |B|\}\right| = C(m-1,b-1)$. Имеем:

$$S(m,n) = |\mathcal{B}| = \sum_{b=1}^{m-(n-1)} \left| \bigcup_{B\in\overline{B} \ \&\ |B|=b} \mathcal{B}_B \right| =$$
$$= \sum_{b=1}^{m-(n-1)} C(m-1,b-1)S(m-b,n-1) =$$
$$= \sum_{i=m-1}^{n-1} C(m-1,m-i-1)S(i,n-1) = \sum_{i=n-1}^{m-1} C(m-1,i)S(i,n-1),$$

где $i := m - b$. $\square$

[1] Джеймс Стирлинг (1699–1770).

5.4.3. Числа Стирлинга первого рода

Число сюръективных функций, то есть число размещений m предметов по n ящикам, таких, что все ящики заняты, называется *числом Стирлинга первого рода* и обозначается $s(m,n)$.

ТЕОРЕМА $s(m,n) = n!\,S(m,n)$.

Доказательство Каждое разбиение множества $\{1,\ldots,m\}$ соответствует семейству множеств уровня сюръективной функции и обратно (см. 1.7.3). Таким образом, число различных семейств множеств уровня сюръективных функций — это число Стирлинга второго рода $S(m,n)$. Всего сюръективных функций $s(m,n) = n!\,S(m,n)$, так как число сюръективных функций с заданным семейством множеств уровня равно числу перестановок множества значений функции. □

5.4.4. Число Белла

Число всех разбиений m-элементного множества называется *числом Белла*[1] и обозначается $B(m)$,

$$B(m) \stackrel{\text{Def}}{=} \sum_{n=0}^{m} S(m,n), \quad B(0) \stackrel{\text{Def}}{=} 1.$$

ТЕОРЕМА $B(m+1) = \sum\limits_{i=0}^{m} C(m,i)B(i)$.

Доказательство Пусть $\mathcal{B}$ — множество всех разбиений множества $M_1 = 1..(m+1)$. Рассмотрим множество подмножеств множества M_1, содержащих элемент $m+1$:

$$\overline{B} := \left\{ B \subset 2^{M_1} \middle| \, m+1 \in B \right\}.$$

Тогда $\mathcal{B} = \bigcup\limits_{B\in\overline{B}} \mathcal{B}_B$, где $\mathcal{B}_B := \{X \in \mathcal{B} \mid B \in X\}$. Пусть $B \in \overline{B}$ и $b = |B|$. Тогда $|\mathcal{B}_B| = B(m+1-b)$. Заметим, что $\left|\left\{B \in \overline{B} \mid |B| = b\right\}\right| = C(m,b-1)$. Следовательно,

$$B(m+1) = |\mathcal{B}| = \sum_{b=1}^{m+1} C(m,b-1)B(m-b+1) =$$

$$= \sum_{i=m}^{0} C(m,m-i)B(i) = \sum_{i=0}^{m} C(m,i)B(i),$$

где $i := m-b+1$. □

5.5. Включения и исключения

Приведённые в предыдущих четырёх разделах формулы и алгоритмы дают способы вычисления комбинаторных чисел для некоторых распространённых комбинаторных конфигураций. Практические задачи не всегда прямо сводятся к

[1] Эрик Темпл Белл (1883–1960).

известным комбинаторным конфигурациям. В этом случае используются различные методы сведения одних комбинаторных конфигураций к другим. В этом и двух следующих разделах рассматриваются три наиболее часто используемых метода.

Мы начинаем с самого простого и прямолинейного, но имеющего ограниченную область применения метода включений и исключений.

5.5.1. Объединение конфигураций

Часто комбинаторная конфигурация является объединением других, число комбинаций в которых вычислить проще. В таком случае требуется уметь вычислять число комбинаций в объединении. В простых случаях формулы для вычисления очевидны:

$$|A \cup B| = |A| + |B| - |A \cap B|,$$
$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|.$$

Пример Сколько существует натуральных чисел, меньших 1000, которые не делятся ни на 3, ни на 5, ни на 7? Всего чисел, меньших тысячи, 999. Из них:

$999 : 3 = 333$ делятся на 3,

$999 : 5 = 199$ делятся на 5,

$999 : 7 = 142$ делятся на 7,

$999 : (3 * 5) = 66$ делятся на 3 и на 5,

$999 : (3 * 7) = 47$ делятся на 3 и на 7,

$999 : (5 * 7) = 28$ делятся на 5 и на 7,

$999 : (3 * 5 * 7) = 9$ делятся на 3, на 5 и на 7.

Имеем: $999 - (333 + 199 + 142 - 66 - 47 - 28 + 9) = 457$.

5.5.2. Формула включений и исключений

Следующая формула, известная как *формула включений и исключений*, позволяет вычислить мощность объединения нескольких множеств, если известны их мощности и мощности всех возможных пересечений.

ТЕОРЕМА

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| + \sum_{1 \leqslant i < j < k \leqslant n} |A_i \cap A_j \cap A_k| - \ldots$$
$$+ (-1)^{n-1} |A_1 \cap \ldots \cap A_n|.$$

Доказательство Индукция по n. Для $n = 2, 3$ теорема проверена в предыдущем подразделе. Пусть

$$\left|\bigcup_{i=1}^{n-1} A_i\right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \ldots + (-1)^{n-2}|A_i \cap \ldots \cap A_{n-1}|.$$

Заметим, что $\left(\bigcup\limits_{i=1}^{n-1} A_i\right) \cap A_n = \bigcup\limits_{i=1}^{n-1} (A_i \cap A_n)$, и по индукционному предположению

$$\left|\bigcup_{i=1}^{n-1} (A_i \cap A_n)\right| = \sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \cdots + \\ + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n|.$$

Тогда

$$\left|\bigcup_{i=1}^{n} A_i\right| = \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n\right| = \left|\bigcup_{i=1}^{n-1} A_i\right| + |A_n| - \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n\right| =$$

$$= \left(\sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \ldots + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1}|\right) +$$

$$+ |A_n| - \left(\sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \ldots + \right.$$

$$\left. + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n|\right) =$$

$$= \left(\sum_{i=1}^{n-1} |A_i| + |A_n|\right) - \left(\sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \sum_{i=1}^{n-1} |A_i \cap A_n|\right) + \ldots -$$

$$- (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n| =$$

$$= \sum_{i=1}^{n} |A_i| - \sum_{1\leqslant i<j\leqslant n} |A_i \cap A_j| + \ldots + (-1)^{n-1}|A_1 \cap \ldots \cap A_n|. \qquad \square$$

ЗАМЕЧАНИЕ

Обозначения сумм с неравенствами в пределах суммирования, использованные в формулировке и доказательстве теоремы, являются не более чем сокращённой формой записи кратных сумм. Например,

$$\sum_{1\leqslant i<j\leqslant n} \quad \text{означает} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n}.$$

5.5.3. Число булевых функций, существенно зависящих от всех своих переменных

Рассмотрим применение формулы включений и исключений на примере следующей задачи. Пусть $p_n \stackrel{\text{Def}}{=} |P_n| = 2^{2^n}$ — число всех булевых функций n переменных, а $\widetilde{p}_n$ — число булевых функций, существенно зависящих от всех n переменных (см. 3.1.2). Пусть P_n^i — множество булевых функций, у которых переменная x_i фиктивная (кроме x_i могут быть и другие фиктивные переменные). Имеем:

$$\widetilde{p}_n = |P_n \setminus (P_n^1 \cup \ldots \cup P_n^n)| = \left| P_n \setminus \bigcup_{i=1}^{n} P_n^i \right|.$$

С другой стороны, $|P_n^i| = 2^{2^{n-1}}$, более того, $|P_n^{i_1} \cap \ldots \cap P_n^{i_k}| = 2^{2^{n-k}}$ (см. упражнение 3.1). Следовательно,

$$\begin{aligned}\widetilde{p}_n &= 2^{2^n} - \left(\sum_{i=1}^{n} |P_n^i| - \sum_{1 \leqslant i < j \leqslant n} |P_n^i \cap P_n^j| + \ldots + (-1)^{n-1} |P_n^1 \cap \ldots \cap P_n^n| \right) = \\ &= 2^{2^n} - \left(C(n,1) 2^{2^{n-1}} - C(n,2) 2^{2^{n-2}} + \ldots + (-1)^{n-1} C(n,n) 2 \right) = \\ &= \sum_{i=0}^{n} (-1)^i C(n,i) 2^{2^{n-i}}.\end{aligned}$$

5.6. Формулы обращения

Очень полезную, но специфическую группу приёмов образуют различные способы преобразования уже полученных комбинаторных выражений. В этом разделе рассматривается один частный, но важный случай.

5.6.1. Теорема обращения

Пусть $a_{n,k}$ и $b_{n,k}$ — некоторые (комбинаторные) числа, зависящие от параметров n и k, причём $0 \leqslant k \leqslant n$. Если известно выражение чисел $a_{n,k}$ через числа $b_{n,k}$, то в некоторых случаях можно найти и выражение чисел $b_{n,k}$ через числа $a_{n,k}$, то есть решить комбинаторное уравнение.

ТЕОРЕМА *Пусть*

$$\forall n \left(\forall k \leqslant n \left(a_{n,k} = \sum_{i=0}^{n} \lambda_{n,k,i} b_{n,i} \right) \right)$$

и пусть

$$\exists \mu_{n,k,i} \left(\forall k \leqslant n \left(\forall m \leqslant n \left(\sum_{i=0}^{n} \mu_{n,k,i} \lambda_{n,i,m} = \begin{cases} 1, & m = k, \\ 0, & m \neq k \end{cases} \right) \right) \right).$$

Тогда

$$\forall k \leqslant n \left(b_{n,k} = \sum_{i=0}^{n} \mu_{n,k,i} a_{n,i} \right).$$

Доказательство

$$\sum_{i=0}^{n} \mu_{n,k,i} a_{n,i} = \sum_{i=0}^{n} \mu_{n,k,i} \left(\sum_{m=0}^{n} \lambda_{n,i,m} b_{n,m} \right) =$$
$$= \sum_{m=0}^{n} \left(\sum_{i=0}^{n} \mu_{n,k,i} \lambda_{n,i,m} \right) b_{n,m} = b_{n,k}. \qquad \square$$

5.6.2. Формулы обращения для биномиальных коэффициентов

Применение теоремы обращения предполагает отыскание для заданных чисел $\lambda_{n,k,i}$ (коэффициентов комбинаторного уравнения) соответствующих чисел $\mu_{n,k,i}$, удовлетворяющих условию теоремы обращения. Особенно часто числами $\lambda_{n,k,i}$ являются биномиальные коэффициенты.

ЛЕММА $\displaystyle\sum_{i=0}^{n} (-1)^{i-m} C(n,i) C(i,m) = \begin{cases} 1, & m = n, \\ 0, & m < n. \end{cases}$

Доказательство Используя формулу 3 из подраздела 5.3.1 и тот факт, что $C(n-m, i-m) = 0$ при $i \leqslant m$, имеем:

$$\sum_{i=0}^{n} (-1)^{i-m} C(n,i) C(i,m) = \sum_{i=0}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) =$$
$$= \sum_{i=m}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) =$$
$$= C(n,m) \sum_{i=m}^{n} (-1)^{i-m} C(n-m, i-m).$$

Но при $m < n$ имеем:

$$\sum_{i=m}^{n} (-1)^{i-m} C(n-m, i-m) = \sum_{j=0}^{n-m} (-1)^{j} C(n-m, j) = 0,$$

где $j := i - m$. С другой стороны, при $m = n$ имеем:

$$C(n,m) \sum_{i=m}^{n} (-1)^{i-m} C(n-m, i-m) = C(n,n)(-1)^{n-n} C(0,0) = 1. \qquad \square$$

ТЕОРЕМА 1 *Если* $a_{n,k} = \sum_{i=0}^{k} C(k,i)b_{n,i}$, *то* $b_{n,k} = \sum_{i=0}^{k}(-1)^{k-i}C(k,i)a_{n,i}$.

Доказательство Здесь $\lambda_{n,k,i} = C(k,i)$ и $\mu_{n,k,i} = (-1)^{k-i}C(k,i)$. При $k \leqslant n, m \leqslant n$ имеем:

$$\sum_{i=0}^{n}\mu_{n,k,i}\lambda_{n,i,m} = \sum_{i=0}^{n}(-1)^{k-i}C(k,i)C(i,m) =$$

$$= \sum_{i=0}^{k}(-1)^{k-i+m-m}C(k,i)C(i,m) =$$

$$= (-1)^{k-m}\sum_{i=0}^{k}(-1)^{i-m}C(k,i)C(i,m) = \begin{cases}1, & k=m,\\ 0, & k\neq m.\end{cases}$$ □

ТЕОРЕМА 2 *Если* $a_{n,k} = \sum_{i=k}^{n} C(i,k)b_{n,i}$, *то* $b_{n,k} = \sum_{i=k}^{n}(-1)^{i-k}C(i,k)a_{n,i}$.

Доказательство Здесь $\lambda_{n,k,i} = C(i,k)$ и $\mu_{n,k,i} = (-1)^{i-k}C(i,k)$. При $k \leqslant n, m \leqslant n$ имеем:

$$\sum_{i=0}^{n}\mu_{n,k,i}\lambda_{n,i,m} = \sum_{i=0}^{n}(-1)^{i-k}C(i,k)C(m,i) =$$

$$= \sum_{i=0}^{n}(-1)^{i-k}C(m,i)C(i,k) = \begin{cases}1, & k=n,\\ 0, & k\neq n.\end{cases}$$ □

5.6.3. Формулы для чисел Стирлинга

В качестве примера использования формул обращения рассмотрим получение явных формул для чисел Стирлинга первого и второго рода. Рассмотрим множество функций $f\colon A \to B$, где $|A| = n$ и $|B| = k$. Число всех таких функций равно k^n. С другой стороны, число функций f, таких, что $|f(A)| = i$, равно $s(n,i)$, поскольку $s(n,i)$ — это число сюръективных функций $f\colon 1..n \to 1..i$. Но множество значений функции (при заданном i) можно выбрать $C(k,i)$ способами. Поэтому

$$k^n = \sum_{i=0}^{k} C(k,i)s(n,i).$$

Обозначив $a_{n,k} := k^n$ и $b_{n,i} := s(n,i)$, имеем по теореме 1 предыдущего подраздела

$$s(n,k) = \sum_{i=0}^{k}(-1)^{k-i}C(k,i)i^n.$$

Учитывая связь чисел Стирлинга первого и второго рода, окончательно имеем:

$$S(n,k) = \frac{1}{n!}\sum_{i=0}^{k}(-1)^{k-i}C(k,i)i^n.$$

5.7. Производящие функции

Для решения комбинаторных задач в некоторых случаях можно использовать методы математического анализа. Разнообразие применяемых здесь приёмов весьма велико и не может быть в полном объёме рассмотрено в рамках этой книги. В данном разделе рассматривается только основная идея метода производящих функций, применение которой иллюстрируется тремя простыми примерами. Более детальное рассмотрение можно найти в литературе, например, в [22].

5.7.1. Основная идея

Пусть есть последовательность комбинаторных чисел a_i и последовательность функций $\varphi_i(x)$. Рассмотрим формальный ряд

$$\mathcal{F}(x) \stackrel{\text{Def}}{=} \sum_i a_i\varphi_i(x).$$

$\mathcal{F}(x)$ называется *производящей функцией* (для заданной последовательности комбинаторных чисел a_i относительно заданной последовательности функций $\varphi_i(x)$).

Обычно используют $\varphi_i(x) \stackrel{\text{Def}}{=} x^i$ или $\varphi_i(x) \stackrel{\text{Def}}{=} x^i/i!$.

Пример Из формулы бинома Ньютона при $y = 1$ имеем:

$$(1+x)^n = \sum_{i=0}^{n} C(n,i)x^i.$$

Таким образом, $(1+x)^n$ является производящей функцией для биномиальных коэффициентов.

5.7.2. Метод неопределённых коэффициентов

Из математического анализа известно, что если

$$\mathcal{F}(x) = \sum_i a_i\varphi_i(x) \quad \text{и} \quad \mathcal{F}(x) = \sum_i b_i\varphi_i(x),$$

то $\forall i \ (a_i = b_i)$ (для рассматриваемых здесь систем функций φ_i).

В качестве примера применения производящих функций докажем следующее тождество.

ТЕОРЕМА $C(2n,n) = \sum\limits_{k=0}^{n} C(n,k)^2$.

Доказательство Имеем: $(1+x)^{2n} = (1+x)^n(1+x)^n$. Следовательно,

$$\sum_{i=0}^{2n} C(2n,i)x^i = \sum_{i=0}^{n} C(n,i)x^i \cdot \sum_{i=0}^{n} C(n,i)x^i.$$

Приравняем коэффициент при x^n:

$$C(2n,n) = \sum_{k=0}^{n} C(n,k)C(n,n-k) = \sum_{k=0}^{n} C(n,k)^2. \qquad \square$$

5.7.3. Числа Фибоначчи

Числа Фибоначчи[1] $F(n)$ определяются следующим образом:

$$F(0) \stackrel{\text{Def}}{=} 1,\ F(1) \stackrel{\text{Def}}{=} 1,\ F(n+2) \stackrel{\text{Def}}{=} F(n+1) + F(n).$$

Найдём выражение для $F(n)$ через n. Для этого найдём производящую функцию $\varphi(x)$ для последовательности чисел $F(n)$. Имеем:

$$\begin{aligned}
\varphi(x) &= \sum_{n=0}^{\infty} F(n)x^n = F(0)x^0 + F(1)x^1 + \sum_{n=2}^{\infty} \left(F(n-2) + F(n-1)\right)x^n = \\
&= 1 + x + \sum_{n=2}^{\infty} F(n-2)x^n + \sum_{n=2}^{\infty} F(n-1)x^n = \\
&= 1 + x + x^2 \sum_{n=2}^{\infty} F(n-2)x^{n-2} + x \sum_{n=2}^{\infty} F(n-1)x^{n-1} = \\
&= 1 + x + x^2 \sum_{n=0}^{\infty} F(n)x^n + x\left(\sum_{n=0}^{\infty} F(n)x^n - F(0)\right) = \\
&= 1 + x + x^2\varphi(x) + x\left(\varphi(x) - 1\right).
\end{aligned}$$

Решая это функциональное уравнение относительно $\varphi(x)$, получаем, что

$$\varphi(x) = \frac{1}{1 - x - x^2}.$$

Последнее выражение нетрудно разложить в ряд по степеням x. Действительно, уравнение $1 - x - x^2 = 0$ имеет корни $x_1 = -(1+\sqrt{5})/2$ и $x_2 = -(1-\sqrt{5})/2$, причём, как нетрудно убедиться,

$$1 - x - x^2 = (1 - ax)(1 - bx), \quad \text{где } a = \frac{1+\sqrt{5}}{2}, \quad b = \frac{1-\sqrt{5}}{2}.$$

[1] Фибоначчи (Леонардо Пизанский, 1180–1228).

Далее,

$$\frac{1}{(1-ax)(1-bx)} = \frac{\alpha}{1-ax} + \frac{\beta}{1-bx}, \quad \text{где } \alpha = \frac{a}{a-b}, \quad \beta = \frac{-b}{a-b}.$$

Из математического анализа известно, что для малых x

$$\frac{1}{1-\gamma x} = \sum_{n=0}^{\infty} \gamma^n x^n.$$

Таким образом,

$$\varphi(x) = \frac{\alpha}{1-ax} + \frac{\beta}{1-bx} = \alpha \sum_{n=0}^{\infty} a^n x^n + \beta \sum_{n=0}^{\infty} b^n x^n =$$

$$= \sum_{n=0}^{\infty} \frac{a}{a-b} a^n x^n - \sum_{n=0}^{\infty} \frac{b}{a-b} b^n x^n = \sum_{n=0}^{\infty} \frac{a^{n+1} - b^{n+1}}{a-b} x^n.$$

Окончательно получаем

$$F(n) = \frac{a^{n+1} - b^{n+1}}{a-b} = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right).$$

5.7.4. Числа Каталана

Числа Каталана[1] $C(n)$ можно определить следующим образом:

$$C(0) \stackrel{\text{Def}}{=} 1, \quad C(n) \stackrel{\text{Def}}{=} \sum_{k=0}^{n-1} C(k)C(n-k-1).$$

Числа Каталана используются при решении различных комбинаторных задач.

Пример Пусть $\langle S, * \rangle$ — полугруппа и нужно вычислить элемент $s_1 * \cdots * s_n$. Сколькими способами это можно сделать? То есть сколькими способами можно расставить скобки, определяющие порядок вычисления выражения? Обозначим число способов $C(n)$. Ясно, что $C(0) = 1$. При любой схеме вычислений на каждом шаге выполняется некоторое вхождение операции $*$ над соседними элементами и результат ставится на их место. Пусть последней выполняется то вхождение операции $*$, которое имеет номер k в исходном выражении. При этом слева от выбранного вхождения знака $*$ находилось и было выполнено $k-1$ знаков операции $*$, а справа, соответственно, $(n-1)-(k-1) = n-k$ знаков операции $*$. Тогда ясно, что

$$C(n) = \sum_{k=1}^{n} C(k-1)C(n-k) = \sum_{k=0}^{n-1} C(k)C(n-k-1),$$

и ответом на поставленный вопрос является число Каталана $C(n)$.

Числа Каталана выражаются через биномиальные коэффициенты. Получим это выражение, используя метод производящих функций.

[1] Евгений Чарльз Каталан (1814–1894).

ТЕОРЕМА $C(n) = \dfrac{C(2n,n)}{n+1}$.

Доказательство Найдём производящую функцию для чисел Каталана

$$\varphi(x) = \sum_{n=0}^{\infty} C(n)x^n.$$

Для этого рассмотрим квадрат этой функции:

$$\begin{aligned}\varphi^2(x) &= \left(\sum_{n=0}^{\infty} C(n)x^n\right)^2 = \sum_{m=0}^{\infty} C(m)x^m \cdot \sum_{n=0}^{\infty} C(n)x^n = \\ &= \sum_{m,n=0}^{\infty} C(m)C(n)x^{m+n} = \sum_{n=0}^{\infty}\sum_{k=0}^{n} C(k)C(n-k)x^n = \\ &= \sum_{n=0}^{\infty} C(n+1)x^n = \sum_{n=0}^{\infty} C(n+1)\frac{x^{n+1}}{x} = \\ &= \frac{1}{x}\left(\sum_{n=0}^{\infty} C(n)x^n - C(0)\right) = \frac{1}{x}\left(\varphi(x) - 1\right).\end{aligned}$$

Решая уравнение $\varphi(x) = x\varphi^2(x) + 1$ относительно функции $\varphi(x)$, имеем:

$$\varphi(x) = \frac{1 \pm \sqrt{1-4x}}{2x}.$$

Обозначим $f(x) := \sqrt{1-4x}$ и разложим $f(x)$ в ряд по формуле Тейлора:

$$f(x) = f(0) + \sum_{k=1}^{\infty} \frac{f^{(k)}(0)}{k!}x^k.$$

Имеем:

$$\begin{aligned}\frac{d^k}{dx^k}(1-4x)^{\frac{1}{2}} &= \frac{1}{2}\cdot\left(\frac{1}{2}-1\right)\cdot\cdots\cdot\left(\frac{1}{2}-k+1\right)\cdot(1-4x)^{\frac{1}{2}-k}\cdot(-4)^k = \\ &= -2^k\cdot 1\cdot 3\cdot\cdots\cdot(2k-3)\cdot(1-4x)^{\frac{1}{2}-k} = \\ &= -2^k\cdot(2k-3)!!\cdot(1-4x)^{\frac{1}{2}-k} = -2^k\frac{(2k-3)!}{2^{k-2}(k-2)!}(1-4x)^{\frac{1}{2}-k} = \\ &= -2\frac{(2k-2)!(k-1)(k-1)!}{(2k-2)(k-1)!(k-1)!}(1-4x)^{\frac{1}{2}-k} = \\ &= -2(k-1)!\frac{(2k-2)!}{(k-1)!(k-1)!}(1-4x)^{\frac{1}{2}-k} = \\ &= -2(k-1)!C(2k-2,k-1)(1-4x)^{\frac{1}{2}-k}.\end{aligned}$$

Таким образом,

$$f^{(k)}(0) = -2(k-1)!C(2k-2,k-1) \quad \text{и} \quad f(x) = 1 - 2\sum_{k=1}^{\infty}\frac{1}{k}C(2k-2,k-1)x^k.$$

Подставляя выражение $f(x)$ в формулу для $\varphi(x)$, следует выбрать знак «минус» перед корнем, чтобы удовлетворить условию $C(0) = 1$. Окончательно имеем:

$$\sum_{n=0}^{\infty} C(n)x^n = \varphi(x) = \frac{1 - f(x)}{2x} = \frac{1 - 1 + 2\sum_{n=1}^{\infty} \frac{1}{n} C(2n-2, n-1)x^n}{2x} =$$

$$= \sum_{n=1}^{\infty} \frac{1}{n} C(2(n-1), n-1)x^{n-1} = \sum_{n=0}^{\infty} \frac{C(2n, n)}{n+1} x^n,$$

и по методу неопределённых коэффициентов $C(n) = C(2n, n)/(n+1)$. □

Комментарии

Сведения из области комбинаторного анализа в том или ином объёме приводятся в любом учебнике по дискретной математике (см., например, [11], [9]). Учебник [13] отличается особенно подробным и доходчивым изложением. Явные формулы для комбинаторных чисел часто используются при оценке размера пространства поиска в переборных задачах программирования. Очень богатый набор полезных формул для комбинаторных чисел можно найти в книге [22]. Все алгоритмы этой главы заимствованы (в модифицированном виде) из книги [8].

Упражнения

5.1. Доказать, что $A(m, n) = A(m-1, n) + nA(m-1, n-1)$.

5.2. Доказать, что множество перестановок с чётным числом инверсий образует группу (*альтернированную* или *знакопеременную* подгруппу симметрической группы).

5.3. Доказать, что $mC(m-1, n-1) = nC(m, n)$.

5.4. Доказать, что число последовательностей длины n, составленных из элементов множества $1..m$ и содержащих каждый элемент множества $1..m$ по крайней мере один раз, равно $m!S(n, m)$.

5.5. Рассмотрим множество функций $f\colon X \to X$, где $|X| = n$. Элемент $x \in X$ называется *неподвижной точкой* функции f, если $f(x) = x$. Пусть H_n — множество функций, не имеющих неподвижных точек. Определить, чему равно $|H_n|$.

5.6. Пусть $\widetilde{p_n}$ — число булевых функций, существенно зависящих от всех своих переменных. Очевидно, что

$$2^{2^n} = \sum_{i=0}^{n} C(n, i)\widetilde{p_i}.$$

Получить явную формулу для $\widetilde{p_n}$, используя формулы обращения.

5.7. Доказать, что любое натуральное число можно представить как сумму попарно различных чисел Фибоначчи.

Глава 6 Кодирование

Вопросы кодирования издавна играли заметную роль в математике.

Примеры

1. *Десятичная позиционная система счисления* — это универсальный способ кодирования чисел, в том числе натуральных. *Римские цифры* — другой способ кодирования небольших натуральных чисел, причём гораздо более наглядный и естественный: палец — I, пятерня — V, две пятерни — X[1]. Однако при этом способе кодирования трудно выполнять арифметические операции над большими числами, поэтому он был вытеснен позиционной десятичной системой.
2. *Декартовы координаты* — способ кодирования геометрических объектов числами.

Ранее средства кодирования играли вспомогательную роль и не рассматривались как отдельный предмет математического изучения, но с появлением компьютеров ситуация радикально изменилась. Кодирование буквально пронизывает информационные технологии и является центральным вопросом при решении самых разных (практически всех) задач программирования. Вот несколько примеров:

- представление данных произвольной природы (например чисел, текста, графики) в памяти компьютера;
- защита информации от несанкционированного доступа;
- обеспечение помехоустойчивости при передаче данных по каналам связи;
- сжатие информации в базах данных.

ЗАМЕЧАНИЕ

Само составление текста программы часто и совершенно справедливо называют кодированием.

[1] Плотники часто маркируют брёвна сруба римскими цифрами, потому что их легко вырубить топором.

Не ограничивая общности, задачу кодирования можно сформулировать следующим образом. Пусть заданы алфавиты $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$ и функция F: $A^* \to B^*$, причём $\mathrm{Dom}\, f = S$, где S — некоторое множество слов в алфавите A, $S \subset A^*$. Тогда функция F называется *кодированием*, элементы множества S — *сообщениями*, а элементы $\beta = F(\alpha)$, $\alpha \in S$, $\beta \in B^*$ — *кодами* (соответствующих сообщений). Обратная функция F^{-1} (если она существует!) называется *декодированием*. Если $|B| = m$, то F называется *m-ичным кодированием*. Наиболее распространенный случай $B = \{0, 1\}$ — *двоичное* кодирование. Именно этот случай рассматривается в последующих разделах; слово «двоичное» опускается. Типичная задача теории кодирования формулируется следующим образом: при заданных алфавитах A, B и множестве сообщений S найти такое кодирование F, которое обладает определёнными свойствами (то есть удовлетворяет заданным ограничениям) и оптимально в некотором смысле. Критерий оптимальности, как правило, связан с минимизацией длин кодов. Свойства, которые требуются от кодирования, бывают самой разнообразной природы:

- Существование декодирования, или *однозначность* кодирования: функция кодирования F обладает тем свойством, что $\alpha_1 \neq \alpha_2 \Longrightarrow F(\alpha_1) \neq F(\alpha_2)$. Это очень естественное свойство, однако даже оно требуется не всегда. Например, трансляция программы на языке высокого уровня в машинные команды — это кодирование, для которого не требуется однозначного декодирования.

- *Помехоустойчивость*, или исправление ошибок: продолжение функции декодирования F^{-1} обладает тем свойством, что $F^{-1}(\beta) = F^{-1}(\beta')$, где $\beta \in \mathrm{Im}\, F$, $\beta' \in B^* \setminus \mathrm{Im}\, F$, если β' в определённом смысле близко к β (см. 6.3).

- Заданная сложность (или простота) кодирования и декодирования. Например, в криптографии изучаются такие способы кодирования, при которых функция F вычисляется просто, но определение значения функции F^{-1} требует очень сложных вычислений (см. 6.5.5).

Большое значение для задач кодирования имеет природа множества сообщений S. При одних и тех же алфавитах A, B и требуемых свойствах кодирования F оптимальные решения для разных S могут кардинально различаться. Для описания множества S (как правило, очень большого или бесконечного) применяются различные методы:

- теоретико-множественное описание, например $S = \{\alpha \mid \alpha \in A^* \ \&\ |\alpha| = n\}$;

- вероятностное описание, например $S = A^*$, и заданы вероятности p_i появления букв a_i в сообщении, $\sum_{i=1}^{n} p_i = 1$;

- логико-комбинаторное описание, например S задано порождающей формальной грамматикой.

В этой главе рассматривается несколько наиболее важных задач теории кодирования и демонстрируется применение большей части вышеупомянутых методов.

6.1. Алфавитное кодирование

Кодирование F может сопоставлять код всему сообщению из множества S как единому целому или же строить код сообщения из кодов его частей. Элементарной частью сообщения является одна буква алфавита A. Этот простейший случай рассматривается в этом и следующих двух разделах.

6.1.1. Таблица кодов

Алфавитное (или *побуквенное*) кодирование задается *схемой* (или *таблицей кодов*) σ:

$$\sigma \stackrel{\text{Def}}{=} \langle a_1 \to \beta_1, \ldots, a_n \to \beta_n \rangle, \qquad a_i \in A, \quad \beta_i \in B^*.$$

Множество кодов букв $V \stackrel{\text{Def}}{=} \{\beta_i\}$ называется множеством *элементарных кодов* (множеством *кодовых слов*). Алфавитное кодирование пригодно для любого множества сообщений S:

$$F\colon A^* \to B^*, \qquad a_{i_1} \ldots a_{i_k} = \alpha \in A^*, \qquad F(\alpha) \stackrel{\text{Def}}{=} \beta_{i_1} \ldots \beta_{i_k}.$$

Пример Рассмотрим алфавиты $A := \{0,1,2,3,4,5,6,7,8,9\}$, $B := \{0,1\}$ и схему

$$\sigma_1 := \langle 0 \to 0, 1 \to 1, 2 \to 10, 3 \to 11, 4 \to 100, 5 \to 101, 6 \to 110,$$
$$7 \to 111, 8 \to 1000, 9 \to 1001 \rangle.$$

Эта схема однозначна, но кодирование не является взаимно однозначным:

$$F_{\sigma_1}(333) = 111111 = F_{\sigma_1}(77),$$

а значит, декодирование невозможно. С другой стороны, схема

$$\sigma_2 := \langle 0 \to 0000, 1 \to 0001, 2 \to 0010, 3 \to 0011, 4 \to 0100, 5 \to 0101, 6 \to 0110,$$
$$7 \to 0111, 8 \to 1000, 9 \to 1001 \rangle,$$

известная под названием «*двоично-десятичное* кодирование», допускает однозначное декодирование.

6.1.2. Разделимые схемы

Рассмотрим схему алфавитного кодирования σ и различные слова, составленные из элементарных кодов. Схема σ называется *разделимой*, если

$$\beta_{i_1} \ldots \beta_{i_k} = \beta_{j_1} \ldots \beta_{j_l} \Longrightarrow k = l \;\&\; \forall t \in 1..k \;\; (i_t = j_t),$$

то есть любое слово, составленное из элементарных кодов, единственным образом разлагается на элементарные коды. Алфавитное кодирование с разделимой схемой допускает декодирование.

Если таблица кодов содержит одинаковые элементарные коды, то есть если

$$\exists i,j \ \ (i \neq j \ \& \ \beta_i = \beta_j),$$

где $\beta_i, \beta_j \in V$, то схема заведомо не является разделимой. Такие схемы далее не рассматриваются, то есть

$$\forall i \neq j \ \ (\beta_i, \beta_j \in V \Longrightarrow \beta_i \neq \beta_j).$$

6.1.3. Префиксные схемы

Схема σ называется *префиксной*, если элементарный код одной буквы не является префиксом элементарного кода другой буквы:

$$\neg\exists \beta_i, \beta_j \in V, \beta \in B^* \ \ (i \neq j \ \& \ \beta_i = \beta_j\beta).$$

ТЕОРЕМА *Префиксная схема является разделимой.*

Доказательство От противного. Пусть кодирование с префиксной схемой σ не является разделимым. Тогда существует такое слово $\beta \in F_\sigma(A^*)$, что

$$\beta = \beta_{i_1} \ldots \beta_{i_k} = \beta_{j_1} \ldots \beta_{j_l} \ \& \ \exists t \ \ (\forall s < t \ \ (\beta_{i_s} = \beta_{j_s} \ \& \ \beta_{i_t} \neq \beta_{j_t})).$$

Поскольку $\beta_{i_t} \ldots \beta_{i_k} = \beta_{jt} \ldots \beta_{jl}$, значит, $\exists \beta' \ \ (\beta_{i_t} = \beta_{j_t}\beta' \vee \beta_{j_t} = \beta_{i_t}\beta')$, но это противоречит тому, что схема префиксная. □

ЗАМЕЧАНИЕ

Свойство быть префиксной достаточно, но не необходимо для разделимости схемы.

Пример Разделимая, но не префиксная схема:

$$A = \{a, b\}, \quad B = \{0, 1\}, \quad \sigma = \langle a \to 0, b \to 01 \rangle.$$

6.1.4. Неравенство Макмиллана

Чтобы схема алфавитного кодирования была разделимой, необходимо, чтобы длины элементарных кодов удовлетворяли определённому соотношению, известному как *неравенство Макмиллана.*

ТЕОРЕМА 1 *Если схема* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *разделима, то*

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leqslant 1, \qquad \text{где } l_i := |\beta_i|.$$

Доказательство Обозначим $l := \max\limits_{i \in 1..n} l_i$. Рассмотрим некоторую k-ю степень левой части неравенства

$$\left(\sum_{i=1}^{n} 2^{-l_i}\right)^k.$$

Раскрывая скобки, имеем сумму

$$\sum_{(i_1,\ldots,i_k)} (2^{l_{i_1}+\ldots+l_{i_k}})^{-1},$$

где $i_1, \ldots, i_k$ — различные наборы номеров элементарных кодов. Обозначим через $\nu(k,t)$ количество входящих в эту сумму слагаемых вида $1/2^t$, где $t = l_{i_1}+\ldots+l_{i_k}$. Для некоторых t может быть, что $\nu(k,t) = 0$. Приводя подобные, имеем сумму

$$\sum_{t=1}^{kl} \frac{\nu(k,t)}{2^t}.$$

Каждому слагаемому вида $(2^{l_{i_1}+\ldots+l_{i_k}})^{-1}$ можно сопоставить код (слово в алфавите B) вида $\beta_{i_1} \ldots \beta_{i_k}$. Это слово состоит из k элементарных кодов и имеет длину t.

Таким образом, $\nu(k,t)$ — это число некоторых слов вида $\beta_{i_1} \ldots \beta_{i_k}$, таких, что $|\beta_{i_1} \ldots \beta_{i_k}| = t$. В силу разделимости схемы $\nu(k,t) \leqslant 2^t$, в противном случае заведомо существовали бы два одинаковых слова $\beta_{i_1} \ldots \beta_{i_k} = \beta_{j_1} \ldots \beta_{j_k}$, допускающих различное разложение. Имеем:

$$\sum_{t=1}^{kl} \frac{\nu(k,t)}{2^t} \leqslant \sum_{t=1}^{kl} \frac{2^t}{2^t} = kl.$$

Следовательно, $\forall k \left(\left(\sum\limits_{i=1}^{n} 2^{-l_i}\right)^k \leqslant kl\right)$, и значит, $\forall k \left(\sum\limits_{i=1}^{n} 2^{-l_i} \leqslant \sqrt[k]{kl}\right)$, откуда

$$\sum_{i=1}^{n} 2^{-l_i} \leqslant \lim_{k\to\infty} \sqrt[k]{kl} = 1. \qquad \square$$

Неравенство Макмиллана является не только необходимым, но и в некотором смысле достаточным условием разделимости схемы алфавитного кодирования.

Теорема 2 *Если числа $l_1, \ldots, l_n$ удовлетворяют неравенству*

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leqslant 1,$$

то существует такая разделимая (даже префиксная) схема алфавитного кодирования $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$, что $\forall i \ (|\beta_i| = l_i)$.

Доказательство Без ограничения общности можно считать, что $l_1 \leqslant l_2 \leqslant \ldots \leqslant l_n$. Разобьем множество $\{l_1, \ldots, l_n\}$ на классы эквивалентности по равенству $\{L_1, \ldots, L_m\}$, $m \leqslant n$. Пусть $\lambda_i \in L_i$, $\mu_i := |L_i|$. Тогда $\sum\limits_{i=1}^{m} \mu_i = n$, $\lambda_1 < \lambda_2 < \ldots < \lambda_m$. В этих обозначениях неравенство Макмиллана можно записать так:

$$\sum_{i=1}^{m} \frac{\mu_i}{2^{\lambda_i}} \leqslant 1.$$

Из этого неравенства следуют m неравенств для частичных сумм:

$$\frac{\mu_1}{2^{\lambda_1}} \leqslant 1 \Longrightarrow \mu_1 \leqslant 2^{\lambda_1},$$

$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} \leqslant 1 \Longrightarrow \mu_2 \leqslant 2^{\lambda_2} - \mu_1 2^{\lambda_2 - \lambda_1},$$

$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} + \ldots + \frac{\mu_m}{2^{\lambda_m}} \leqslant 1 \Longrightarrow \mu_m \leqslant 2^{\lambda_m} - \mu_1 2^{\lambda_m - \lambda_1} - \\ - \mu_2 2^{\lambda_m - \lambda_2} - \ldots - \mu_{m-1} 2^{\lambda_m - \lambda_{m-1}}.$$

Рассмотрим слова длины λ_1 в алфавите B. Поскольку $\mu_1 \leqslant 2^{\lambda_1}$, из этих слов можно выбрать μ_1 различных слов $\beta_1, \ldots, \beta_{\mu_1}$ длины λ_1. Исключим из дальнейшего рассмотрения все слова из B^*, начинающиеся со слов $\beta_1, \ldots, \beta_{\mu_1}$. Далее рассмотрим множество слов в алфавите B длиной λ_2 и не начинающихся со слов $\beta_1, \ldots, \beta_{\mu_1}$. Таких слов будет $2^{\lambda_2} - \mu_1 2^{\lambda_2 - \lambda_1}$. Но $\mu_2 \leqslant 2^{\lambda_2} - \mu_1 2^{\lambda_2 - \lambda_1}$, значит, можно выбрать μ_2 различных слов. Обозначим их $\beta_{\mu_1+1}, \ldots, \beta_{\mu_1+\mu_2}$. Исключим слова, начинающиеся с $\beta_{\mu_1+1}, \ldots, \beta_{\mu_1+\mu_2}$, из дальнейшего рассмотрения. И далее, используя неравенства для частичных сумм, мы будем на i-м шаге выбирать μ_i слов длины λ_i, $\beta_{\mu_1+\mu_2+\ldots+\mu_{i-1}}, \ldots, \beta_{\mu_1+\mu_2+\ldots+\mu_{i-1}+\mu_i}$, причём эти слова не будут начинаться с тех слов, которые были выбраны раньше. В то же время длины этих слов всё время растут (так как $\lambda_1 < \lambda_2 < \ldots < \lambda_m$), поэтому они не могут быть префиксами тех слов, которые выбраны раньше. Итак, в конце имеем набор из n слов $\beta_1, \ldots, \beta_{\mu_1+\ldots+\mu_m} = \beta_n$, $|\beta_1| = l_1, \ldots, |\beta_n| = l_n$, коды $\beta_1, \ldots, \beta_n$ не являются префиксами друг друга, а значит, схема $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ будет префиксной и, по теореме предыдущего подраздела, разделимой. □

Пример *Азбука Морзе*[1] — это схема алфавитного кодирования

$\langle A \to 01$, $B \to 1000$, $C \to 1010$, $D \to 100$, $E \to 0$, $F \to 0010$, $G \to 110$,
$H \to 0000$, $I \to 00$, $J \to 0111$, $K \to 101$, $L \to 0100$, $M \to 11$, $N \to 10$,
$O \to 111$, $P \to 0110$, $Q \to 1101$, $R \to 010$, $S \to 000$, $T \to 1$, $U \to 001$,
$V \to 0001$, $W \to 011$, $X \to 1001$, $Y \to 1011$, $Z \to 1100\rangle$,

где по историческим и техническим причинам 0 называется *точкой* и обозначается знаком «·», а 1 называется *тире* и обозначается знаком «–». Имеем:

[1] Самуэль Морзе (1791–1872).

$$
\begin{array}{lllllll}
1/4 + & 1/16 + & 1/16 + & 1/8 + & 1/2 + & 1/16 + & 1/8 + \\
+\ 1/16 + & 1/4 + & 1/16 + & 1/8 + & 1/16 + & 1/4 + & 1/4 + \\
+\ 1/8 + & 1/16 + & 1/16 + & 1/8 + & 1/8 + & 1/2 + & 1/8 + \\
+\ 1/16 + & 1/8 + & 1/16 + & 1/16 + & 1/16 = & & \\
\end{array}
$$

$= 2/2 + 4/4 + 7/8 + 12/16 = 3{+}5/8 > 1.$

Таким образом, неравенство Макмиллана для азбуки Морзе не выполнено, и эта схема не является разделимой. На самом деле в азбуке Морзе имеются дополнительные элементы — паузы между буквами (и словами), которые позволяют декодировать сообщения. Эти дополнительные элементы определены неформально, поэтому приём и передача сообщений с помощью азбуки Морзе, особенно с высокой скоростью, являлись некоторым искусством, а не простой технической процедурой.

СЛЕДСТВИЕ *Если схема алфавитного кодирования* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *разделима, то существует префиксная схема* $\sigma' = \langle a_i \to \beta_i' \rangle_{i=1}^n$, *причём* $\forall\, i\ (|\beta_i| = |\beta_i'|)$.

Пример Схема $\langle a \to 0, b \to 01 \rangle$ — разделимая, но не префиксная, а схема $\langle a \to 0, b \to 10 \rangle$ — префиксная (и разделимая).

6.2. Кодирование с минимальной избыточностью

Для практики важно, чтобы коды сообщений имели по возможности наименьшую длину. Алфавитное кодирование пригодно для любых сообщений, то есть для $S = A^*$. Если больше про множество S ничего не известно, то точно сформулировать задачу оптимизации затруднительно. Однако на практике часто доступна дополнительная информация. Например, для текстов на естественных языках известно распределение вероятности появления букв в сообщении. Использование такой информации позволяет строго поставить и решить задачу построения оптимального алфавитного кодирования.

6.2.1. Минимизация длины кода сообщения

Если задана разделимая схема алфавитного кодирования $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$, то любая схема $\sigma' = \langle a_i \to \beta_i{}' \rangle_{i=1}^n$, где последовательность $\langle \beta'_1, \ldots, \beta'_n \rangle$ является перестановкой последовательности $\langle \beta_1, \ldots, \beta_n \rangle$, также будет разделимой. Если длины элементарных кодов равны, то перестановка элементарных кодов в схеме не влияет на длину кода сообщения. Но если длины элементарных кодов различны, то длина кода сообщения зависит от состава букв в сообщении и от того, какие элементарные коды каким буквам назначены. Если заданы конкретное сообщение и конкретная схема кодирования, то нетрудно подобрать такую перестановку элементарных кодов, при которой длина кода сообщения будет минимальна. Пусть $k_1, \ldots, k_n$ — количества вхождений букв $a_1, \ldots, a_n$ в сообщение

$s \in S$, а $l_1, \ldots, l_n$ — длины элементарных кодов $\beta_1, \ldots, \beta_n$ соответственно. Тогда, если $k_i \leqslant k_j$ и $l_i \geqslant l_j$, то $k_i l_i + k_j l_j \leqslant k_i l_j + k_j l_i$. Действительно, пусть $k_j = k + a$, $k_i = k$ и $l_j = l$, $l_i = l + b$, где $a, b \geqslant 0$. Тогда

$$(k_i l_j + k_j l_i) - (k_i l_i + k_j l_j) = (kl + (k+a)(l+b)) - (k(l+b) + l(k+a)) =$$
$$= (kl + al + bk + ab + kl) - (kl + al + kl + bk) = ab \geqslant 0.$$

Отсюда вытекает алгоритм назначения элементарных кодов, при котором длина кода конкретного сообщения $s \in S$ будет минимальна: нужно отсортировать буквы сообщения s в порядке убывания количества вхождений, элементарные коды отсортировать в порядке возрастания длины и назначить коды буквам в этом порядке.

ЗАМЕЧАНИЕ

Этот простой метод решает задачу минимизации длины кода только для фиксированного сообщения $s \in S$ и фиксированной схемы σ.

6.2.2. Цена кодирования

Пусть заданы алфавит $A = \{a_1, \ldots, a_n\}$ и вероятности появления букв в сообщении $P = \langle p_1, \ldots, p_n \rangle$ (p_i — вероятность появления буквы a_i). Не ограничивая общности, можно считать, что $p_i + \ldots + p_n = 1$ и $p_1 \geqslant \ldots \geqslant p_n > 0$ (то есть можно сразу исключить буквы, которые не могут появиться в сообщении, и упорядочить буквы по убыванию вероятности их появления). Для каждой (разделимой) схемы $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ алфавитного кодирования математическое ожидание коэффициента увеличения длины сообщения при кодировании σ (обозначается l_σ) определяется следующим образом:

$$l_\sigma(P) \stackrel{\text{Def}}{=} \sum_{i=1}^{n} p_i l_i, \qquad \text{где} \quad l_i \stackrel{\text{Def}}{=} |\beta_i|,$$

и называется средней *ценой* (или *длиной*) кодирования σ при распределении вероятностей P.

Пример Для разделимой схемы $A = \{a, b\}$, $B = \{0, 1\}$, $\sigma = \{a \to 0, b \to 01\}$ при распределении вероятностей $\langle 0,5; 0,5 \rangle$ цена кодирования составляет $0,5 * 1 + + 0,5 * 2 = 1,5$, а при распределении вероятностей $\langle 0,9; 0,1 \rangle$ она равна $0,9 * 1 + + 0,1 * 2 = 1,1$.

Введём обозначения:

$$l_*(P) \stackrel{\text{Def}}{=} \inf_\sigma l_\sigma(P), \qquad p_* \stackrel{\text{Def}}{=} \min_{i-1}^{n} p_i, \qquad L \stackrel{\text{Def}}{=} \lfloor \log_2(n-1) \rfloor + 1.$$

Очевидно, что всегда существует разделимая схема $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$, такая, что $\forall i \ (|\beta_i|) = L$. Такая схема называется схемой *равномерного* кодирования. Следовательно, $1 \leqslant l_*(P) \leqslant L$, и достаточно учитывать только такие схемы, для которых $\forall i \ (p_i l_i) \leqslant L$, где l_i — целое и $l_i \leqslant L/p_*$. Таким образом, имеется лишь конечное число схем σ, для которых $l_*(P) \leqslant l_\sigma(P) \leqslant L$. Значит, существует схема σ_*, на которой инфимум достигается: $l_{\sigma_*}(P) = l_*(P)$.

Алфавитное (разделимое) кодирование σ_*, для которого $l_{\sigma_*}(P) = l_*(P)$, называется кодированием с *минимальной избыточностью*, или *оптимальным* кодированием, для распределения вероятностей P.

6.2.3. Алгоритм Фано

Рекурсивный алгоритм Фано[1] строит разделимую префиксную схему алфавитного кодирования, близкого к оптимальному.

Алгоритм 6.1 Построение кодирования, близкого к оптимальному

Вход: P : **array** $[1..n]$ **of real** — массив вероятностей появления букв в сообщении, упорядоченный по невозрастанию; $1 \geqslant P[1] \geqslant \ldots \geqslant P[n] > 0$, $P[1] + \ldots + P[n] = 1$.
Выход: C : **array** $[1..n, 1..L]$ **of** $0..1$ — массив элементарных кодов.
Fano$(1, n, 0)$ { вызов рекурсивной процедуры Fano }

Основная работа по построению элементарных кодов выполняется следующей рекурсивной процедурой Fano.

Вход: b — индекс начала обрабатываемой части массива P, e — индекс конца обрабатываемой части массива P, k — длина уже построенных кодов в обрабатываемой части массива C.
Выход: заполненный массив C.

```
if e > b then
  k := k + 1 { место для очередного разряда в коде }
  m := Med(b, e) { деление массива на две части }
  for i from b to e do
    C[i, k] := i > m { в первой части добавляем 0, во второй — 1 }
  end for
  Fano(b, m, k) { обработка первой части }
  Fano(m + 1, e, k) { обработка второй части }
end if
```

Функция Med находит *медиану* указанной части массива $P[b..e]$, то есть определяет такой индекс m $(b \leqslant m < e)$, что сумма элементов $P[b..m]$ наиболее близка к сумме элементов $P[(m+1)..e]$.

Вход: b — индекс начала обрабатываемой части массива P, e — индекс конца обрабатываемой части массива P.
Выход: m — индекс медианы, то есть $\min\limits_{m \in b..(e-1)} \left| \sum\limits_{i=b}^{m} P[i] - \sum\limits_{i=m+1}^{e} P[i] \right|$.

```
S_b := 0 { сумма элементов первой части }
for i from b to e − 1 do
  S_b := S_b + P[i] { вначале все, кроме последнего }
end for
S_e := P[e] { сумма элементов второй части }
m := e { начинаем искать медиану с конца }
```

[1] Роберт М. Фано (р. 1917).

repeat
  $|d := S_b - S_e|$ { разность сумм первой и второй частей }
  $m := m - 1$ { сдвигаем границу медианы вниз }
  $S_b := S_b - P[m]; S_e := S_e + P[m]$ { перевычисляем суммы }
until $|S_b - S_e| \geqslant d$
return m

Обоснование При каждом удлинении кодов в одной части коды удлиняются нулями, а в другой — единицами. Таким образом, коды одной части не могут быть префиксами другой. Удлинение кода заканчивается тогда и только тогда, когда длина части равна 1, то есть остается единственный код. Таким образом, схема по построению префиксная, а потому разделимая. □

Пример Коды, построенные алгоритмом Фано для заданного распределения вероятностей ($n = 7$).

p_i	$C[i]$	l_i	$p_i l_i$
0,20	00	2	0,40
0,20	010	3	0,60
0,19	011	3	0,57
0,12	100	3	0,36
0,11	101	3	0,33
0,09	110	3	0,27
0,09	111	3	0,27
$l_\sigma(P)$			2,80

6.2.4. Оптимальное кодирование

Оптимальные схемы алфавитного кодирования обладают определёнными структурными свойствами, устанавливаемыми в следующих леммах. Нарушение этих свойств для какой-либо схемы означает, что схема не оптимальна.

ЛЕММА 1 *Пусть* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *— схема оптимального кодирования для распределения вероятностей* $P = p_1 \geqslant \ldots \geqslant p_n > 0$. *Тогда если* $p_i > p_j$, *то* $l_i \leqslant l_j$.

Доказательство От противного. Пусть $i < j$, $p_i > p_j$ и $l_i > l_j$. Тогда рассмотрим схему $\sigma' = \{a_1 \to \beta_1, \ldots, a_i \to \beta_j, \ldots, a_j \to \beta_i, \ldots, a_n \to \beta_n\}$. Имеем: $l_\sigma - l_{\sigma'} = (p_i l_i + p_j l_j) - (p_i l_j + p_j l_i) = (p_i - p_j)(l_i - l_j) > 0$, что противоречит тому, что схема σ — оптимальна. □

Таким образом, не ограничивая общности, можно считать, что $l_1 \leqslant \ldots \leqslant l_n$.

ЗАМЕЧАНИЕ

Фактически, лемма 1 повторяет для вероятностей то, что было обнаружено для частот в подразделе 6.2.1.

ЛЕММА 2 *Если $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \ldots \geqslant p_n > 0$, то среди элементарных кодов имеются два кода максимальной длины, которые различаются только в последнем разряде.*

Доказательство По предыдущей лемме кодовые слова максимальной длины являются последними в схеме. Далее от противного.

[1] Пусть кодовое слово максимальной длины одно и имеет вид $\beta_n = \beta b$, где $b = 0 \vee b = 1$. Имеем: $\forall i \in 1..n-1 \ (l_i \leqslant |\beta|)$. Так как схема префиксная, то слова $\beta_1, \ldots, \beta_{n-1}$ не являются префиксами β. С другой стороны, β не является префиксом слов $\beta_1, \ldots, \beta_{n-1}$, иначе было бы $\beta = \beta_j$, а значит, β_j было бы префиксом β_n. Тогда схема $\sigma' := \langle a_1 \to \beta_1, \ldots, a_n \to \beta \rangle$ тоже префиксная, причём $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ.

[2] Пусть теперь два кодовых слова β_{n-1} и β_n максимальной длины различаются не в последнем разряде, то есть $\beta_{n-1} = \beta' b'$, $\beta_n = \beta'' b''$, $\beta' \neq \beta''$, причём β', β'' не являются префиксами для $\beta_1, \ldots, \beta_{n-2}$ и наоборот. Тогда схема $\sigma' := \langle a_1 \to \beta_1, \ldots, a_{n-2} \to \beta_{n-2}, a_{n-1} \to \beta' b', a_n \to \beta'' \rangle$ также является префиксной, причём $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ.

[3] Пусть кодовых слов максимальной длины больше двух. Тогда среди них найдутся два, которые отличаются не только в последнем разряде, что противоречит уже доказанному пункту 2. □

ТЕОРЕМА *Если $\sigma_{n-1} = \langle a_i \to \beta_i \rangle_{i=1}^{n-1}$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \ldots \geqslant p_{n-1} > 0$ и $p_j = q' + q''$, причём*

$$p_1 \geqslant \ldots \geqslant p_{j-1} \geqslant p_{j+1} \geqslant \ldots \geqslant p_{n-1} \geqslant q' \geqslant q'' > 0,$$

то кодирование со схемой

$$\sigma_n = \langle a_1 \to \beta_1, \ldots, a_{j-1} \to \beta_{j-1}, a_{j+1} \to \beta_{j+1}, \ldots, a_{n-1} \to \beta_{n-1}, a_j \to \beta_j 0, a_n \to \beta_j 1 \rangle$$

является оптимальным префиксным кодированием для распределения вероятностей $P_n = p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_{n-1}, q', q''$.

Доказательство Заметим следующее.

[1] Если σ_{n-1} было префиксным, то σ_n тоже будет префиксным по построению.

[2] Цена кодирования для схемы σ_n больше цены кодирования для схемы σ_{n-1} на величину p_j:

$$\begin{aligned} l_{\sigma_n}(P_n) &= p_1 l_1 + p_2 l_2 + \ldots + p_{j-1} l_{j-1} + p_{j+1} l_{j+1} + \ldots + \\ &+ p_{n-1} l_{n-1} + q'(l_j + 1) + q''(l_j + 1) = \\ &= p_1 l_1 + \ldots + p_{n-1} l_{n-1} + l_j(q' + q'') + (q' + q'') = \\ &= p_1 l_1 + \ldots + p_{n-1} l_{n-1} + l_j p_j + p_j = l_{\sigma_{n-1}}(P_{n-1}) + p_j. \end{aligned}$$

[3] Пусть некоторая схема $\sigma'_n := \{a_i \to \beta_i\}_{i=1}^n$ оптимальна для P_n. Тогда по лемме 2 два кода максимальной длины различаются в последнем разряде: $\sigma'_n = \langle a_1 \to \beta'_1, \ldots, a_{n-2} \to \beta'_{n-2}, a_{n-1} \to \beta 0, a_n \to \beta 1\rangle$. Положим $l' := |\beta|$ и рассмотрим схему $\sigma'_{n-1} := \langle a_1 \to \beta'_1, \ldots, a_j \to \beta, \ldots, a_{n-2} \to \beta'_{n-2}\rangle$, где j определено так, чтобы $p_{j-1} \geqslant q' + q'' \geqslant p_{j+1}$. Цена кодирования для схемы σ'_n больше цены кодирования для схемы σ'_{n-1} также на величину p_j:

$$\begin{aligned} l_{\sigma'_n}(P_n) &= l_1 p_1 + \ldots + l_{n-2}p_{n-2} + (l'+1)q' + (l'+1)q'' = \\ &= l_1 p_1 + \ldots + l_{n-2}p_{n-2} + l'(q'+q'') + (q'+q'') = \\ &= l_{\sigma'_{n-1}}(P_{n-1}) + p_j. \end{aligned}$$

[4] Схема σ'_n — префиксная, значит, схема σ'_{n-1} — тоже префиксная.

[5] Схема σ_{n-1} — оптимальна, значит, $l_{\sigma'_{n-1}}(P_{n-1}) \geqslant l_{\sigma_{n-1}}(P_{n-1})$.

[6] Имеем: $l_{\sigma_n}(P_n) = l_{\sigma_{n-1}}(P_{n-1}) + p_j \leqslant l_{\sigma'_{n-1}}(P_{n-1}) + p_j = l_{\sigma'_n}(P_n)$. Но схема σ'_n оптимальна, значит, схема σ_n также оптимальна. □

6.2.5. Алгоритм Хаффмена

Рекурсивный алгоритм Хаффмена[1] строит схему оптимального префиксного алфавитного кодирования для заданного распределения вероятностей появления букв.

Алгоритм 6.2 Построение оптимальной схемы — рекурсивная процедура Huffman

Вход: n — количество букв, P : **array** $[1..n]$ **of real** — массив вероятностей букв, упорядоченный по убыванию.

Выход: C : **array** $[1..n, 1..L]$ **of** $0..1$ — массив элементарных кодов, ℓ : **array** $[1..n]$ **of** $1..L$ — массив длин элементарных кодов схемы оптимального префиксного кодирования.

```
if n = 2 then
  C[1,1] := 0; ℓ[1] := 1 { первый элемент }
  C[2,1] := 1; ℓ[2] := 1 { второй элемент }
else
  q := P[n − 1] + P[n] { сумма двух последних вероятностей }
  j := Up(n, q) { поиск места и вставка суммы }
  Huffman(P, n − 1) { рекурсивный вызов }
  Down(n, j) { достраивание кодов }
end if
```

Функция Up находит в массиве P место, в котором должно находиться число q (см. предыдущую теорему), и вставляет это число, сдвигая вниз остальные элементы.

[1] Дэвид А. Хаффмен (1925-1999).

Вход: n — длина обрабатываемой части массива P, q — вставляемая сумма.
Выход: изменённый массив P.

```
j := 1 { место вставляемого элемента }
for i from n − 1 downto 2 do
  if P[i − 1] < q then
    P[i] := P[i − 1] { сдвиг элемента массива }
  else
    j := i { определение места вставляемого элемента }
    exit for i { всё сделано — цикл не нужно продолжать }
  end if
end for
P[j] := q { запись вставляемого элемента }
return j
```

Процедура Down строит оптимальный код для n букв на основе построенного оптимального кода для $n-1$ буквы. Для этого код буквы с номером j временно исключается из массива C путем сдвига вверх кодов букв с номерами, большими j, а затем в конец обрабатываемой части массива C добавляется пара кодов, полученных из кода буквы с номером j удлинением на 0 и 1 соответственно. Здесь $C[i,*]$ означает вырезку из массива, то есть i-ю строку массива C.

Вход: n — длина обрабатываемой части массива P, j — номер «разделяемой» буквы.
Выход: оптимальные коды в первых n элементах массивов C и ℓ.

```
c := C[j,*] { запоминание кода буквы j }
l := ℓ[j] { и длины этого кода }
for i from j to n − 2 do
  C[i,*] := C[i + 1,*] { сдвиг кода }
  ℓ[i] := ℓ[i + 1] { и его длины }
end for
C[n − 1,*] := c; C[n,*] := c { копирование кода буквы j }
C[n − 1, l + 1] := 0; C[n, l + 1] := 1 { наращивание кодов }
ℓ[n − 1] := l + 1; ℓ[n] := l + 1 { и увеличение длин }
```

Обоснование Для пары букв при любом распределении вероятностей оптимальное кодирование очевидно: первой букве нужно назначить код 0, а второй — 1. Именно это и делается в первой части оператора **if** основной процедуры Huffman. Рекурсивная часть алгоритма в точности следует доказательству теоремы предыдущего подраздела. С помощью функции Up в исходном упорядоченном массиве P отбрасываются две последние (наименьшие) вероятности, и их сумма вставляется в массив P, так чтобы массив (на единицу меньшей длины) остался упорядоченным. Заметим, что при этом место вставки сохраняется в локальной переменной j. Так происходит до тех пор, пока не останется массив из двух элементов, для которого оптимальный код известен. После этого в обратном порядке строятся оптимальные коды для трёх, четырёх и т. д. элементов. Заметим, что при этом массив вероятностей P уже не нужен — нужна только последовательность номеров кодов, которые должны быть изъяты из массива кодов и продублированы в конце с добавлением разряда. А эта последовательность хранится в экземплярах локальной переменной j, соответствующих рекурсивным вызовам процедуры Huffman. □

Пример Построение оптимального кода Хаффмена для $n = 7$. В левой части таблицы показано изменение массива P, а в правой части — массива C. Позиция, соответствующая текущему значению переменной j, выделена полужирным начертанием шрифта.

0,20	0,20	**0,23**	**0,37**	**0,40**	**0,60**	**0**	**1**	**00**	**01**	10	10
0,20	0,20	0,20	0,23	0,37	0,40	1	00	01	10	11	11
0,19	0,19	0,20	0,20	0,23			01	10	11	000	000
0,12	**0,18**	0,19	0,20					11	000	**001**	010
0,11	0,12	0,18							001	010	011
0,09	0,11									011	0010
0,09											0011

Цена кодирования составляет

$$0,20 \times 2 + 0,20 \times 2 + 0,19 \times 3 + 0,12 \times 3 + 0,11 \times 3 + 0,09 \times 4 + 0,09 \times 4 = 2,78,$$

что несколько лучше, чем в кодировании, полученном алгоритмом Фано.

6.3. Помехоустойчивое кодирование

Надёжность электронных устройств по мере их совершенствования всё время возрастает, но, тем не менее, в их работе возможны ошибки, как систематические, так и случайные. Сигнал в канале связи может быть искажён помехой, поверхность магнитного носителя может быть повреждена, в разъёме может быть потерян контакт. Ошибки аппаратуры ведут к искажению или потере передаваемых или хранимых данных. При определённых условиях, некоторые из которых рассматриваются в этом разделе, можно применять методы кодирования, позволяющие правильно декодировать исходное сообщение, несмотря на ошибки в данных кода. В качестве исследуемой модели достаточно рассмотреть *канал связи с помехами*, потому что к этому случаю легко сводятся остальные. Например, запись на диск можно рассматривать как передачу данных в канал, а чтение с диска — как приём данных из канала.

6.3.1. Кодирование с исправлением ошибок

Пусть имеется канал связи, содержащий источник помех. Помехи проявляются в том, что принятое сообщение может отличаться от переданного. Опишем это отличие в функциональной форме:

$$K \xrightarrow{C} K', \quad K, K' \in B^*,$$

где K —множество переданных, а K' — соответствующее множество принятых по каналу сообщений, подразумевая, что разные вызовы «функции» C с одним и тем же аргументом могут возвращать различные результаты. Кодирование F (вместе с декодированием F^{-1}), обладающее тем свойством, что

$$S \xrightarrow{F} K \xrightarrow{C} K' \xrightarrow{F^{-1}} S,\ \forall s \in S \subset A^*\ \left(F^{-1}(C(F(s))) = s\right),$$

называется *помехоустойчивым*, или *самокорректирующимся*, или кодированием *с исправлением ошибок*. Без ограничения общности можно считать, что $A = B = \{0, 1\}$. Кроме того, естественно предположить, что содержательное кодирование (вычисление F и F^{-1}) выполняется на устройстве, свободном от помех, то есть F и F^{-1} являются функциями в обычном смысле.

ЗАМЕЧАНИЕ

Функция F^{-1} является декодированием для кодирования F, но она *не* является обратной функцией для функции F в смысле определений подраздела 1.6.2.

Если про источник помех C ничего не известно, то функции F и F^{-1} не могут быть определены, за исключением тривиальных случаев, когда $S = \varnothing$ или $|S| = 1$. Таким образом, для решения поставленной задачи необходимо иметь описание возможных ошибок (проявлений помех).

Ошибки в канале могут быть следующих типов:

- $0 \mapsto 1$, $1 \mapsto 0$ — ошибка типа замещения разряда;
- $0 \mapsto \varepsilon$, $1 \mapsto \varepsilon$ — ошибка типа выпадения разряда;
- $\varepsilon \mapsto 1$, $\varepsilon \mapsto 0$ — ошибка типа вставки разряда.

Канал характеризуется верхними оценками количества ошибок каждого типа, которые возможны при передаче через канал сообщения определённой длины n. Общая *характеристика* ошибок канала (то есть их количество и типы) обозначается $\delta = \langle \delta_1, \delta_2, \delta_3 \rangle$, где $\delta_1, \delta_2, \delta_3$ — верхние оценки количества ошибок каждого типа соответственно.

Пример Допустим, что имеется канал с характеристикой $\delta = \langle 1, 0, 0 \rangle$, то есть в канале возможно не более одной ошибки типа замещения разряда при передаче сообщения длины n. Пусть требуется передавать через этот канал поток сообщений, каждое длины n. Рассмотрим следующее кодирование: $F(a) := aaa$ (то есть каждый разряд в сообщении утраивается) и декодирование $F^{-1}(abc) := := a + b + c > 1$ (то есть разряд восстанавливается методом «голосования»). Это кодирование кажется помехоустойчивым для данного канала, однако на самом деле это не так. Дело в том, что хотя можно предполагать, что при передаче сообщения длины $3n$ возможно не более 3 ошибок типа замещения разряда, но места этих ошибок совершенно не обязательно распределены равномерно по всему сообщению. Ошибки замещения могут произойти в соседних разрядах, и метод голосования восстановит разряд неверно. Чтобы метод голосования сработал, вместо одного сообщения длины $3n$ придётся передать 3 сообщения длины n, то есть уменьшить втрое скорость передачи потока сообщений через канал.

6.3.2. Возможность исправления всех ошибок

Пусть E_s^δ — множество слов, которые могут быть получены из слова s в результате всех возможных комбинаций допустимых в канале ошибок типа δ, то есть $s \in S \subset A^*$, $E_s^\delta \subset B^*$. Если $s' \in E_s^\delta$, то кратчайшую последовательность ошибок, которая позволяет получить из слова s слово s', будем обозначать $E^\delta\langle s, s'\rangle$. Заметим, что таких последовательностей может быть несколько. Другими словами,

$$E_s^\delta = \left\{ s' \in B^* \mid \exists\, E^\delta\langle s, s'\rangle \right\}.$$

Количество ошибок в кратчайшей последовательности $E^\delta\langle s, s'\rangle$ обозначим $|E^\delta\langle s, s'\rangle|$. Если характеристика канала подразумевается, то индекс δ не указывается.

Пример Пусть $\delta = \langle 2, 1, 1\rangle$ для слов длины 2. Тогда $|E^\delta\langle 01, 10\rangle| = 2$, причём существует несколько различных кратчайших последовательностей ошибок, в частности: $01 \xrightarrow{0\mapsto 1} 11 \xrightarrow{1\mapsto 0} 10$; $01 \xrightarrow{0\mapsto \varepsilon} 1 \xrightarrow{\varepsilon\mapsto 0} 10$; $01 \xrightarrow{\varepsilon\mapsto 0} 010 \xrightarrow{0\mapsto \varepsilon} 10$.

ТЕОРЕМА *Чтобы существовало помехоустойчивое кодирование F с исправлением всех ошибок типа δ, необходимо и достаточно, чтобы* $\forall\, s_1, s_2 \in S\ \left(E_{s_1}^\delta \cap E_{s_2}^\delta = \varnothing\right)$.

Доказательство

[$\Longrightarrow$] Если кодирование помехоустойчивое, то $E_{s_1}^\delta \cap E_{s_2}^\delta = \varnothing$, иначе F^{-1} невозможно было бы определить как функцию.

[$\Longleftarrow$] По теореме 1.2.7 из условия $\forall\, s_1, s_2 \in S\ \left(E_{s_1}^\delta \cap E_{s_2}^\delta = \varnothing\right)$ следует, что существует разбиение $\mathcal{B} = \{B_s\}_{s\in S}$ множества B^*, причём $\forall\, s \in S\ \left(E_s^\delta \subset B_s\right)$. По разбиению $\mathcal{B}$ требуемая функция $F^{-1}\colon B^* \to S$ строится следующим образом: **if** $s' \in B_s$ **then** $F^{-1}(s') := s$ **end if**. □

Пример Рассмотрим канал, в котором в любом передаваемом разряде происходит ошибка типа замещения с вероятностью p, причём замещения различных разрядов статистически независимы. Такой канал называется *двоичным симметричным*. В этом случае любое слово $s \in E_2^n$ может быть преобразовано в любое другое слово $s' \in E_2^n$ замещениями разрядов. Таким образом, $\forall\, s\ \left(E_s = E_2^n\right)$, и исправить все ошибки в двоичном симметричном канале невозможно даже при сколь угодно малом $p > 0$.

Будем говорить, что F является кодированием с исправлением p ошибок типа δ, если существует декодирование F^{-1} такое, что

$$\forall\, s \in S\ \left(\forall\, s' \in E_{F(s)}^\delta\ \left(|E^\delta(F(s), s')| \leqslant p \Longrightarrow F^{-1}(s') = s\right)\right).$$

Пример Пусть $\delta = \langle 1, 0, 0\rangle$ для слов длины n. Метод голосования, рассмотренный в примере предыдущего подраздела, является кодированием с исправлением одной ошибки типа замещения, но не является кодированием с исправлением всех ошибок при $n > 1$.

6.3.3. Кодовое расстояние

Неотрицательная функция $d(x,y)\colon M \times M \to \mathbb{R}_+$ называется *расстоянием* (или *метрикой*) на множестве M, если выполнены следующие условия (аксиомы метрики):

1. $d(x,y) = 0 \iff x = y$.
2. $d(x,y) = d(y,x)$.
3. $d(x,y) \leqslant d(x,z) + d(y,z)$.

Множество $\{y \mid d(x,y) \leqslant p\}$ называется *шаром*, или *метрической окрестностью*, с центром в x и радиусом p.

Пусть некоторый канал имеет характеристику δ. Рассмотрим функцию

$$d_\delta(\beta',\beta'') \stackrel{\text{Def}}{=} \begin{cases} \min\limits_{\{E^\delta\langle\beta',\beta''\rangle\}} |E^\delta\langle\beta',\beta''\rangle|, & \text{если } \beta'' \in E^\delta_{\beta'}, \\ +\infty, & \text{если } \beta'' \notin E^\delta_{\beta'}. \end{cases}$$

Эта функция называется *расстоянием Хэмминга*[1].

ЗАМЕЧАНИЕ

Мы рассматриваем симметричные ошибки, то есть если в канале допустима ошибка $0 \mapsto 1$, то допустима и ошибка $1 \mapsto 0$.

ЛЕММА *Введенная функция d_δ является расстоянием.*

Доказательство

[1] $d_\delta(\beta',\beta'') = 0 \iff \beta' = \beta''$, поскольку длина кратчайшей последовательности преобразований равна нулю тогда и только тогда, когда слова совпадают.

[2] $d_\delta(\beta',\beta'') = d_\delta(\beta'',\beta')$, поскольку ошибки симметричны, и из последовательности $E^\delta\langle\beta',\beta''\rangle$ можно получить последовательность $E^\delta\langle\beta'',\beta'\rangle$, применяя обратные ошибки в обратном порядке.

[3] $d_\delta(\beta',\beta'') \leqslant d_\delta(\beta',\beta''') + d_\delta(\beta'',\beta''')$, поскольку конкатенация последовательностей $E^\delta\langle\beta',\beta'''\rangle$ и $E^\delta\langle\beta''',\beta''\rangle$ является *некоторой* последовательностью, преобразующей β' в β'', а $d_\delta(\beta',\beta'')$ является длиной кратчайшей из таких последовательностей. □

Пример Расстояние Хэмминга в E_2^n. Пусть $\delta = \langle n,0,0\rangle$, то есть допускаются только ошибки типа замещения разрядов. Рассмотрим слова $\beta' = b'_1 \ldots b'_n$ и $\beta'' = b''_1 \ldots b''_n$. Тогда $d(\beta',\beta'') \stackrel{\text{Def}}{=} \sum\limits_{i=1}^{n} (b'_i \neq b''_i)$. То есть расстоянием Хэмминга между битовыми шкалами одной длины является число несовпадающих разрядов.

[1] Ричард Весли Хэмминг (1915–1998).

ЗАМЕЧАНИЕ

В последней формуле, так же как и в других местах этой книги, используется неявное приведение **false** $\mapsto 0$, **true** $\mapsto 1$.

Пусть $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ — схема алфавитного кодирования, а d — некоторая метрика на B^*. Тогда минимальное расстояние между элементарными кодами

$$d(\sigma) \stackrel{\text{Def}}{=} \min_{1 \leqslant i < j \leqslant n} d(\beta_i, \beta_j)$$

называется *кодовым расстоянием* схемы σ в метрике d.

ТЕОРЕМА *Алфавитное кодирование со схемой* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ *и с кодовым расстоянием Хэмминга* $d(\sigma) = \min\limits_{\beta', \beta'' \in V} d_\delta(\beta', \beta'')$ *является кодированием с исправлением* p *ошибок типа* δ *тогда и только тогда, когда* $d(\sigma) > 2p$.

Доказательство Если можно исправить ошибки в каждом из кодовых слов, то можно исправить ошибки и во всем сообщении. Поэтому достаточно рассматривать только кодовые слова. Заметим, что множество E_x^δ для канала, который допускает не более p ошибок типа δ,— это шар радиуса p с центром в x. Таким образом, $E_{\beta'}^\delta \cap E_{\beta''}^\delta = \varnothing \iff d(\sigma) > 2p$. По теореме 6.3.2 условие $E_{\beta'}^\delta \cap E_{\beta''}^\delta = \varnothing$ равносильно существованию требуемого декодирования. □

Пример Рассмотрим схему равномерного кодирования $\sigma = \langle a_i \to \beta_i \rangle_n^{i=1}$, где $\forall\, i \ \ (|\beta_i| = m)$, $m \geqslant \lfloor \log_2(n-1) \rfloor + 1$. Такая схема может исправлять h ошибок типа замещения в m разрядах, если и только если

$$\min_{1 \leqslant i < j \leqslant n} \left(\sum_{k=1}^{m} \beta_i[k] \neq \beta_j[k] \right) > 2h,$$

то есть если и только если любые два кодовых слова различаются не менее чем в $2h$ разрядах.

6.3.4. Код Хэмминга для исправления одного замещения

Очевидно, что для помехоустойчивости вместе с основным сообщением нужно передавать какую-то дополнительную информацию, с помощью которой в случае ошибок можно восстановить исходное сообщение. При этом нельзя считать, что дополнительная информация не подвержена ошибкам. Задача состоит в том, чтобы определить, какую дополнительную информацию следует передавать, чтобы её объём был минимальным, а процедуры кодирования и декодирования по возможности простыми.

Рассмотрим построение *кода Хэмминга*, который позволяет исправлять одиночные ошибки типа замещения. Пусть сообщение $\alpha = a_1 \ldots a_m$ кодируется словом $\beta = b_1 \ldots b_n$, $n > m$. Обозначим $k := n - m$. Пусть канал допускает не более одной ошибки типа замещения в слове длины n.

ОТСТУПЛЕНИЕ

Рассматриваемый случай простейший, но одновременно практически очень важный. Таким свойством, как правило, обладают внутренние шины передачи данных в современных компьютерах. При этом часто применяют аппаратно реализуемый метод повышения надёжности, который называется *контролем чётности*. Суть метода заключается в том, что к шине добавляется дополнительный разряд, значение которого определяется при передаче как двоичная сумма остальных разрядов. Таким образом, общее количество единиц, передаваемых по шине, всегда чётно. Если при приёме переданное количество единиц оказывается нечётным, диагностируется произошедшая ошибка. Контроль чётности позволяет обнаружить одиночную ошибку типа замещения разряда, но не позволяет её исправить.

При заданном n количество дополнительных разрядов k подбирается таким образом, чтобы $2^k \geqslant n+1$. Имеем:

$$2^k \geqslant n+1 \Longrightarrow \frac{2^n}{n+1} \geqslant 2^{n-k} \Longrightarrow \frac{2^n}{n+1} \geqslant 2^m.$$

Пример Для сообщения длиной $m = 32$ достаточно $k = 6$ дополнительных разрядов, поскольку $64 = 2^6 > 32 + 6 + 1 = 39$.

Определим последовательности натуральных чисел в соответствии с их представлениями в двоичной системе счисления следующим образом:
$V_1 := 1, 3, 5, 7, 9, 11, \ldots$ — все числа, у которых разряд 1 равен 1;
$V_2 := 2, 3, 6, 7, 10, \ldots$ — все числа, у которых разряд 2 равен 1;
$V_3 := 4, 5, 6, 7, 12, \ldots$ — все числа, у которых разряд 3 равен 1,
и т. д. По определению последовательность V_k начинается с числа 2^{k-1}. Также рассмотрим последовательности V'_k, где V'_k получается из V_k отбрасыванием первого элемента. Рассмотрим в слове $b_1 \ldots b_n$ k разрядов с номерами $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $\ldots 2^{k-1}$. Эти разряды назовём *контрольными*. Остальные разряды, а их ровно m, назовём *информационными*. Поместим в информационные разряды все разряды слова $a_1 \ldots a_m$ как они есть. Контрольные разряды определим следующим образом:
$b_1 := b_3 + b_5 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_1;
$b_2 := b_3 + b_6 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_2;
$b_4 := b_5 + b_6 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_3;
и вообще, $b_{2^{j-1}} := \sum\limits_{i \in V'_j} b_i$, причём сложение выполняется по модулю 2. Пусть после прохождения через канал получен код $c_1 \ldots c_n$, то есть $c_1 \ldots c_n := C(b_1 \ldots b_n)$, причём

$$\exists I \ \left((c_I = b_I \vee c_I = \overline{b_I}) \ \& \ \forall i \neq I \ (c_i = b_i) \right).$$

Здесь I — номер разряда, в котором, возможно, произошла ошибка замещения. Пусть это число имеет следующее двоичное представление: $I = i_l \ldots i_1$. Определим число $J = j_l \ldots j_1$ следующим образом:
$j_1 := c_1 + c_3 + c_5 + c_7 + \ldots$, то есть сумма всех разрядов с номерами из V_1;
$j_2 := c_2 + c_3 + c_6 + c_7 + \ldots$, то есть сумма всех разрядов с номерами из V_2;
$j_3 := c_4 + c_5 + c_6 + c_7 + \ldots$, то есть сумма всех разрядов с номерами из V_3;
и вообще, $j_p := \sum\limits_{q \in V_p} c_q$, где сложение выполняется по модулю 2.

ТЕОРЕМА *В указанных обозначениях $I = J$.*

Доказательство Эти числа равны, потому что поразрядно равны их двоичные представления. Действительно, пусть $i_1 = 0$. Тогда $I \notin V_1$, и значит,

$$j_1 = c_1 + c_3 + c_5 + \ldots = b_1 + b_3 + b_5 + \ldots = 0$$

по определению разряда b_1. Пусть теперь $i_1 = 1$. Тогда $I \in V_1$, и значит,

$$j_1 = c_1 + c_3 + c_5 + \ldots = b_1 + b_3 + b_5 + \ldots + \overline{b_x} + \ldots = 1,$$

так как если в сумме по модулю 2 изменить ровно один разряд, то изменится и значение всей суммы. Итак, $i_1 = j_1$. Аналогично (используя последовательности V_2 и т. д.) имеем $i_2 = j_2$ и т. д. Таким образом, $I = J$. □

Отсюда вытекает метод декодирования с исправлением ошибки: нужно вычислить число J. Если $J = 0$, то ошибки нет, иначе $c_J := \overline{c_J}$. После этого из исправленного сообщения извлекаются информационные разряды, которые уже не содержат ошибок.

ЗАМЕЧАНИЕ

Универсальный метод построения наилучшего помехоустойчивого кодирования для произвольного m и произвольного типа канала δ неизвестен. Однако для большинства встречающихся на практике комбинаций m и δ задача построения кодирования с исправлением всех ошибок решена.

6.4. Сжатие данных

Материал раздела 6.2 показывает, что при кодировании наблюдается некоторый баланс между временем и памятью. Затрачивая дополнительные усилия при кодировании и декодировании, можно сэкономить память, и наоборот, пренебрегая оптимальным использованием памяти, можно существенно выиграть во времени кодирования и декодирования. Конечно, этот баланс имеет место только в определённых пределах, и нельзя сократить расход памяти до нуля или построить мгновенно работающие алгоритмы кодирования. Для алфавитного кодирования пределы возможного установлены в разделе 6.2. Для достижения дальнейшего прогресса нужно рассмотреть неалфавитное кодирование.

6.4.1. Сжатие текстов

Допустим, что имеется некоторое сообщение, которое закодировано каким-то общепринятым способом (для текстов это, например, код ASCII) и хранится в памяти компьютера. Заметим, что равномерное кодирование (в частности, ASCII) не является для текстов оптимальным. Действительно, в текстах обычно используется существенно меньше, чем 256 символов (в зависимости от языка — примерно 60–80 с учётом знаков препинания, цифр, строчных и прописных букв). Кроме того, вероятности появления букв различны, и для каждого естественного языка известны (с некоторой точностью) частоты появления букв в тексте. Таким образом, можно задаться некоторым набором букв и частотами их

появления в тексте и с помощью алгоритма Хаффмена построить оптимальное алфавитное кодирование текстов (для заданного алфавита и языка). Простые расчёты показывают, что такое кодирование для распространенных естественных языков будет иметь цену кодирования несколько меньше 6, то есть даст выигрыш по сравнению с кодом ASCII на 25% или несколько больше.

ЗАМЕЧАНИЕ

Методы кодирования, позволяющие построить (без потери информации!) коды сообщений меньшей длины по сравнению с исходным сообщением, называются методами *сжатия* (или *упаковки*) данных. Качество сжатия определяется *коэффициентом сжатия*, который обычно измеряется в процентах и показывает, на сколько процентов кодированное сообщение короче исходного.

Известно, что практические программы сжатия (rar, zip и др.) имеют гораздо лучшие показатели, чем код Хаффмена: при сжатии текстовых файлов коэффициент сжатия достигает 70% и более. Это означает, что в таких программах используется *не* алфавитное кодирование.

6.4.2. Предварительное построение словаря

Рассмотрим следующий способ кодирования:

1. Исходное сообщение по некоторому алгоритму разбивается на последовательности букв, называемые *словами* (слово может иметь одно или несколько вхождений в исходный текст сообщения).
2. Полученное множество слов считается буквами нового алфавита. Для этого алфавита строится разделимая схема алфавитного кодирования (равномерного кодирования или оптимального кодирования, если для каждого слова подсчитать число вхождений в текст). Полученная схема обычно называется *словарем*, так как она сопоставляет слову код.
3. Далее код сообщения строится как пара — код словаря и последовательность кодов слов из данного словаря.
4. При декодировании исходное сообщение восстанавливается путем замены кодов слов на слова из словаря.

Пример Допустим, что требуется кодировать тексты на русском языке. В качестве алгоритма деления на слова примем естественные правила языка: слова отделяются друг от друга пробелами или знаками препинания. Можно принять допущение, что в каждом конкретном тексте имеется не более 2^{16} различных слов (обычно гораздо меньше). Таким образом, каждому слову можно сопоставить номер — целое число из двух байтов (равномерное кодирование). Поскольку в среднем слова русского языка состоят более чем из двух букв, такое кодирование даёт существенное сжатие текста (около 75% для обычных текстов на русском языке). Если текст достаточно велик (сотни тысяч или миллионы букв, то есть сотни и тысячи страниц), то дополнительные затраты на хранение словаря оказываются сравнительно небольшими.

ЗАМЕЧАНИЕ

Данный метод попутно позволяет решить задачу *полнотекстового поиска*, то есть определить, содержится ли заданное слово (или слова) в данном тексте, причём для этого не нужно просматривать весь текст (достаточно просмотреть только словарь).

Указанный метод можно усовершенствовать следующим образом. На шаге 2 следует применить алгоритм Хаффмена для построения оптимального кода, а на шаге 1 — решить экстремальную задачу разбиения текста на слова таким образом, чтобы среди всех возможных разбиений выбрать то, которое даёт наименьшую цену кодирования на шаге 2. Такое кодирование будет «абсолютно» оптимальным. К сожалению, указанная экстремальная задача очень трудоёмка, поэтому на практике не используется — время на предварительную обработку большого текста оказывается чрезмерно велико даже при использовании современных быстродействующих компьютеров.

6.4.3. Алгоритм Лемпела–Зива

На практике используется следующая идея, которая известна также как *адаптивное сжатие*. За один проход по тексту одновременно динамически строится словарь и кодируется текст. При этом нет необходимости хранить словарь: за счёт того, что при декодировании используется тот же самый алгоритм построения словаря, словарь динамически восстанавливается.

Ниже приведена простейшая реализация этой идеи, известная как *алгоритм Лемпела*[1]*–Зива*[2]. Вначале словарь D : **array** [**int**] **of string** содержит пустое слово, имеющее код 0. Далее в тексте последовательно выделяются слова. Выделяемое слово — это максимально длинное слово из уже имеющихся в словаре плюс ещё один символ. В сжатое представление записываются найденный код слова и расширяющая буква, а словарь пополняется расширенной комбинацией.

ЗАМЕЧАНИЕ

В алгоритмах этого раздела знак +, применённый к целым числам, означает сложение, а применённый к словам или символам — конкатенацию.

Алгоритм 6.3 Упаковка по методу Лемпела–Зива

Вход: исходный текст, заданный массивом кодов символов f : **array** $[1..n]$ **of char**.

Выход: сжатый текст, представленный последовательностью пар $\langle p, q\rangle$, где p — номер слова в словаре, q — код дополняющей буквы.

$D[0] := \varepsilon; d := 0$ { начальное состояние словаря }
$k := 1$ { номер текущей буквы в исходном тексте }
while $k \leqslant n$ **do**
  $p := \mathrm{FD}(k)$ { p — индекс найденного слова в словаре }

[1] Абрахам Лемпел (р. 1936).
[2] Якоб Зив (р. 1931).

```
    l := Length(D[p]) { l — длина найденного слова в словаре }
    yield ⟨p, f[k + l]⟩ { код найденного слова и ещё одна буква }
    d := d + 1; D[d] := D[p] + f[k + l] { пополнение словаря }
    k := k + l + 1 { продвижение вперед по исходному тексту }
  end while
```

Слово в словаре ищется с помощью несложной функции FD.

Вход: k — номер символа в исходном тексте, начиная с которого нужно искать в тексте слова из словаря.

Выход: p — индекс самого длинного слова в словаре, совпадающего с символами $f[k]..f[k+l]$. Если такого слова в словаре нет, то $p = 0$.

```
l := 0; p := 0 { начальное состояние }
for i from 1 to d do
  m := Length(D[i]) { длина слова в словаре }
  if D[i] = f[k..k + m − 1] & m > l then
    p := i; l := m { нашли более подходящее слово }
  end if
end for
return p
```

Распаковка осуществляется следующим алгоритмом.

Алгоритм 6.4 Распаковка по методу Лемпела–Зива

Вход: сжатый текст, представленный массивом пар g : **array** $[1..m]$ **of record** p : **int**; q : **char end record**, где p — номер слова в словаре, q — код дополняющей буквы.

Выход: исходный текст, заданный последовательностью строк и символов.

```
D[0] := ε; d := 0 { начальное состояние словаря }
for k from 1 to m do
  p := g[k].p { p — индекс слова в словаре }
  q := g[k].q { q — дополнительная буква }
  yield D[p] + q { вывод слова и ещё одной буквы }
  d := d + 1; D[d] := D[p] + q { пополнение словаря }
end for
```

ЗАМЕЧАНИЕ

На практике применяют различные усовершенствования этой схемы:

1. Словарь можно сразу инициализировать, например, кодами символов (то есть считать, что однобуквенные слова уже известны).
2. В текстах часто встречаются регулярные последовательности: пробелы и табуляции в таблицах и т. п. Сопоставлять каждой подпоследовательности такой последовательности отдельное слово в словаре нерационально. В таких случаях лучше применить специальный приём, например, закодировать последовательность пробелов парой $\langle k, s \rangle$, где k — количество пробелов, а s — код пробела.

6.5. Шифрование

Защита информации, хранящейся в компьютере, от несанкционированного доступа, искажения и уничтожения в настоящее время является серьёзной социальной проблемой. Применяются различные подходы к решению этой проблемы:

- Поставить между злоумышленником и данными в компьютере непреодолимый «физический» барьер, то есть исключить саму возможность доступа к данным путем изоляции компьютера с данными в охраняемом помещении, применения аппаратных ключей защиты и т. п. Такой подход надёжен, но он затрудняет доступ к данным для законных пользователей, а потому постепенно уходит в прошлое.
- Поставить между злоумышленником и данными в компьютере «логический» барьер, то есть проверять наличие прав на доступ к данным и блокировать доступ при отсутствии таких прав. Для этого применяются различные системы паролей, регистрация и идентификация пользователей, разграничение прав доступа и т. п. Практика показывает, что борьба между «хакерами» и разработчиками модулей защиты операционных систем идёт с переменным успехом.
- Хранить данные таким образом, чтобы они могли «сами за себя постоять». Другими словами, так закодировать данные, чтобы, даже получив к ним доступ, злоумышленник не смог бы нанести ущерба.

Этот раздел посвящён обсуждению методов кодирования, применяемых в последнем случае.

6.5.1. Криптография

Шифрование — это кодирование данных с целью защиты от несанкционированного доступа. Процесс кодирования сообщения называется *шифрованием* (или *зашифровкой*), а процесс декодирования — *расшифровыванием* (или *расшифровкой*). Само кодированное сообщение называется *шифрованным* (или просто *шифровкой*), а применяемый метод называется *шифром*.

Основное требование к шифру состоит в том, чтобы расшифровка (и, может быть, зашифровка) была возможна только при наличии санкции, то есть некоторой дополнительной информации (или устройства), которая называется *ключом* шифра. Процесс декодирования шифровки *без* ключа называется *дешифрованием* (или *дешифрацией*, или просто *раскрытием* шифра).

Область знаний о методах создания и раскрытия шифров называется *криптографией* (или *тайнописью*).

Свойство шифра противостоять раскрытию называется *криптостойкостью* (или *надёжностью*) и обычно оценивается сложностью алгоритма дешифрации.

ОТСТУПЛЕНИЕ

В практической криптографии криптостойкость шифра оценивается из экономических соображений. Если раскрытие шифра стоит (в денежном выражении, включая необходимые компьютерные ресурсы, специальные устройства, услуги специалистов и т. п.) больше, чем сама зашифрованная информация, то шифр считается достаточно надёжным.

Криптография известна с глубокой древности и использует самые разнообразные шифры, как чисто информационные, так и механические. В настоящее время наибольшее практическое значение имеет защита данных в компьютере, поэтому далее рассматриваются программные шифры для сообщений в алфавите $\{0,1\}$.

6.5.2. Шифрование с помощью случайных чисел

Пусть имеется датчик *псевдослучайных* чисел, работающий по некоторому определённому алгоритму. Часто используют следующий алгоритм:

$$T_{i+1} := (a \cdot T_i + b) \mod c,$$

где T_i — предыдущее псевдослучайное число, T_{i+1} — следующее псевдослучайное число, а коэффициенты a, b, c постоянны и хорошо известны. Обычно $c = 2^n$, где n — разрядность процессора, $a \bmod 4 = 1$, а b — нечётное. В этом случае последовательность псевдослучайных чисел имеет период c (см. [15]).

Процесс шифрования определяется следующим образом. Шифруемое сообщение представляется в виде последовательности слов $S_0, S_1, \ldots$, каждое длины n, которые складываются по модулю 2 со словами последовательности $T_0, T_1, \ldots$, то есть

$$C_i := S_i +_2 T_i.$$

ЗАМЕЧАНИЕ

Последовательность $T_0, T_1, \ldots$ называется *гаммой шифра*.

Процесс расшифровывания заключается в том, чтобы ещё раз сложить шифрованную последовательность с той же самой гаммой шифра:

$$S_i := C_i +_2 T_i.$$

Ключом шифра является начальное значение T_0, которое является секретным и должно быть известно только отправителю и получателю шифрованного сообщения.

ЗАМЕЧАНИЕ

Шифры, в которых для зашифровки и расшифровки используется один и тот же ключ, называются *симметричными*.

Если период последовательности псевдослучайных чисел достаточно велик, чтобы гамма шифра была длиннее сообщения, то дешифровать сообщение можно только подбором ключа. При увеличении n экспоненциально увеличивается криптостойкость шифра.

ОТСТУПЛЕНИЕ

Этот очень простой и эффективный метод часто применяют «внутри» программных систем, например, для защиты данных на локальном диске. Для защиты данных, передаваемых по открытым каналам связи, особенно в случае многостороннего обмена сообщениями, этот метод применяют не так часто, поскольку возникают трудности с надёжной передачей секретного ключа многим пользователям.

6.5.3. Криптостойкость

Описанный в предыдущем подразделе метод шифрования обладает существенным недостатком. Если известна хотя бы часть исходного сообщения, то всё сообщение может быть легко дешифровано. Действительно, пусть известно одно исходное слово S_i. Тогда

$$T_i := C_i +_2 S_i,$$

и далее вся правая часть гаммы шифра определяется по указанной формуле датчика псевдослучайных чисел.

ЗАМЕЧАНИЕ

На практике часть сообщения вполне может быть известна злоумышленнику. Например, многие текстовые редакторы помещают в начало файла документа одну и ту же служебную информацию. Если злоумышленнику известно, что исходное сообщение подготовлено в данном редакторе, то он сможет легко дешифровать сообщение.

Для повышения криптостойкости симметричных шифров применяют различные приёмы:

1. Вычисление гаммы шифра по ключу более сложным (или секретным) способом.
2. Применение вместо $+_2$ более сложной (но обратимой) операции для вычисления шифровки.
3. Предварительное перемешивание битов исходного сообщения по фиксированному алгоритму.

Наиболее надёжным симметричным шифром считается DES (Data Encryption Standard), в котором используется сразу несколько методов повышения криптостойкости.

6.5.4. Модулярная арифметика

Для объяснения метода шифрования с открытым ключом, описанного в следующем подразделе, нужны некоторые факты из теории чисел, изложенные здесь без доказательств.

В этом подразделе рассматриваются только целые числа. Говорят, что число a *сравнимо по модулю* n с числом b (обозначение $a \equiv b \bmod n$), если a и b при делении на n дают один и тот же остаток:

$$a \equiv b \bmod n \stackrel{\text{Def}}{=} a \bmod n = b \bmod n.$$

ЗАМЕЧАНИЕ

В левой части этого определения слово mod является частью обозначения трехместного отношения, а в правой части слово mod является обозначением двухместной операции определения остатка от деления.

Отношение *сравнимости* рефлексивно, симметрично и транзитивно и является отношением эквивалентности. Классы эквивалентности по отношению сравнимости (по модулю n) называются *вычетами* (по модулю n). Множество вычетов по модулю n обозначается $\mathbb{Z}_n$. Обычно из каждого вычета выбирают одного представителя — неотрицательное число, которое при делении на n дает частное 0. Это позволяет считать, что $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$, и упростить обозначения. Над вычетами (по модулю n) определены операции сложения и умножения по модулю n, обозначаемые, соответственно, $+_n$ и $\cdot_n$ и определяемые следующим образом:

$$a +_n b \stackrel{\text{Def}}{=} (a + b) \bmod n, \qquad a \cdot_n b \stackrel{\text{Def}}{=} (a \cdot b) \bmod n.$$

ЗАМЕЧАНИЕ

Если из контекста ясно, что подразумеваются операции по модулю n, то индекс n опускается.

Легко видеть, что $\langle \mathbb{Z}_n; +_n \rangle$ является абелевой группой, а $\langle \mathbb{Z}_n; +_n, \cdot_n \rangle$ — коммутативным кольцом с единицей.

ЗАМЕЧАНИЕ

Если n — простое число, то $\langle \mathbb{Z}_n; +_n, \cdot_n \rangle$ является полем.

Рассмотрим $\mathbb{Z}_n^*$ — подмножество множества вычетов $\mathbb{Z}_n$, взаимно простых с n.

ЗАМЕЧАНИЕ

Числа a и b называются *взаимно простыми*, если их наибольший общий делитель равен 1.

Можно показать, что $\langle \mathbb{Z}_n^*; \cdot_n \rangle$ — абелева группа. Таким образом, для элементов множества $\mathbb{Z}_n^*$ существуют обратные по умножению по модулю n.

Функция $\varphi(n) \stackrel{\text{Def}}{=} |\mathbb{Z}_n^*|$ называется *функцией Эйлера*.

ЗАМЕЧАНИЕ

Если p — простое число, то $\varphi(p) = p - 1$, и вообще, $\varphi(n) < n$.

Можно показать, что

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_k}\right),$$

где $p_1, \ldots, p_k$ — все простые делители n.

Имеет место следующая теорема.

ТЕОРЕМА (Эйлера) *Если* $n > 1$, *то*

$$\forall a \in \mathbb{Z}_n^* \left(a^{\varphi(n)} \equiv 1 \bmod n\right).$$

Отсюда непосредственно вытекает

ТЕОРЕМА (малая теорема Ферма[1]) *Если* $p > 1$ — *простое число, то*

$$\forall a \in \mathbb{Z}_p^* \left(a^{p-1} \equiv 1 \bmod p\right).$$

Имеет место следующее утверждение.

ТЕОРЕМА *Если числа* $n_1, \ldots, n_k$ *попарно взаимно просты,* $n = n_1 n_2 \cdots n_k$ — *их произведение,* x *и* a — *целые числа, то*

$$x \equiv a \bmod n \iff \forall i \in 1..k \ (x \equiv a \bmod n_i).$$

ЗАМЕЧАНИЕ

Последнее утверждение известно как «китайская теорема об остатках».

6.5.5. Шифрование с открытым ключом

В настоящее время широкое распространение получили шифры *с открытым ключом*. Эти шифры не являются симметричными — для зашифровки и расшифровки используются разные ключи. При этом ключ, используемый для зашифровки, является открытым (не секретным) и может быть сообщен всем желающим отправить шифрованное сообщение, а ключ, используемый для расшифровки, является закрытым и хранится в секрете получателем шифрованных сообщений. Даже знание всего зашифрованного сообщения и открытого ключа, с помощью которого оно было зашифровано, не позволяет дешифровать сообщение (без знания закрытого ключа).

Шифрование с открытым ключом производится следующим образом:

1. Получателем сообщений производится генерация открытого ключа (пара чисел n и e) и закрытого ключа (число d). Для этого:
   - выбираются два простых числа p и q;
   - вычисляется первая часть открытого ключа $n := pq$;
   - определяется вторая часть открытого ключа — выбирается небольшое нечётное число e, взаимно простое с числом $(p-1)(q-1)$ (заметим, что $(p-1)(q-1) = pq(1-1/p)(1-1/q) = \varphi(n)$);
   - определяется закрытый ключ: $d := e^{-1} \bmod (p-1)(q-1)$.

   После чего открытый ключ (числа n и e) сообщается всем отправителям сообщений.

[1] Пьер де Ферма (1601–1665).

2. Отправитель шифрует сообщение (разбивая его, если нужно, на слова S_i длиной менее $\log_2 n$ разрядов):

$$C_i := (S_i)^e \bmod n,$$

и отправляет получателю.

3. Получатель расшифровывает сообщение с помощью закрытого ключа d:

$$P_i := (C_i)^d \bmod n.$$

ТЕОРЕМА *Шифрование с открытым ключом корректно, то есть в предыдущих обозначениях $P_i = S_i$.*

Доказательство Легко видеть, что $P_i = (S_i)^{ed} \bmod n$. Покажем, что

$$\forall M < n \ \left(M^{ed} \equiv M \bmod n\right).$$

Действительно, числа d и e взаимно обратны по модулю $(p-1)(q-1)$, то есть

$$ed = 1 + k(p-1)(q-1)$$

при некотором k. Если $M \not\equiv 0 \bmod p$, то по малой теореме Ферма имеем:

$$M^{ed} \equiv M\left(M^{(p-1)}\right)^{k(q-1)} \equiv M \cdot 1^{k(q-1)} \equiv M \bmod p.$$

Если $M \equiv 0 \bmod p$, то сравнение $M^{ed} \equiv M \bmod p$, очевидно, выполняется. Таким образом,

$$\forall 0 \leqslant M < n \ \left(M^{ed} \equiv M \bmod p\right).$$

Совершенно аналогично имеем

$$\forall 0 \leqslant M < n \ \left(M^{ed} \equiv M \bmod q\right),$$

и по китайской теореме об остатках

$$\forall M < n \ \left(M^{ed} \equiv M \bmod n\right).$$

Поскольку $S_i < n$ и $P_i < n$, заключаем, что $\forall i \ (P_i = S_i)$. □

Пример Генерация ключей:

1. $p := 3$, $q := 11$.
2. $n := pq = 3 * 11 = 33$.
3. $(p-1)(q-1) = 2 * 10 = 20$, $e := 7$.
4. $d := 7^{-1} \bmod 20 = 3$, $(7 * 3 \bmod 20 = 1)$.

Пусть $S_1 := 3$, $S_2 := 1$, $S_3 := 2$ $(S_1, S_2, S_3 < n = 33)$. Тогда код определяется следующим образом:

1. $C_1 := 3^7 \bmod 33 = 2187 \bmod 33 = 9$.
2. $C_2 := 1^7 \bmod 33 = 1 \bmod 33 = 1$.
3. $C_3 := 2^7 \bmod 33 = 128 \bmod 33 = 29$.

При расшифровке имеем:

1. $P_1 := 9^3 \bmod 33 = 729 \bmod 33 = 3$.
2. $P_2 := 1^3 \bmod 33 = 1 \bmod 33 = 1$.
3. $P_3 := 29^3 \bmod 33 = 24389 \bmod 33 = 2$.

ОТСТУПЛЕНИЕ

Шифры с открытым ключом сравнительно просты в реализации, очень практичны (поскольку нет необходимости пересылать по каналам связи закрытый ключ и можно безопасно хранить его в одном месте) и в то же время обладают высочайшей криптостойкостью. Кажется, что дешифровать сообщение несложно: достаточно разложить открыто опубликованное число n на множители, восстановив числа p и q, и далее можно легко вычислить секретный ключ d. Однако дело заключается в следующем. В настоящее время известны эффективные алгоритмы определения простоты чисел, которые позволяют за несколько минут подобрать пару очень больших простых чисел (по 100 и больше цифр в десятичной записи). В то же время неизвестны эффективные алгоритмы разложения очень больших чисел на множители. Разложение на множители числа в 200 и больше цифр потребовало бы сотен лет работы самого лучшего суперкомпьютера. При практическом применении шифров с открытым ключом используют действительно большие простые числа (не менее 100 цифр в десятичной записи, а обычно значительно больше). В результате вскрыть этот шифр оказывается невозможно, если не существует эффективных алгоритмов разложения на множители (что очень вероятно, хотя и не доказано строго).

6.5.6. Цифровая подпись

Шифр с открытым ключом позволяет выполнять и многие другие полезные операции, помимо шифрования и посылки сообщений в одну сторону. Прежде всего, для организации многосторонней секретной связи каждому из участников достаточно сгенерировать свою пару ключей (открытый и закрытый), а затем сообщить всем партнёрам свой открытый ключ.

Заметим, что операции зашифровки и расшифровки по существу одинаковы, и различаются только показателем степени, а потому коммутируют:

$$M = (M^e)^d \bmod n = M^{ed} \bmod n = M^{de} \bmod n = (M^d)^e \bmod n = M.$$

Это обстоятельство позволяет применять различные приёмы, известные как *цифровая* (или *электронная*) подпись.

Рассмотрим следующую схему взаимодействия корреспондентов X и Y. Отправитель X кодирует сообщение S своим закрытым ключом ($C := S^d \bmod n$) и посылает получателю Y пару $\langle S, C\rangle$, то есть подписанное сообщение. Получатель Y, получив такое сообщение, кодирует подпись сообщения открытым ключом отправителя X, то есть вычисляет $S' := C^e \bmod n$. Если оказывается, что $S = S'$, то это означает, что (нешифрованное!) сообщение S действительно было отправлено корреспондентом X. Если же $S \neq S'$, то сообщение было искажено при передаче или фальсифицировано.

ОТСТУПЛЕНИЕ

В подобного рода схемах возможны различные проблемы, которые носят уже не математический, а социальный характер. Например, допустим, что злоумышленник Z имеет техническую возможность контролировать всю входящую корреспонденцию получателя Y незаметно для последнего. Тогда, перехватив сообщение отправителя X, в котором сообщался открытый ключ e, злоумышленник Z может подменить открытый ключ отправителя X своим собственным открытым ключом. После этого злоумышленник сможет фальсифицировать все сообщения отправителя X, подписывая их своей цифровой подписью, и, таким образом, действовать от имени отправителя X. Другими словами, цифровая подпись удостоверяет, что сообщение S пришло из того же источника, из которого был получен открытый ключ e, но не более того.

Можно подписывать и шифрованные сообщения. Для этого отправитель X сначала кодирует своим закрытым ключом сообщение S, получая цифровую подпись C, а затем кодирует полученную пару $\langle S, C\rangle$ открытым ключом получателя Y. Получив такое сообщение, получатель Y сначала расшифровывает сообщение своим закрытым ключом, а потом убеждается в подлинности полученного сообщения, сравнив его с результатом применения открытого ключа отправителя X к подписи C.

ЗАМЕЧАНИЕ

К сожалению, даже эти меры не смогут защитить от злоумышленника Z, сумевшего подменить открытый ключ отправителя X. Конечно, в этом случае злоумышленник не сможет дешифровать исходное сообщение, но он сможет подменить исходное сообщение фальсифицированным, а получатель не сможет обнаружить подмену сообщения.

Комментарии

Вопросы, затронутые в этой главе, очень существенны для практических информационных технологий, которые невозможны без кодирования, сжатия данных и шифрования. Разумеется, здесь описаны простейшие варианты, в реальных современных программах применяются более изощрённые методы. Общие вопросы кодирования достаточно подробно описаны в [10] и [11]. Алгоритмы связанные с оптимальным и помехоустойчивым кодированием заимствованы (в переработанном виде) из [10]. По вопросам сжатия данных помимо специальной литературы см. [9]. Шифрованию посвящено множество специальных монографий. Лаконичное изложение основных идей можно найти в [17].

Упражнения

6.1. Является ли схема алфавитного кодирования

$$\langle a \to 0, b \to 10, c \to 011, d \to 1011, e \to 1111\rangle$$

префиксной? Разделимой?

6.2. Построить оптимальное префиксное алфавитное кодирование для алфавита $\{a, b, c, d\}$ со следующим распределением вероятностей появления букв:

$$p_a = 1/2,\ p_b = 1/4,\ p_c = p_d = 1/8.$$

6.3. Показать, что для несимметричных ошибок функция

$$d_\delta(\beta', \beta'') \stackrel{\text{Def}}{=} 2 \min_{\{\beta''' \in B^*\}} \max \left(\min_{\{E^\delta\langle\beta', \beta'''\rangle\}} |E^\delta\langle\beta', \beta'''\rangle|, \min_{\{E^\delta\langle\beta''', \beta''\rangle\}} |E^\delta\langle\beta''', \beta''\rangle| \right)$$

является расстоянием.

6.4. Проследить работу алгоритма сжатия Лемпела—Зива на примере следующего исходного текста: abaabaaab.

6.5. Пусть в системе программирования имеется процедура Randomize, которая получает целочисленный параметр и инициализирует датчик псевдослучайных чисел, и функция без параметров Rnd, которая выдает следующее псевдослучайное число в интервале $[0, 1]$. Составить алгоритмы шифровки и расшифровки с закрытым ключом.

Глава 7 Графы

Эта глава открывает заключительную часть книги, целиком посвященную графам и алгоритмам на них. Среди дисциплин и методов дискретной математики теория графов и особенно алгоритмы на графах находят наиболее широкое применение в программировании. Как показано в разделе 7.5, между понятием графа и понятием бинарного отношения, рассмотренным в главе 1, имеется глубокая связь — можно сказать, что это равнообъёмные понятия. Возникает естественный вопрос, почему же тогда графам оказывается столь заметное предпочтение при изучении дискретной математики и программирования? Дело в том, что теория графов предоставляет очень удобный *язык* для описания программных (да и многих других) моделей. Этот тезис можно пояснить следующей аналогией. Понятие отношения также можно полностью выразить через понятие множества (см. замечание в подразделе 1.4.1 и далее). Однако независимое определение понятия отношения *удобнее* — введение специальных терминов и обозначений упрощает изложение теории и делает её более понятной. То же относится и к теории графов. Стройная система специальных терминов и обозначений теории графов позволяет просто и доступно описывать сложные и тонкие вещи. Особенно важна возможность наглядной графической интерпретации понятия графа (см. 7.1.4). Само название «граф» подразумевает наличие графической интерпретации. Картинки часто позволяют сразу «усмотреть» суть дела на интуитивном уровне, дополняя и украшая утомительные текстовые доказательства и сложные формулы. Эта глава практически полностью посвящена описанию языка теории графов.

7.1. Определения графов

Как это ни удивительно, но для понятия «граф» нет общепризнанного единого определения. Разные авторы, особенно применительно к разным приложениям, называют словом «граф» очень похожие, но все-таки различные объекты. Здесь используется терминология [28], которая была выбрана из соображений максимального упрощения определений и доказательств.

7.1.1. История теории графов

Теория графов многократно переоткрывалась разными авторами при решении различных прикладных задач.

1. *Задача о Кёнигсбергских мостах.* На рис. 7.1 представлен схематический план центральной части города Кёнигсберг (ныне Калининград), включающий два берега реки Перголя, два острова на ней и семь соединяющих их мостов. Задача состоит в том, чтобы обойти все четыре участка суши, пройдя по каждому мосту один раз, и вернуться в исходную точку. В 1736 году Эйлером[1] было показано, что решения этой задачи не существует. Точнее говоря, Эйлер получил необходимое и достаточное условие существования решения для всех задач типа задачи о Кёнигсбергских мостах (см. 10.2.1).

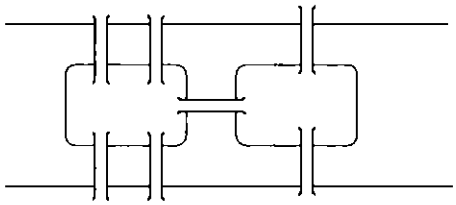

Рис. 7.1. Иллюстрация к задаче о Кёнигсбергских мостах

2. *Задача о трёх домах и трёх колодцах.* Имеется три дома и три колодца, каким-то образом расположенные на плоскости. Требуется провести от каждого дома к каждому колодцу тропинку так, чтобы тропинки не пересекались (рис. 7.2). Эта задача также не имеет решения. В 1930 году Куратовским[1] было доказано намного более сильное утверждение, а именно достаточность условия существования решения для всех задач типа задачи о трёх домах и трёх колодцах (необходимость этого условия была известна и ранее, см. 10.8.2).

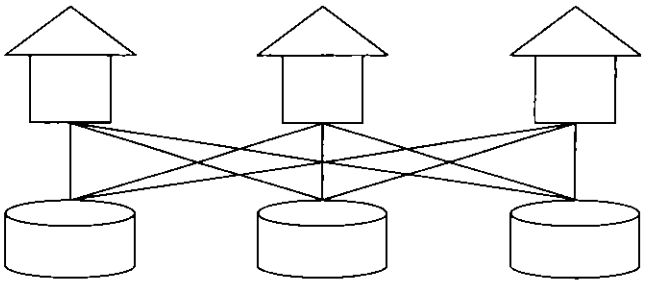

Рис. 7.2. Иллюстрация к задаче о трёх домах и трёх колодцах

3. *Задача о четырёх красках.* Разделение плоскости на неперекрывающиеся области называется *картой*. Области на карте называются соседними, если они имеют общую границу. Задача состоит в раскрашивании карты таким образом, чтобы никакие две соседние области не были закрашены одним цветом (рис. 7.3). С конца XIX века известна гипотеза, что для этого достаточно четырёх красок. В 1976 году Аппель и Хейкен опубликовали решение задачи

[1] Леонард Эйлер (1707–1783).

[1] Казимир Куратовский (1896–1979).

о четырёх красках, которое базировалось на переборе вариантов с помощью компьютера.

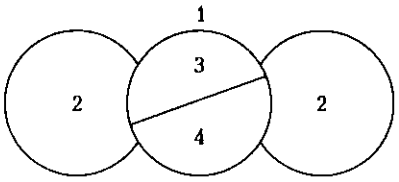

Рис. 7.3. Иллюстрация к задаче о четырёх красках

ОТСТУПЛЕНИЕ

Решение задачи о четырёх красках Аппелем и Хейкеном «программным путем» явилось прецедентом, породившим бурную дискуссию, которая отнюдь не закончена. Суть опубликованного решения состоит в том, чтобы перебрать большое, но конечное число (около 2000) типов потенциальных контрпримеров к теореме о четырёх красках и показать, что ни один случай контрпримером не является. Этот перебор был выполнен программой примерно за тысячу часов работы суперкомпьютера. Проверить «вручную» полученное решение невозможно — объём перебора выходит далеко за рамки человеческих возможностей. Многие математики ставят вопрос: можно ли считать такое «программное доказательство» действительным доказательством? Ведь в программе могут быть ошибки... Методы формального доказательства правильности программ не применимы к программам такой сложности, как обсуждаемая. Тестирование не может гарантировать отсутствие ошибок и в данном случае вообще невозможно. Таким образом, остаётся уповать на программистскую квалификацию авторов и верить, что они всё сделали правильно.

7.1.2. Основное определение

Графом $G(V, E)$ называется совокупность двух множеств — непустого множества V (множества *вершин*) и множества E двухэлементных подмножеств множества V (E — множество *рёбер*),

$$G(V,E) \stackrel{\text{Def}}{=} \langle V; E\rangle\,, \quad V \neq \varnothing, \quad E \subset 2^V \;\&\; \forall e \in E\ (|e| = 2)\,.$$

ЗАМЕЧАНИЕ

Легко видеть, что любое множество E двухэлементных подмножеств множества V определяет симметричное бинарное отношение на множестве V. Поэтому можно считать, что

$$E \subset V \times V, \quad E = E^{-1}$$

и трактовать ребро не только как множество $\{v_1, v_2\}$, но и как пару (v_1, v_2).

Число вершин графа G обозначим p, а число рёбер — q:

$$p \stackrel{\text{Def}}{=} p(G) \stackrel{\text{Def}}{=} |V|, \quad q \stackrel{\text{Def}}{=} q(G) \stackrel{\text{Def}}{=} |E|.$$

Если хотят явно упомянуть числовые характеристики графа, то говорят, (p, q)-граф.

7.1.3. Смежность

Пусть v_1, v_2 — вершины, $e = (v_1, v_2)$ — соединяющее их ребро. Тогда вершина v_1 и ребро e *инцидентны*, ребро e и вершина v_2 также инцидентны. Два ребра, инцидентные одной вершине, называются *смежными*; две вершины, инцидентные одному ребру, также называются *смежными*. Множество вершин, смежных с вершиной v, называется *множеством смежности* (или *окрестностью*) вершины v и обозначается $\Gamma^+(v)$:

$$\Gamma^+(v) \stackrel{\text{Def}}{=} \{u \in V \mid (u, v) \in E\}, \qquad \Gamma^*(v) \stackrel{\text{Def}}{=} \Gamma^+(v) + v.$$

ЗАМЕЧАНИЕ

Если не оговорено противное, то символ Γ без индекса подразумевает Γ^+, то есть саму вершину в окрестность не включают.

Очевидно, что $u \in \Gamma(v) \iff v \in \Gamma(u)$. Если $A \subset V$ — множество вершин, то $\Gamma(A)$ — множество всех вершин, смежных с вершинами из A:

$$\Gamma(A) \stackrel{\text{Def}}{=} \{u \in V \mid \exists v \in A \ (u \in \Gamma(v))\} = \bigcup_{v \in A} \Gamma(v).$$

7.1.4. Диаграммы

Обычно граф изображают *диаграммой*: вершины — точками (или кружками), рёбра — линиями.

Пример На рис. 7.4 приведен пример диаграммы графа, имеющего четыре вершины и пять рёбер. В этом графе вершины v_1 и v_2, v_2 и v_3, v_3 и v_4, v_4 и v_1, v_2 и v_4 смежны, а вершины v_1 и v_3 не смежны. Смежные рёбра: e_1 и e_2, e_2 и e_3, e_3 и e_4, e_4 и e_1, e_1 и e_5, e_2 и e_5, e_3 и e_5, e_4 и e_5. Несмежные рёбра: e_1 и e_3, e_2 и e_4.

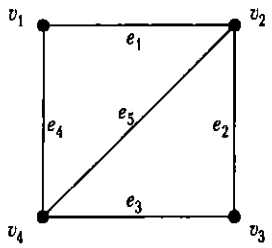

Рис. 7.4. Диаграмма графа

7.1.5. Орграфы, псевдографы, мультиграфы и гиперграфы

Часто рассматриваются следующие родственные графам объекты:

1. Если элементами множества E являются *упорядоченные* пары (т. е. $E \subset V \times V$), то граф называется *ориентированным* (или *орграфом*). В этом случае элементы множества V называются *узлами*, а элементы множества E — *дугами*.
2. Если элементом множества E может быть пара *одинаковых* (не различных) элементов V, то такой элемент множества E называется *петлей*, а граф называется *графом с петлями* (или *псевдографом*).
3. Если E является не множеством, а *мультимножеством*, содержащим некоторые элементы по несколько раз, то эти элементы называются *кратными рёбрами*, а граф называется *мультиграфом*.
4. Если элементами множества E являются не обязательно двухэлементные, а *любые* (непустые) подмножества множества V, то такие элементы множества E называются *гипердугами*, а граф называется *гиперграфом*.
5. Если задана функция $F\colon V \to M$ и/или $F\colon E \to M$, то множество M называется множеством *пометок*, а граф называется *помеченным* (или *нагруженным*). В качестве множества пометок обычно используются буквы или целые числа. Если функция F инъективна, то есть разные вершины (рёбра) имеют разные пометки, то граф называют *нумерованным*.

ЗАМЕЧАНИЕ

Отношение смежности является в некотором смысле определяющим для графов и подобных им объектов (см. разделы 7.4 и 7.5). При этом следует учитывать особенности каждого типа объектов. В орграфе вершина v смежна с вершиной u, если существует дуга (u, v). При этом вершина u может быть несмежна с вершиной v. Отношение смежности в графе симметрично, а в орграфе оно вовсе не обязано быть симметричным. В графе обычно считают отношение смежности рефлексивным, то есть полагают, что вершина смежна сама с собой. В псевдографе, напротив, вершину не считают смежной с собой, если у неё нет петли. В гиперграфе две вершины считаются смежными, если они принадлежат одному гиперребру. В гиперорграфе гипердуга обычно проводится из одного узла в множество узлов (возможно, пустое). В таком случае отношение смежности оказывается уже не бинарным отношением на V, а отношением из V в 2^V. Эти и подобные естественные вариации определений обычно считают ясными из контекста.

Далее выражение «граф $G(V, E)$» означает неориентированный непомеченный граф без петель и кратных рёбер с множеством вершин V и множеством рёбер E.

7.1.6. Изоморфизм графов

Говорят, что два графа, $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$, *изоморфны* (обозначается $G_1 \sim G_2$, или $G_1 = G_2$), если существует биекция $h\colon V_1 \to V_2$, сохраняющая смежность (см. 2.1.6):

$$e_1 = (u, v) \in E_1 \iff e_2 = \big(h(u), h(v)\big) \in E_2.$$

ТЕОРЕМА *Изоморфизм графов есть отношение эквивалентности.*

Доказательство Действительно, изоморфизм обладает всеми необходимыми свойствами:

[Рефлексивность] $G \sim G$, где требуемая биекция есть тождественная функция.

[Симметричность] если $G_1 \sim G_2$ с биекцией h, то $G_2 \sim G_1$ с биекцией h^{-1}.

[Транзитивность] если $G_1 \sim G_2$ с биекцией h и $G_2 \sim G_3$ с биекцией g, то $G_1 \sim G_3$ с биекцией $g \circ h$. □

Графы рассматриваются *с точностью до изоморфизма*, то есть рассматриваются классы эквивалентности по отношению изоморфизма (см. 2.1.5).

Пример Три внешне различные диаграммы, приведённые на рис. 7.5, являются диаграммами одного и того же графа $K_{3,3}$.

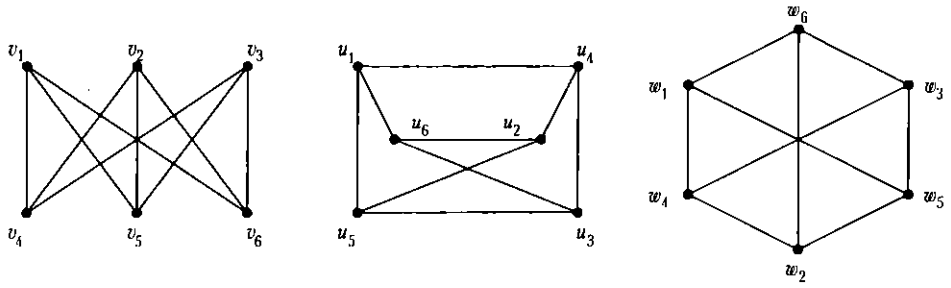

Рис. 7.5. Диаграммы изоморфных графов

Числовая характеристика, одинаковая для всех изоморфных графов, называется *инвариантом* графа. Так, $p(G)$ и $q(G)$ — инварианты графа G. Неизвестно никакого простого набора инвариантов, определяющих граф с точностью до изоморфизма.

Пример Количество вершин, рёбер и количество смежных вершин для каждой вершины не определяют граф даже в простейших случаях! На рис. 7.6 представлены диаграммы графов, у которых указанные инварианты совпадают, но графы при этом не изоморфны.

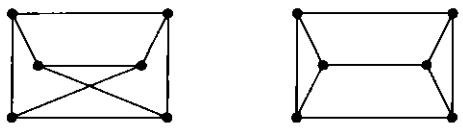

Рис. 7.6. Диаграммы неизоморфных графов с совпадающими инвариантами

7.2. Элементы графов

После рассмотрения определений, относящихся к графам как к целым объектам, естественно дать определения различным элементам графов.

7.2.1. Подграфы

Граф $G'(V', E')$ называется *подграфом* (или *частью*) графа $G(V, E)$ (обозначается $G' \subset G$), если $V' \subset V \;\&\; E' \subset E$. Если $V' = V$, то G' называется *остовным подграфом* G. Если $V' \subset V \;\&\; E' \subset E \;\&\; (V' \neq V \vee E' \neq E)$, то граф G' называется *собственным* подграфом графа G. Подграф $G'(V', E')$ называется *правильным* подграфом графа $G(V, E)$, если G' содержит все возможные рёбра G:

$$\forall u, v \in V' \;\; ((u, v) \in E \Longrightarrow (u, v) \in E').$$

Правильный подграф $G'(V', E')$ графа $G(V, E)$ определяется подмножеством вершин V'.

ЗАМЕЧАНИЕ

Иногда подграфами называют только правильные подграфы, а неправильные подграфы называют *изграфами*.

7.2.2. Валентность

Количество рёбер, инцидентных вершине v, называется *степенью* (или *валентностью*) вершины v и обозначается $d(v)$:

$$\forall v \in V \;\; (0 \leqslant d(v) \leqslant p - 1), \qquad d(v) = |\Gamma^+(v)|.$$

Таким образом, степень $d(v)$ вершины v совпадает с количеством смежных с ней вершин. Количество вершин, не смежных с v, обозначают $\overline{d}(v)$. Ясно, что

$$\forall v \in V \;\; \left(d(v) + \overline{d}(v) = p - 1\right).$$

Обозначим *минимальную* степень вершины графа G через $\delta(G)$, а *максимальную* — через $\Delta(G)$:

$$\delta\left(G(V, E)\right) \stackrel{\text{Def}}{=} \min_{v \in V} d(v), \qquad \Delta\left(G(V, E)\right) \stackrel{\text{Def}}{=} \max_{v \in V} d(v).$$

Ясно, что $\delta(G)$ и $\Delta(G)$ являются инвариантами. Если степени всех вершин равны k, то граф называется *регулярным* степени k:

$$\delta(G) = \Delta(G) = k, \quad \forall v \in V \;\; (d(v) = k).$$

Степень регулярности обозначается $r(G)$. Для нерегулярных графов $r(G)$ не определено.

Примеры

На рис. 7.7 приведена диаграмма регулярного графа степени 3. На рис. 7.6 приведены диаграммы двух регулярных, но неизоморфных графов степени 3.

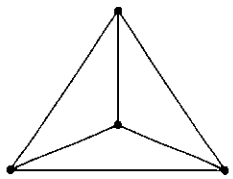

Рис. 7.7. Диаграмма регулярного графа степени 3

Если степень вершины равна нулю (то есть $d(v) = 0$), то вершина называется *изолированной*. Если степень вершины равна единице (то есть $d(v) = 1$), то вершина называется *концевой*, или *висячей*. Для орграфа число дуг, исходящих из узла v, называется *полустепенью исхода*, а число входящих — *полустепенью захода*. Обозначаются эти числа, соответственно, $d^-(v)$ и $d^+(v)$.

ТЕОРЕМА (Лемма о рукопожатиях) *Сумма степеней вершин графа (мультиграфа) равна удвоенному количеству рёбер:*

$$\sum_{v \in V} d(v) = 2q.$$

Доказательство При подсчёте суммы степеней вершин каждое ребро учитывается два раза: для одного конца ребра и для другого. □

СЛЕДСТВИЕ 1 *Число вершин нечётной степени чётно.*

Доказательство По теореме сумма степеней всех вершин — чётное число. Сумма степеней вершин чётной степени чётна, значит, сумма степеней вершин нечётной степени также чётна, следовательно, их чётное число. □

СЛЕДСТВИЕ 2 *Сумма полустепеней узлов орграфа равна удвоенному количеству дуг:*

$$\sum_{v \in V} d^-(v) + \sum_{v \in V} d^+(v) = 2q.$$

Доказательство Сумма полустепеней узлов орграфа равна сумме степеней вершин графа (мультиграфа), полученного из орграфа забыванием ориентации дуг. □

7.2.3. Маршруты, цепи, циклы

Маршрутом в графе называется чередующаяся последовательность вершин и рёбер, начинающаяся и кончающаяся вершиной, $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, в которой любые два соседних элемента инцидентны, причём однородные элементы (вершины, рёбра) через один смежны или совпадают.

ЗАМЕЧАНИЕ

Это определение подходит также для псевдо-, мульти- и орграфов. При этом в графе (орграфе) достаточно указать только последовательность вершин (узлов) или только последовательность рёбер (дуг).

Если $v_0 = v_k$, то маршрут *замкнут*, иначе — *открыт*. Если все рёбра различны, то маршрут называется *цепью*. Если все вершины (а значит, и рёбра) различны, то маршрут называется *простой* цепью. В цепи $v_0, e_1, \ldots, e_k, v_k$ вершины v_0 и v_k называются *концами* цепи. Говорят, что цепь с концами u и v *соединяет* вершины u и v. Цепь, соединяющая вершины u и v, обозначается $\langle u, v\rangle$. Если нужно указать граф G, которому принадлежит цепь, то добавляют индекс: $\langle u, v\rangle_G$. Нетрудно показать, что если есть какая-либо цепь, соединяющая вершины u и v, то есть и простая цепь, соединяющая эти вершины. Замкнутая цепь называется *циклом*; замкнутая простая цепь называется *простым циклом*. Число циклов в графе G обозначается $z(G)$. Граф без циклов называется *ациклическим*.

ЗАМЕЧАНИЕ

Для псевдографов обычно особо оговаривают, считаются ли петли циклами.

Для орграфов цепь называется *путем*, а цикл — *контуром*. Путь в орграфе из узла u в узел v обозначают $\langle \vec{u, v}\rangle$.

Пример В графе, диаграмма которого приведена на рис. 7.8:

1) v_1, v_3, v_1, v_4 — маршрут, но не цепь;
2) $v_1, v_3, v_5, v_2, v_3, v_4$ — цепь, но не простая цепь;
3) v_1, v_4, v_3, v_2, v_5 — простая цепь;
4) $v_1, v_3, v_5, v_2, v_3, v_4, v_1$ — цикл, но непростой цикл;
5) v_1, v_3, v_4, v_1 — простой цикл.

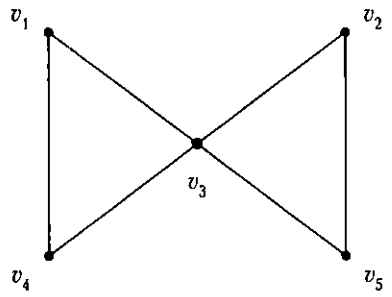

Рис. 7.8. Маршруты, цепи, циклы

7.2.4. Связность

Говорят, что две вершины в графе *связаны*, если существует соединяющая их (простая) цепь. Граф, в котором все вершины связаны, называется *связным*. Нетрудно показать, что отношение связанности вершин является эквивалентностью. Классы эквивалентности по отношению связанности называются *компонентами связности* графа. Число компонентов связности графа G обозначается $k(G)$. Граф G является связным тогда и только тогда, когда $k(G) = 1$. Если $k(G) > 1$, то G — *несвязный* граф. Граф, состоящий только из изолированных вершин (в котором $k(G) = p(G)$ и $r(G) = 0$), называется *вполне несвязным*.

7.2.5. Расстояние между вершинами, ярусы и диаметр графа

Длиной маршрута называется количество рёбер в нём (с учётом повторений). Если маршрут $M = v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, то длина M равна k (обозначается $|M| = k$). *Расстоянием* между вершинами u и v (обозначается $d(u, v)$) называется длина кратчайшей цепи $\langle u, v\rangle$, а сама кратчайшая цепь называется *геодезической*,

$$d(u, v) \stackrel{\text{Def}}{=} \min_{\{\langle u,v\rangle\}} |\langle u, v\rangle|.$$

Если для любых двух вершин графа существует единственная геодезическая цепь, то граф называется *геодезическим*.

ЗАМЕЧАНИЕ

Если $\neg\exists\ (\langle u, v\rangle)$, то по определению $d(u, v) \stackrel{\text{Def}}{=} +\infty$.

Множество вершин, находящихся на заданном расстоянии n от вершины v (обозначение $D(v, n)$), называется *ярусом*:

$$D(v, n) \stackrel{\text{Def}}{=} \{u \in V \mid d(v, u) = n\}.$$

Ясно, что множество вершин V всякого связного графа однозначно разбивается на ярусы относительно данной вершины. *Диаметром* графа G называется длиннейшая геодезическая. Длина диаметра обозначается $D(G)$:

$$D(G) \stackrel{\text{Def}}{=} \max_{u,v \in V} d(u, v).$$

7.2.6. Эксцентриситет и центр

Эксцентриситетом $e(v)$ вершины v в связном графе $G(V, E)$ называется максимальное расстояние от вершины v до других вершин графа G:

$$e(v) \stackrel{\text{Def}}{=} \max_{u \in V} d(v, u).$$

Заметим, что наиболее эксцентричные вершины — это концы диаметра. *Радиусом* $R(G)$ графа G называется наименьший из эксцентриситетов вершин:

$$R(G) \stackrel{\text{Def}}{=} \min_{v \in V} e(v).$$

Вершина v называется *центральной*, если её эксцентриситет совпадает с радиусом графа, $e(v) = R(G)$. Множество центральных вершин называется *центром* графа и обозначается $C(G)$:

$$C(G) \stackrel{\text{Def}}{=} \{v \in V \mid e(v) = R(G)\}.$$

Пример На рис. 7.9 указаны эксцентриситеты вершин и центры двух графов. Вершины, составляющие центр, выделены жирными точками.

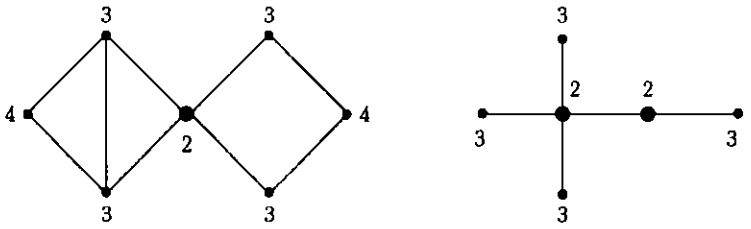

Рис. 7.9. Эксцентриситеты вершин и центры графов

7.3. Виды графов и операции над графами

В данном разделе рассматриваются различные частные случаи графов и вводятся операции над графами и их элементами. Заметим, что не все используемые нами обозначения операций над графами являются традиционными и общепринятыми.

7.3.1. Виды графов

Граф, состоящий из одной вершины, называется *тривиальным*. Граф, состоящий из простого цикла с k вершинами, обозначается C_k.

Пример C_3 — *треугольник.*

Граф, в котором любые две вершины смежны, называется *полным*. Полный граф с p вершинами обозначается K_p, он имеет максимально возможное число рёбер:

$$q(K_p) = \frac{p(p-1)}{2}.$$

Полный подграф (некоторого графа) называется *кликой* (этого графа).

7.3.2. Двудольные графы

Граф $G(V, E)$ называется *двудольным* (или *графом*, или *чётным* графом), если множество V может быть разбито на два непересекающихся множества V_1 и V_2 ($V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$), причём всякое ребро из E инцидентно вершине из V_1 и вершине из V_2 (то есть соединяет вершину из V_1 с вершиной из V_2). Множества

V_1 и V_2 называются *долями* двудольного графа. Если двудольный граф содержит все рёбра, соединяющие множества V_1 и V_2, то он называется *полным двудольным графом*. Если $|V_1| = m$ и $|V_2| = n$, то полный двудольный граф обозначается $K_{m,n}$.

Пример На рис. 7.5 приведена диаграмма графа $K_{3,3}$.

ТЕОРЕМА *Граф является двудольным тогда и только тогда, когда все его простые циклы имеют чётную длину.*

Доказательство

[$\Longrightarrow$] От противного. Пусть $G(V_1,V_2;E)$ — двудольный граф и $v_1, v_2, \ldots, v_{2k+1}, v_1$ — простой цикл нечётной длины. Пусть $v_1 \in V_1$, тогда $v_2 \in V_2$, $v_3 \in V_1$, $v_4 \in V_2, \ldots$, $v_{2k+1} \in V_1$. Имеем: $v_1, v_{2k+1} \in V_1$ и $(v_1, v_{2k+1}) \in E$, что противоречит двудольности.

[$\Longleftarrow$] Не ограничивая общности, можно считать, что G — связный граф, поскольку каждый компонент связности можно рассматривать отдельно. Разобьём множество V на подмножества V_1 и V_2 с помощью следующей процедуры.

Вход: граф $G(V, E)$.
Выход: Множества V_1 и V_2 — доли графа.
```
select v ∈ V { произвольная вершина }
V1 := v { в начале первая доля содержит v,... }
V2 := ∅ { ...а вторая пуста }
for u ∈ V − v do
  if d(v, u) — чётно then
    V1 := V1 + u { помещаем вершину u в первую долю }
  else
    V2 := V2 + u { помещаем вершину u во вторую долю }
  end if
end for
```

Далее от противного. Пусть есть две вершины в одной доле, соединённые ребром. Пусть для определённости $u, w \in V_2$ и $(u, w) \in E$. Рассмотрим геодезические $\langle v, u\rangle$ и $\langle v, w\rangle$ (здесь v — та произвольная вершина, которая использовалась в алгоритме построения долей графа). Тогда длины $|\langle v, u\rangle|$ и $|\langle v, w\rangle|$ нечётны. Эти геодезические имеют общие вершины (по крайней мере, вершину v). Рассмотрим наиболее удалённую от v общую вершину геодезических $\langle v, u\rangle$ и $\langle v, w\rangle$ и обозначим её v' (может оказаться так, что $v = v'$). Имеем: $|\langle v', u\rangle| + |\langle v', w\rangle| = |\langle v, u\rangle| + |\langle v, w\rangle| - 2|\langle v, v'\rangle|$ — чётно и $v', \ldots, u, w, \ldots, v'$ — простой цикл нечётной длины, что противоречит условию. Если же $u, w \in V_1$, то длины $|\langle v, u\rangle|$ и $|\langle v, w\rangle|$ чётны, и аналогично имеем: $v', \ldots, u, w, \ldots, v'$ — простой цикл нечётной длины. □

СЛЕДСТВИЕ *Ациклические графы двудольны.*

7.3.3. Направленные орграфы и сети

Если в графе ориентировать все рёбра, то получится орграф, который называется *направленным*, или *антисимметричным*. Направленный орграф, полученный из полного графа, называется *турниром*.

ЗАМЕЧАНИЕ

В антисимметричном орграфе не может быть «встречных» дуг (u, v) и (v, u), а в произвольном орграфе такое допустимо.

ОТСТУПЛЕНИЕ

Название «турнир» имеет следующее происхождение. Рассмотрим спортивное соревнование для пар участников (или пар команд), где не предусматриваются ничьи. Пометим вершины орграфа участниками и проведем дуги от победителей к побеждённым. В таком случае турнир в смысле теории графов — это как раз результат однокругового турнира в спортивном смысле.

Если в орграфе полустепень захода некоторого узла равна нулю (то есть $d^+(v) = 0$), то такой узел называется *источником*, если же нулю равна полустепень исхода (то есть $d^-(v) = 0$), то узел называется *стоком*. Направленный слабосвязный (см. 8.5.1) орграф с одним источником и одним стоком называется *сетью*.

7.3.4. Операции над графами

Введем следующие операции над графами:

1. *Дополнением графа* $G_1(V_1, E_1)$ (обозначение — $\overline{G}_1(V_1, E_1)$) называется граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = \overline{E_1} = \{e \in V_1 \times V_1 \mid e \notin E_1\} = V \times V \setminus E$.

   Пример $\overline{K_1} = K_1$.

2. *Объединением (дизъюнктным) графов* $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ (обозначение — $G_1(V_1, E_1) \cup G_2(V_2, E_2)$, при условии $V_1 \cap V_2 = \varnothing$) называется граф $G(V, E)$, где $V = V_1 \cup V_2$ & $E = E_1 \cup E_2$.

   Пример $\overline{K}_{3,3} = C_3 \cup C_3$.

3. *Соединением графов* $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ (обозначение — $G_1(V_1, E_1) + G_2(V_2, E_2)$, при условии $V_1 \cap V_2 = \varnothing$) называется граф $G(V, E)$, где $V = V_1 \cup V_2$ & $E = E_1 \cup E_2 \cup \{e = (v_1, v_2) \mid v_1 \in V_1$ & $v_2 \in V_2\}$.

   Пример $K_{3,3} = \overline{C}_3 + \overline{C}_3$.

4. *Удаление вершины* v из графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) - v$ при условии $v \in V_1$) даёт граф $G_2(V_2, E_2)$, где
$$V_2 = V_1 - v \;\&\; E_2 = E_1 \setminus \{e = (v_1, v_2) \mid v_1 = v \vee v_2 = v\}.$$

   Пример $C_3 - v = K_2$.

5. *Удаление ребра* e из графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) - e$ при условии $e \in E_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = E_1 - e$.

   Пример $K_2 - e = \overline{K}_2$.

6. *Добавление вершины* v в граф $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) + v$ при условии $v \notin V_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1 + v$ & $E_2 = E_1$.

   Пример $K_2 + v = K_2 \cup K_1$.

7. *Добавление ребра* e в граф $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) + e$ при условии $e \notin E_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = E_1 + e$.
8. *Стягивание (правильного) подграфа* A графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1)/A$ при условии $A \subset V_1, v \notin V_1$) даёт граф $G_2(V_2, E_2)$, где
$$V_2 = (V_1 \setminus A) + v,$$
$$E_2 = E_1 \setminus \{e = (u, w) \mid u \in A \vee w \in A\} \cup \{e = (u, v) \mid u \in \Gamma(A) \setminus A\}.$$

   Пример $K_4/C_3 = K_2$.

9. *Размножение вершины* v графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) \uparrow v$ при условии $v \in V_1, v' \notin V_1$) даёт граф $G_2(V_2, E_2)$, где
$$V_2 = V_1 + v' \;\&\; E_2 = E_1 \cup \{(v, v')\} \cup \{e = (u, v') \mid u \in \Gamma^+(v)\}.$$

   Пример $K_2 \uparrow v = C_3$.

Некоторые из примеров, приведённых в определениях операций, нетрудно обобщить. В частности, легко показать, что имеют место следующие соотношения:
$$K_{m,n} = \overline{K}_m + \overline{K}_n, \quad K_{p-1} = K_p - v,$$
$$G + v = G \cup K_1, \quad K_{p-1} = K_p/K_2,$$
$$K_p/K_{p-1} = K_2, \quad K_p = K_{p-1} \uparrow v.$$

Введённые операции обладают рядом простых свойств, которые легко вывести из определений. В частности:
$$G_1 \cup G_2 = G_2 \cup G_1, \qquad G_1 + G_2 = G_2 + G_1,$$
$$G_1 = G_2 \Longleftrightarrow \overline{G}_1 = \overline{G}_2, \qquad \overline{G_1 \cup G_2} = \overline{G}_1 + \overline{G}_2.$$

ЗАМЕЧАНИЕ

Операции добавления и удаления ребра взаимно обратны:

$$\forall e \in E, e' \notin E \ \left((G - e) + e = (G + e') - e' = G\right).$$

В то же время $\forall v' \notin V \ ((G + v') - v' = G)$, но если вершина v графа G не изолированная, то $(G - v) + v \neq G$, потому что в этом случае $q((G - v) + v) < q(G)$.

Результат выполнения нескольких последовательных операций удаления или добавления вершин или рёбер не зависит от порядка выполнения операций:

$$(G + v_1) + v_2 = (G + v_2) + v_1, \qquad (G - v_1) - v_2 = (G - v_2) - v_1,$$
$$(G + e_1) + e_2 = (G + e_2) + e_1, \qquad (G - e_1) - e_2 = (G - e_2) - e_1.$$

Поэтому операции удаления и добавления вершин и рёбер можно обобщить, и допускать в качестве второго аргумента множества вершин и рёбер.

ОТСТУПЛЕНИЕ

Приведённые определения операций над графами и примеры к ним дают повод затронуть один тонкий вопрос, связанный с различиями в традициях математических и программных обозначений. Рассмотрим пример к операции дизъюнктного объединения графов ($\overline{K}_{3,3} = C_3 \cup C_3$). Если введённые обозначения понимать буквально, то этот пример кажется противоречащим определению. Действительно, определение требует, чтобы множества вершин объединяемых графов не пересекались. Если считать, что C_3 обозначает конкретный треугольник, то придётся признать, что в выражении $C_3 \cup C_3$ множества вершин не только пересекаются, но и совпадают, а значит, операция объединения неприменима. На самом деле C_3 обозначает как класс треугольников (все треугольники изоморфны как графы), так и отдельный объект — *экземпляр* этого класса. Как именно следует понимать обозначение, считается ясным из контекста. Если стремиться к (излишней в данном случае) строгости и однозначности обозначений, то приведённый пример можно было бы записать, например, так:

$$\forall G_1(V_1, E_1) \in C_3, G_2(V_2, E_2) \in C_3 \ \left(V_1 \cap V_2 \neq \varnothing \Longrightarrow \exists G_3(V_3, E_3) \in K_{3,3} \ \left(G_1 \cup G_2 \sim \overline{G}_3\right)\right).$$

Подобная неоднозначность обозначений присуща и программированию, хотя и в меньшей степени. Например, ключевое слово **int** в различных контекстах может обозначать встроенный тип данных (класс объектов), операцию порождения нового объекта этого типа (экземпляра класса), операцию преобразования другого объекта в объект типа **int** (явное приведение). Следуя стилю объектно-ориентированных языков программирования, приведённый пример можно было бы записать так:

$$\overline{\mathbf{new} K_{3,3}} = \mathbf{new} C_3 \cup \mathbf{new} C_3.$$

Ради краткости и простоты изложения в этой книге принят значительно менее строгий стиль обозначений в надежде на то, что программистская интуиция и здравый смысл позволят читателю избежать заблуждений, несмотря на вольности в обозначениях.

7.4. Представление графов в программах

Следует еще раз подчеркнуть, что конструирование структур данных для представления в программе объектов математической модели — это основа искусства практического программирования. Мы приводим четыре различных базовых представления графов. Выбор наилучшего представления определяется требованиями конкретной задачи. Более того, на практике используются, как правило, некоторые комбинации или модификации указанных представлений, общее число которых необозримо. Но все они так или иначе основаны на тех базовых идеях, которые описаны в этом разделе.

7.4.1. Требования к представлению графов

Известны различные способы представления графов в памяти компьютера, которые различаются объёмом занимаемой памяти и скоростью выполнения операций над графами. Представление выбирается, исходя из потребностей конкретной задачи. Далее приведены четыре наиболее часто используемых представления с указанием характеристики $n(p,q)$ — объёма памяти для каждого представления. Здесь p — число вершин, а q — число рёбер.

ЗАМЕЧАНИЕ

Значение характеристики $n(p,q)$ указывается с помощью символа O, который означает совпадение по порядку величины (или равенство с точностью до мультипликативной константы c, см. замечание в подразделе 1.3.4). Применительно к измерению занимаемой памяти использование символа O связано с тем, что память может быть измерена в битах, байтах, машинных словах или иных единицах. Коэффициент c при этом меняется, а порядок величины остаётся.

Представления иллюстрируются на конкретных примерах графа G и орграфа D, диаграммы которых представлены на рис. 7.10.

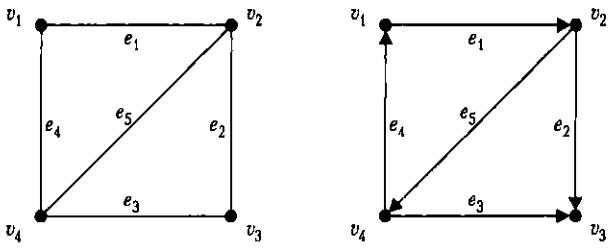

Рис. 7.10. Диаграммы графа (*слева*) и орграфа (*справа*), используемых в качестве примеров

7.4.2. Матрица смежности

Представление графа с помощью квадратной булевой матрицы

$$M : \textbf{array}\ [1..p, 1..p]\ \textbf{of}\ 0..1,$$

отражающей смежность вершин, называется *матрицей смежности*, где

$$M[i,j] = \begin{cases} 1, & \text{если вершина } v_i \text{ смежна с вершиной } v_j, \\ 0, & \text{если вершины } v_i \text{ и } v_j \text{ не смежны.} \end{cases}$$

Для матрицы смежности $n(p,q) = O(p^2)$.

Пример

$$G: \begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix} \qquad D: \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$

ЗАМЕЧАНИЕ

Матрица смежности графа симметрична относительно главной диагонали, поэтому достаточно хранить только верхнюю (или нижнюю) треугольную матрицу.

7.4.3. Матрица инциденций

Представление графа с помощью матрицы

H : **array** $[1..p, 1..q]$ **of** $0..1$, для орграфов H : **array** $[1..p, 1..q]$ **of** $-1..1$,

отражающей инцидентность вершин и рёбер, называется *матрицей инциденций*, где для неориентированного графа

$$H[i,j] = \begin{cases} 1, & \text{если вершина } v_i \text{ инцидентна ребру } e_j, \\ 0, & \text{в противном случае,} \end{cases}$$

а для ориентированного графа

$$H[i,j] = \begin{cases} 1, & \text{если узел } v_i \text{ инцидентен дуге } e_j \text{ и является её концом,} \\ 0, & \text{если узел } v_i \text{ и ребро } e_j \text{ не инцидентны,} \\ -1, & \text{если узел } v_i \text{ инцидентен дуге } e_j \text{ и является её началом.} \end{cases}$$

Для матрицы инциденций $n(p,q) = O(pq)$.

Пример

$$G: \begin{vmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{vmatrix} \qquad D: \begin{vmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{vmatrix}$$

ЗАМЕЧАНИЕ

Для связных графов $q > p$, поэтому матрица смежности несколько компактнее матрицы инциденций.

7.4.4. Списки смежности

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей

$$\Gamma : \textbf{array}\ [1..p]\ \textbf{of}\ \uparrow N$$

на списки смежных вершин, где элемент списка представлен структурой

$$N = \textbf{record}\ v : 1..p;\ n : \uparrow N\ \textbf{end record},$$

называется *списком смежности*. В случае представления неориентированных графов списками смежности $n(p, q) = O(p+2q)$, а в случае ориентированных графов $n(p, q) = O(p + q)$.

ЗАМЕЧАНИЕ

Массив Γ также можно представить списком.

Пример Списки смежности для графа G и орграфа D представлены на рис. 7.11.

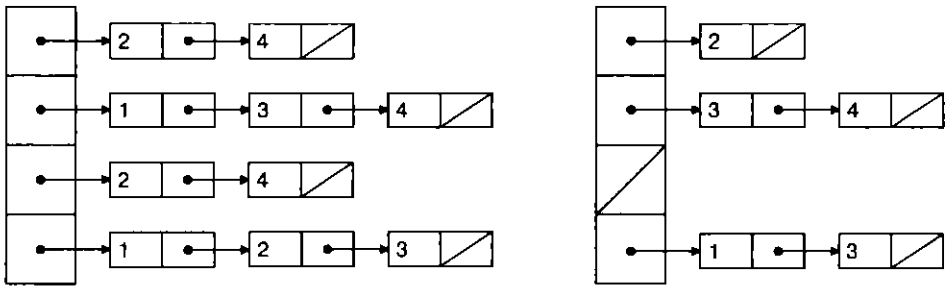

Рис. 7.11. Списки смежности для графа G (*слева*) и орграфа D (*справа*)

7.4.5. Массив дуг

Представление графа с помощью массива структур

$$E : \textbf{array}\ [1..q]\ \textbf{of record}\ b, e : 1..p\ \textbf{end record},$$

отражающего список пар смежных вершин (или, для орграфов, узлов), называется *массивом рёбер* (*массивом дуг*). Для массива рёбер (или дуг) $n(p, q) = O(2q)$.

ЗАМЕЧАНИЕ

Для представления графов с изолированными вершинами может понадобиться хранить ещё и число p, если только система программирования не позволяет извлечь это число из массива структур E.

Пример Представление с помощью массива рёбер (дуг) показано в следующей таблице (для графа G слева, а для орграфа D справа).

b	e	b	e
1	2	1	2
1	4	2	3
2	3	2	4
2	4	4	1
3	4	4	3

ЗАМЕЧАНИЕ

Указанные представления пригодны для графов и орграфов, а после некоторой модификации — также и для псевдографов, мультиграфов и гиперграфов.

7.4.6. Обходы графов

Обход графа — это некоторое систематическое перечисление его вершин (и/или рёбер). Наибольший интерес представляют обходы, использующие локальную информацию (списки смежности). Среди всех обходов наиболее известны поиск *в ширину* и *в глубину*. Алгоритмы поиска в ширину и в глубину лежат в основе многих конкретных алгоритмов на графах.

Алгоритм 7.1 Поиск в ширину и в глубину

Вход: граф $G(V, E)$, представленный списками смежности Γ.
Выход: последовательность вершин обхода.

```
for v ∈ V do
  x[v] := 0 { вначале все вершины не отмечены }
end for
select v ∈ V { начало обхода — произвольная вершина }
v → T { помещаем v в структуру данных T... }
x[v] := 1 { ... и отмечаем вершину v }
repeat
  u ← T { извлекаем вершину из структуры данных T... }
  yield u { ... и возвращаем её в качестве очередной пройденной вершины }
  for w ∈ Γ(u) do
    if x[w] = 0 then
      w → T { помещаем w в структуру данных T... }
      x[w] := 1 { ... и отмечаем вершину w }
    end if
  end for
until T = ∅
```

Если в алгоритме 7.1 структуры данных T — это стек (LIFO — Last In First Out), то обход называется *поиском в глубину*. Если T — это очередь (FIFO — First In First Out), то обход называется *поиском в ширину*.

Пример В следующей таблице показаны протоколы поиска в глубину и в ширину для графа, диаграмма которого приведена на рис. 7.12. Предполагается, что начальной является вершина 1. Слева в таблице протокол поиска в глубину, а справа — в ширину. На рис. 7.12 сплошные стрелки с номерами рядом с рёбрами показывают движение по графу при поиске в глубину, а пунктирные — в ширину.

u	T	u	T
1	2,4	1	2,4
4	2,3	2	4,3
3	2	4	3
2	∅	3	∅

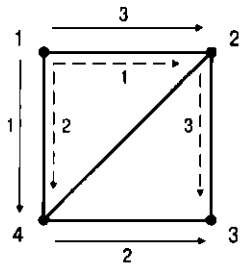

Рис. 7.12. Диаграмма графа к примеру обхода в ширину и в глубину

ТЕОРЕМА *Если граф G связен (и конечен), то поиск в ширину и поиск в глубину обходят все вершины по одному разу.*

Доказательство

[Единственность обхода вершины] Обходятся только вершины, попавшие в T. В T попадают только неотмеченные вершины. При попадании в T вершина отмечается. Следовательно, любая вершина будет обойдена не более одного раза.

[Завершаемость алгоритма] Всего в T может попасть не более p вершин. На каждом шаге одна вершина удаляется из T. Следовательно, алгоритм завершит работу не более чем через p шагов.

[Обход всех вершин] От противного. Пусть алгоритм закончил работу и вершина w не обойдена. Значит, w не попала в T. Следовательно, она не была отмечена. Отсюда следует, что все вершины, смежные с w, не были обойдены и отмечены. Аналогично, любые вершины, связанные с неотмеченными, сами не отмечены (после завершения алгоритма). Но G связен, значит, существует путь $\langle v, w \rangle$. Следовательно, вершина v не отмечена. Но она была отмечена на первом шаге! □

СЛЕДСТВИЕ 1 *Пусть* $(u_1, \ldots, u_i, \ldots, u_j, \ldots, u_p)$ *— обход (то есть последовательность вершин) при поиске в ширину. Тогда*

$$\forall i < j \ (d(u_1, u_i) \leqslant d(u_1, u_j)) .$$

Другими словами, расстояние текущей вершины от начальной является монотонно возрастающей функцией времени поиска в ширину, вершины обходятся в порядке возрастания расстояния от начальной вершины.

СЛЕДСТВИЕ 2 *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в глубину. Тогда*

$$\forall i \geqslant 1 \ (d(u_1, u_i) \leqslant i \leqslant p) .$$

Другими словами, время поиска в глубину любой вершины не менее расстояния от начальной вершины и не более общего числа вершин, причём в худшем случае время поиска в глубину может быть максимальным, независимо от расстояния до начальной вершины.

СЛЕДСТВИЕ 3 *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в ширину, а* $D(u_1, 1)$, $D(u_1, 2), \ldots$ *— ярусы графа относительно вершины* u_1. *Тогда*

$$\forall i \geqslant 1 \left(\sum_{j=0}^{d(u_1,u_i)-1} |D(u_1, j)| \leqslant i \leqslant \sum_{j=0}^{d(u_1,u_i)} |D(u_1, j)| \right) .$$

Другими словами, время поиска в ширину ограничено снизу количеством вершин во всех ярусах, находящихся на расстоянии меньшем, чем расстояние от начальной вершины до текущей, и ограничено сверху количеством вершин в ярусах, начиная с яруса текущей вершины и включая все меньшие ярусы.

СЛЕДСТВИЕ 4 *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в ширину, а* $(v_1, \ldots, v_j, \ldots, v_p)$ *— обход при поиске в глубину, где* $u_i = v_j$. *Тогда в среднем* $i = 2j$.

Другими словами, поиск в глубину в среднем вдвое быстрее, чем поиск в ширину.

7.5. Орграфы и бинарные отношения

Целью заключительного раздела данной главы является установление связи теории графов с другими разделами дискретной математики.

7.5.1. Графы и отношения

Любой орграф $G(V, E)$ с петлями, но без кратных дуг, задаёт бинарное отношение E на множестве V, и обратно. А именно, пара элементов принадлежит отношению $(a, b) \in E \subset V \times V$ тогда и только тогда, когда в графе G есть дуга (a, b). Полный граф соответствует универсальному отношению. Граф (неориентированный) соответствует симметричному отношению. Дополнение графов есть дополнение отношений. Изменение направления всех дуг соответствует обратному отношению и т. д.

ОТСТУПЛЕНИЕ

Таким образом, имеется полная аналогия между орграфами и бинарными отношениями — фактически, это один и тот же класс объектов, только описанный разными средствами. Отношения (в частности, функции) являются базовым средством для построения подавляющего большинства математических моделей, используемых при решении практических задач. С другой стороны, графы допускают наглядное представление в виде диаграмм. Этим обстоятельством объясняется широкое использование диаграмм различного вида (которые суть представления графов или родственных объектов) при кодировании и особенно при проектировании в программировании.

7.5.2. Достижимость и частичное упорядочение

В качестве примера связи между орграфами и бинарными отношениями рассмотрим отношения частичного порядка с точки зрения теории графов. Узел u в орграфе $G(V, E)$ *достижим* из узла v, если существует путь из v в u. Путь из v в u обозначим $\langle \vec{v,u} \rangle$. Отношение достижимости можно представить матрицей T : **array** $[1..p, 1..p]$ **of** 0..1, где $T[i,j] = 1$, если узел v_j достижим из узла v_i, и $T[i,j] = 0$, если узел v_j недостижим из узла v_i. Рассмотрим отношение строгого частичного порядка $\succ$, которое характеризуется следующими аксиомами:

1. Антирефлексивность: $\forall v \in V \ (\neg(v \succ v))$.
2. Транзитивность: $\forall u, v, w \ ((v \succ w \ \& \ w \succ u \Longrightarrow v \succ u))$.
3. Антисимметричность: $\forall u, v \ (\neg(u \succ v \ \& \ v \succ u))$.

Отношению строгого частичного порядка $\succ \subset V \times V$ можно сопоставить орграф $G(V, E)$, в котором $a \succ b \Longleftrightarrow (a, b) \in E$.

ТЕОРЕМА 1 *Если отношение E есть строгое частичное упорядочение, то орграф $G(V, E)$ не имеет контуров.*

Доказательство От противного. Пусть в G есть контур. Рассмотрим любую дугу (a, b) в этом контуре. Тогда имеем $a \succ b$, по $b \succ a$ по транзитивности, что противоречит антисимметричности упорядочения. □

ТЕОРЕМА 2 *Если орграф $G(V, E)$ не имеет контуров, то отношение достижимости есть строгое частичное упорядочение.*

Доказательство

[Антирефлексивность] Нет контуров, следовательно, нет петель.

[Транзитивность] Если существуют пути из v в w и из w в u, то существует и путь из v в u.

[Антисимметричность] От противного. Пусть $\exists u, v \ (u \succ v \ \& \ v \succ u)$, то есть существует путь $\langle \vec{v,u} \rangle$ из v в u и путь $\langle \vec{u,v} \rangle$ из u в v. Следовательно, существует контур вида $\langle \vec{u,v} \rangle + \langle \vec{v,u} \rangle$, что противоречит условию. □

ТЕОРЕМА 3 *Если орграф не имеет контуров, то в нем есть узел, полустепень захода которого равна 0.*

Доказательство От противного. Пусть такого узла нет, тогда для любого узла найдётся узел, из которого есть дуга в данный узел. Следовательно, имеем контур против направления стрелок. □

ЗАМЕЧАНИЕ

Эта теорема позволяет найти минимальный элемент в конечном частично упорядоченном множестве, который требуется в алгоритме топологической сортировки (алгоритм 1.12). А именно, минимальный элемент — это источник, то есть узел, которому в матрице смежности соответствует нулевой столбец.

7.5.3. Транзитивное замыкание

Если E — бинарное отношение на V, то транзитивным замыканием (см. 1.5.1) E^+ на V будет отношение достижимости на орграфе $G(V, E)$.

ТЕОРЕМА *Пусть M — матрица смежности орграфа $G(V, E)$. Тогда $M^k[i, j] = 1$ в том и только в том случае, если существует путь длиной k из узла v_i в узел v_j.*

Доказательство Индукция по k. База: $k = 1$, $M^1 = M$ — пути длины 1. Пусть M^{k-1} содержит пути длины $k - 1$. Тогда

$$M^k[i, j] = \bigvee_{l=1}^{p} \left(M^{k-1}[i, l] \;\&\; M[l, j]\right),$$

то есть путь длины k из узла i в узел j существует тогда и только тогда, когда найдётся узел l, такой, что существует путь длины $k - 1$ из i в l и дуга (l, j), то есть

$$\exists \langle \vec{i, j} \rangle \;\; \left(|\langle \vec{i, j} \rangle|\right) = k \Longleftrightarrow \exists l \;\; \left(\left(\exists \langle \vec{i, l} \rangle \;\; \left(|\langle \vec{i, l} \rangle| = k - 1\right) \;\&\; (l, j) \in E\right)\right). \qquad \square$$

Если T — матрица достижимости, то очевидно, что

$$T = \bigvee_{k=1}^{p-1} M^k.$$

Трудоёмкость прямого вычисления по этой формуле составит $O(p^4)$. Матрица достижимости T может быть вычислена по матрице смежности M алгоритмом Уоршалла (алгоритм 1.11) за $O(p^3)$.

Комментарии

Эта глава является вводной главой той части книги, которая посвящена теории графов, поэтому здесь уместно привести краткий обзор учебной литературы по теории графов. Классическими учебниками по теории графов являются книги [18] и [28]. Первая является библиографической редкостью, а последняя монография переиздавалась в нашей стране и является образцовой по глубине и широте охвата материала и одновременно по ясности и лаконичности изложения. Терминология и обозначения книги [28] взяты за основу в данном учебнике. Книга [28], хотя и имеет сравнительно небольшой объём, содержит большое число прекрасных упражнений и различные сведения справочного характера. В меньшей степени в ней представлены программистские аспекты теории графов. Существуют и другие доступные учебники по теории графов: [5] и [25]. Последняя книга доступна даже неподготовленному читателю, имеет небольшой объём, но в то же время вполне достаточна для первоначального знакомства. Разделы, посвящённые теории графов, как правило, присутствуют во всех учебниках по дискретной математике, хотя такие разделы, по естественным причинам, часто не отличаются полнотой охвата материала. Особого упоминания заслуживают книги [21], [4] и [20]. Содержание этих книг абсолютно точно соответствует их названиям и является необходимым для программиста дополнением к классическим монографиям типа [28]. Эти источники, особенно [21], в значительной мере повлияли на отбор материала для нашего учебника. Монография [20] отличается большим объёмом и энциклопедической полнотой изложения. Единственный в этой главе алгоритм 7.1 носит настолько общеизвестный характер, что трудно указать его источник. Идея этого алгоритма лежит в основе огромного числа конкретных алгоритмов на графах. Как правило, она предполагается заранее известной читателю, и изложение сразу погружается в технические детали применения общей концепции поиска в ширину и в глубину для решения конкретной задачи. В то же время свойства алгоритма поиска, приведённые в виде следствий к теореме, обосновывающей алгоритм 7.1, хотя и являются вполне тривиальными, не всегда осознаются практическими программистами при решении задач. Именно поэтому представляется важным предпослать обсуждению конкретных алгоритмов на графах изложение общей идеи в предельно простой и рафинированной форме.

Упражнения

7.1. Построить пример графов G_1 и G_2, для которых $p_1 = p_2$, $q_1 = q_2$, $\delta_1 = \delta_2$, $\Delta_1 = \Delta_2$, но $G_1 \not\sim G_2$ (кроме примера подраздела 7.1.6).

7.2. Доказать, что в любом нетривиальном графе всегда существуют вершины одинаковой степени.

7.3. *Задача Рамсея*[1]. Доказать, что среди любых 6 человек есть 3 попарно знакомых или 3 попарно незнакомых.

[1] Фрэнк Рамсей (1903-1930).

7.4. Рассмотрим матрицу смежности рёбер Q : **array** $[1..q, 1..q]$ **of** 0..1, где

$$Q[i,j] = \begin{cases} 1, & \text{если ребро } i \text{ смежно с ребром } j, \\ 0, & \text{в противном случае.} \end{cases}$$

Является ли матрица смежности рёбер Q представлением в компьютере графа $G(V, E)$?

7.5. Описать в терминах теории графов отношение эквивалентности на конечном множестве.

Глава 8 Связность

В этой главе обсуждается важное для приложений понятие связности, доказывается фундаментальная теорема — теорема Менгера[1] и рассматриваются наиболее популярные алгоритмы поиска кратчайших путей.

8.1. Компоненты связности

В русском языке есть как слово «компонент» мужского рода, так и слово «компонента» женского рода, оба варианта допустимы. В современной языковой практике чаще используется слово мужского рода. Однако исторически сложилось так, что «компонента связности» имеет женский род, и в данном случае мы подчиняемся традиции.

8.1.1. Объединение графов и компоненты связности

ТЕОРЕМА *Граф связен тогда и только тогда, когда его нельзя представить в виде объединения двух графов.*

Доказательство

[$\Longrightarrow$] От противного. Пусть $k(G) = 1$ и граф G состоит из двух компонент, то есть $G = G_1(V_1, E_1) \cup G_2(V_2, E_2)$, где $V_1 \cap V_2 = \varnothing$, $E_1 \cap E_2 = \varnothing$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$. Возьмём $v_1 \in V_1$, $v_2 \in V_2$. Тогда $\exists \langle v_1, v_2 \rangle$ $(v_1 \in V_1 \ \& \ v_2 \in V_2)$. В этой цепи $\exists e = (a, b)$ $(a \in V_1 \ \& \ b \in V_2)$. Но тогда $e \notin E_1$, $e \notin E_2$, следовательно, $e \notin E$.

[$\Longleftarrow$] От противного. Пусть $k(G) > 1$. Тогда существуют две вершины u и v, не соединённые цепью. Следовательно, вершины u и v принадлежат разным компонентам связности. Положим G_1 — компонента связности для u, а G_2 — все остальное. Имеем $G = G_1 \cup G_2$. □

[1] Карл Менгер (1902–1985).

ЗАМЕЧАНИЕ

Несвязный граф всегда можно представить как объединение связных компонент. Эти компоненты можно рассматривать независимо. Поэтому во многих случаях можно без ограничения общности предполагать, что рассматриваемый граф связен.

8.1.2. Точки сочленения, мосты и блоки

Вершина графа называется *точкой сочленения*, или *разделяющей вершиной*, если её удаление увеличивает число компонент связности. *Мостом* называется ребро, удаление которого увеличивает число компонент связности. *Блоком* называется связный граф, не имеющий точек сочленения.

Пример В графе, диаграмма которого представлена на рис. 8.1:

1) вершины u, v — точки сочленения, и других точек сочленения нет;
2) ребро x — мост, и других мостов нет;
3) правильные подграфы $\{a, b, w\}$, $\{b, u, w\}$, $\{a, b, u, w\}$, $\{c, d, v\}$, $\{e, f, v\}$ — блоки, и других блоков нет.

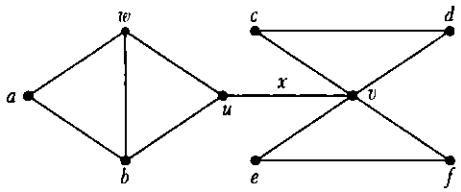

Рис. 8.1. Точки сочленения, мосты и блоки

ЛЕММА *В любом нетривиальном графе есть по крайней мере две вершины, которые не являются точками сочленения.*

Доказательство Например, таковыми являются концы любого диаметра. □

ТЕОРЕМА 1 *Пусть $G(V, E)$ — связный граф и $v \in V$. Тогда следующие утверждения эквивалентны:*

1. *v — точка сочленения.*
2. $\exists u, w \in V \ (u \neq w \ \& \ \forall \ \langle u, w\rangle_G \ (v \in \langle u, w\rangle))$.
3. $\exists U, W \ (U \cap W = \varnothing \ \& \ U \cup W = V - v \ \& \ \forall u \in U, w \in W \ (\forall \langle u, w\rangle_G \ (v \in \langle u, w\rangle_G)))$.

Доказательство

[1⟹3] Рассмотрим $G' := G - v$. Граф G' не связен, $k(G') > 1$, следовательно, G' имеет по крайней мере две компоненты связности, то есть $V - v = U \cup W$, где U — множество вершин одной из компонент связности, а W — все остальные вершины. Пусть $u \in U$, $w \in W$. Тогда $\neg\exists\langle u, w\rangle_{G'}$. Но $k(G) = 1$, следовательно, $\exists\langle u, w\rangle_G$, и значит, $\forall\, \langle u, w\rangle_G \ (v \in \langle u, w\rangle_G)$.

[3⟹2] Тривиально.

[2⟹1] Рассмотрим $G' := G - v$. Имеем: $\neg\exists\langle u, w\rangle_{G'}$. Следовательно, $k(G') > 1$, откуда v — точка сочленения. □

СЛЕДСТВИЕ *Если вершина инцидентна мосту и не является висячей, то она является точкой сочленения.*

Доказательство Пусть (u, v) — мост и $d(v) > 1$. Тогда v смежна с w, причём $w \neq u$. Далее разбор случаев. Если $\exists\ \langle u, w\rangle\ (v \notin \langle u, w\rangle)$, то ребро (u, v) принадлежит циклу, что противоречит тому, что (u, v) — мост. Если же $\neg\exists\ \langle u, w\rangle\ (v \notin \langle u, w\rangle)$, то $\forall\ \langle u, w\rangle\ (v \in \langle u, w\rangle)$, и значит, v — точка сочленения по пункту 2 теоремы. □

ЗАМЕЧАНИЕ

Отнюдь не всякая точка сочленения является концом моста.

ТЕОРЕМА 2 *Пусть $G(V, E)$ — связный граф и $x \in E$. Тогда следующие утверждения эквивалентны:*

1. *x — мост.*
2. *x не принадлежит ни одному простому циклу.*
3. $\exists u, w \in V\ (\forall\langle u, w\rangle_G\ (x \in \langle u, w\rangle_G))$.
4. $\exists U, W\ (U \cap W = \varnothing\ \&\ U \cup W = V\ \&\ \forall u \in U\ (\forall w \in W\ (\forall\langle u, w\rangle_G\ (x \in \langle u, w\rangle)_G)))$.

Доказательство Упражнение 8.1б. □

8.1.3. Вершинная и рёберная связность

При взгляде на диаграммы связных графов (сравните, например, рис. 7.5 и 8.1) возникает естественное впечатление, что связный граф может быть «более» или «менее» связен. В этом подразделе рассматриваются некоторые количественные меры связности.

Вершинной связностью графа G (обозначается $\varkappa(G)$) называется наименьшее число вершин, удаление которых приводит к несвязному или тривиальному графу.

Пример

$\varkappa(G) = 0$, если G несвязен;
$\varkappa(G) = 1$, если G имеет точку сочленения;
$\varkappa(K_p) = p - 1$, если K_p — полный граф.

Граф G называется *n-связным*, если $\varkappa(G) = n$.

Рёберной связностью графа G (обозначается $\lambda(G)$) называется наименьшее число рёбер, удаление которых приводит к несвязному или тривиальному графу.

Пример

$\lambda(G) = 0$, если G несвязен;
$\lambda(G) = 1$, если G имеет мост;
$\lambda(K_p) = p - 1$, если K_p — полный граф.

ТЕОРЕМА $\varkappa(G) \leqslant \lambda(G) \leqslant \delta(G)$.

Доказательство

[$\varkappa \leqslant \lambda$] Если $\lambda = 0$, то $\varkappa = 0$. Если $\lambda = 1$, то есть мост, а значит, либо есть точка сочленения, либо $G = K_2$. В любом случае $\varkappa = 1$. Пусть теперь $\lambda \geqslant 2$. Тогда, удалив $\lambda - 1$ ребро, получим граф G', имеющий мост (u, v). Для каждого из удаляемых $\lambda - 1$ рёбер удалим инцидентную вершину, отличную от u и v. Если после такого удаления вершин граф несвязен, то $\varkappa \leqslant \lambda - 1 < \lambda$. Если связен, то удалим один из концов моста — u или v. Имеем $\varkappa \leqslant \lambda$.

[$\lambda \leqslant \delta$] Если $\delta = 0$, то $\lambda = 0$. Пусть вершина v имеет наименьшую степень: $d(v) = \delta$. Удалим все рёбра, инцидентные v. Тогда $\lambda \leqslant \delta$. □

8.1.4. Оценка числа рёбер

Легко заметить, что инварианты $p(G)$, $q(G)$ и $k(G)$ не являются независимыми.

ТЕОРЕМА *Пусть p — число вершин, q — число рёбер, k — число компонент связности графа. Тогда*

$$p - k \leqslant q \leqslant \frac{(p-k)(p-k+1)}{2}.$$

Доказательство

[$p - k \leqslant q$] Индукция по p. База: $p = 1$, $q = 0$, $k = 1$. Пусть оценка верна для всех графов с числом вершин меньше p. Рассмотрим граф G, $p(G) = p$. Удалим в G вершину v, которая не является точкой сочленения: $G' := G - v$. Тогда если v — изолированная вершина, то $p' = p - 1$, $k' = k - 1$, $q' = q$. Имеем $p - k = p' - k' \leqslant \leqslant q' = q$. Если v — не изолированная вершина, то $p' = p - 1$, $k' = k$, $q' < q$. Имеем: $p - k = 1 + p' - k' \leqslant 1 + q' \leqslant q$.

[$q \leqslant (p-k)(p-k+1)/2$] Метод выделения «критических» графов. Число рёбер q графа с p вершинами и k компонентами связности не превосходит числа рёбер в графе с p вершинами и k компонентами связности, в котором все компоненты связности суть полные графы. Следовательно, достаточно рассматривать только графы, в которых все компоненты связности полные. Среди всех графов с p вершинами и k полными компонентами связности наибольшее число рёбер имеет граф $\widetilde{K_p}$, в котором все рёбра сосредоточены в одной компоненте связности:

$$\widetilde{K_p} := K_{p-k+1} \cup \bigcup_{i=1}^{k-1} K_1.$$

Действительно, пусть есть две компоненты связности G_1 и G_2, такие, что $1 < p_1 = p(G_1) \leqslant p_2 = p(G_2)$. Если перенести одну вершину из компоненты связности G_1 в компоненту связности G_2, то число рёбер изменится на $\Delta q = p_2 - (p_1 - 1) = p_2 - p_1 + 1 > 0$.

Следовательно, число рёбер возросло. Таким образом, достаточно рассмотреть граф $\widetilde{K_p}$. Имеем:

$$q(\widetilde{K_p}) = (p-k+1)(p-k+1-1)/2 + 0 = (p-k)(p-k+1)/2. \qquad \square$$

СЛЕДСТВИЕ 1 *Если $q > (p-1)(p-2)/2$, то граф связен.*

Доказательство Рассмотрим неравенство $(p-1)(p-2)/2 < q \leqslant (p-k)(p-k+1)/2$ при различных значениях k.

$k = 1$	$(p-1)(p-2)/2 < (p-1)p/2$	выполняется,
$k = 2$	$(p-1)(p-2)/2 < (p-2)(p-1)/2$	не выполняется,
$k = 3$	$(p-1)(p-2)/2 < (p-3)(p-2)/2$	не выполняется

и т. д. $\square$

СЛЕДСТВИЕ 2 *В связном графе $p - 1 \leqslant q \leqslant p(p-1)/2$.*

Доказательство Следует из теоремы при $k = 1$. $\square$

8.2. Теорема Менгера

Интуитивно очевидно, что граф тем более связен (при использовании различных мер связности), чем больше существует различных цепей, соединяющих одну вершину с другой, и тем менее связен, чем меньше нужно удалить промежуточных вершин, чтобы отделить одну вершину от другой. В этом разделе устанавливается теорема Менгера, которая придает этим неформальным наблюдениям точный и строгий смысл.

8.2.1. Непересекающиеся цепи и разделяющие множества

Пусть $G(V, E)$ — связный граф, u и v — две его несмежные вершины. Две цепи $\langle u, v\rangle$ называются *вершинно-непересекающимися*, если у них нет общих вершин, отличных от u и v. Две цепи $\langle u, v\rangle$ называются *рёберно-непересекающимися*, если у них нет общих рёбер. Если две цепи вершинно не пересекаются, то они и рёберно не пересекаются. Обозначим через $P(u, v)$ множество вершинно-непересекающихся цепей $\langle u, v\rangle$.

Множество S вершин (и/или рёбер) графа G *разделяет* две вершины u и v, если u и v принадлежат разным компонентам связности графа $G-S$. *Разделяющее множество рёбер* называется *разрезом*. *Разделяющее множество вершин* для вершин u и v обозначим $S(u, v)$.

Для заданных вершин u и v множества $P(u, v)$ и $S(u, v)$ можно выбирать различным образом.

Из определений следует, что любое множество $P(u, v)$ вершинно-непересекающихся цепей обладает тем свойством, что

$$\bigcap_{\langle u,v\rangle \in P(u,v)} \langle u, v\rangle = \{u, v\},$$

а любое разделяющее множество вершин $S(u, v)$ обладает тем свойством, что

$$G - S = G_1 \cup G_2 \ \&\ v \in G_1 \ \&\ u \in G_2.$$

ЗАМЕЧАНИЕ

Если u и v принадлежат разным компонентам связности графа G, то $|P(u, v)| = 0$ и $|S(u, v)| = 0$. Поэтому без ограничения общности можно считать, что G — связный граф.

Пример В графе, диаграмма которого представлена на рис. 8.2, можно выбрать множества вершинно-непересекающихся путей $P_1 = \{\langle u, a, d, v\rangle, \langle u, b, e, v\rangle\}$ и $P_2 = \{\langle u, b, d, v\rangle, \langle u, c, e, v\rangle\}$. Заметим, что путь $\langle u, c, b, d, v\rangle$ образует множество P_3 вершинно-непересекающихся путей, состоящее из одного элемента. Множества вершин $S_1 = \{a, b, c\}$ и $S_2 = \{d, e\}$ являются разделяющими для вершин u и v.

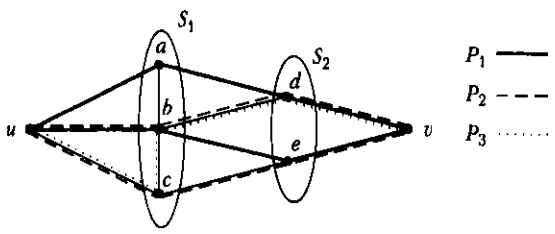

Рис. 8.2. Вершинно-непересекающиеся пути и разделяющие множества вершин

8.2.2. Теорема Менгера в «вершинной форме»

ТЕОРЕМА (Менгера) *Пусть u и v — несмежные вершины в графе G. Наименьшее число вершин в множестве, разделяющем u и v, равно наибольшему числу вершинно-непересекающихся простых $\langle u, v\rangle$-цепей:*

$$\max |P(u,v)| = \min |S(u,v)|.$$

ЗАМЕЧАНИЕ

Пусть G — связный граф и вершины u и v несмежны. Обозначим $P := P(u,v)$, $S := S(u,v)$. Легко видеть, что $|P| \leqslant |S|$. Действительно, любая $\langle u,v\rangle$-цепь проходит через S. Если бы $|P| > |S|$, то в S была бы вершина, принадлежащая более чем одной цепи из P, что противоречит выбору P. Таким образом, $\forall P\ (\forall S\ (|P| \leqslant |S|))$. Следовательно, $\max |P| \leqslant \min |S|$. Утверждение теоремы состоит в том, что в любом графе *существуют* такие P и S, что достигается равенство $|P| = |S|$.

Доказательство Пусть G — связный граф, u и v — несмежные вершины. Совместная индукция по p и q. База: наименьший граф, удовлетворяющий условиям теоремы, состоит из трех вершин u, w, v и двух рёбер (u,w) и (w,v). В нём $P(u,v) = \{\langle u,w,v\rangle\}$ и $S(u,v) = \{w\}$. Таким образом, $|P(u,v)| = |S(u,v)| = 1$. Пусть утверждение теоремы верно для всех графов с числом вершин меньше p и/или числом рёбер меньше q. Рассмотрим граф G с p вершинами и q ребрами. Пусть $u, v \in V$, причём u, v — не смежны и S — некоторое наименьшее множество вершин, разделяющее u и v. Обозначим $n := |S|$. Рассмотрим три случая.

[1] Пусть в S есть вершины, несмежные с u и несмежные с v. Тогда граф $G - S$ состоит из двух *нетривиальных* графов G_1 и G_2. Образуем два новых графа G_u и G_v, стягивая нетривиальные графы G_1 и G_2 к вершинам u и v соответственно: $G_u := G/G_1$, $G_v := G/G_2$ (рис. 8.3).

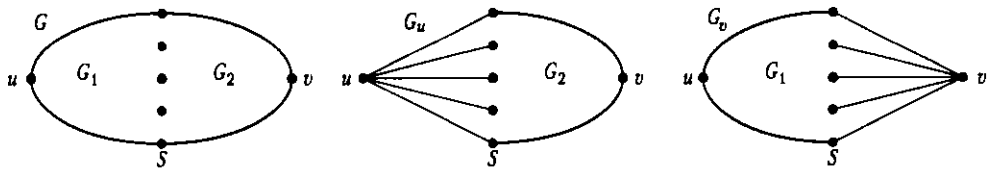

Рис. 8.3. К доказательству теоремы Менгера, случай 1

S по-прежнему является наименьшим разделяющим множеством для u и v как в G_u, так и в G_v. Так как G_1 и G_2 нетривиальны, то G_u и G_v имеют меньше вершин и/или рёбер, чем G. Следовательно, по индукционному предположению в G_u и в G_v имеется n вершинно-непересекающихся простых цепей. Комбинируя отрезки этих цепей от u до S и от S до v, получаем n вершинно-непересекающихся простых цепей в G.

[2] Пусть все вершины S смежны с u или с v (для определённости пусть с u) и среди вершин S есть вершина w, смежная одновременно с u и с v (рис. 8.4).

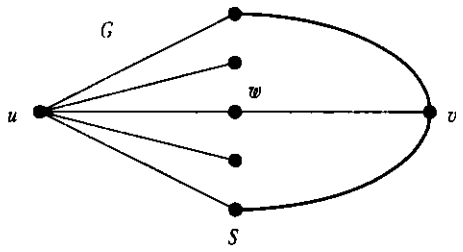

Рис. 8.4. К доказательству теоремы Менгера, случай 2

Рассмотрим граф $G' := G - w$. В нем $S - w$ — разделяющее множество для u и v, причём наименьшее. По индукционному предположению в G' имеется $|S - w| = n - 1$ вершинно-непересекающихся простых цепей. Добавим к ним цепь $\langle u, w, v\rangle$. Она простая и вершинно не пересекается с остальными. Таким образом, имеем n вершинно-непересекающихся простых цепей в G.

[3] Пусть теперь все вершины S смежны с u или с v (для определённости пусть с u) и среди вершин S нет вершин, смежных одновременно с вершиной u и с вершиной v. Рассмотрим *кратчайшую* $\langle u, v\rangle$-цепь $\langle u, w_1, w_2, \ldots, v\rangle$, $w_1 \in S$, $w_2 \neq v$ (рис. 8.5).

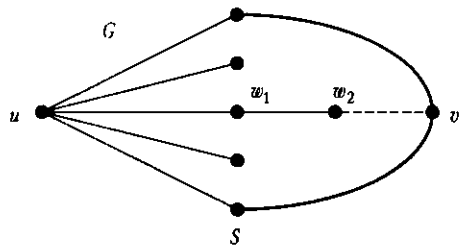

Рис. 8.5. К доказательству теоремы Менгера, случай 3

Рассмотрим граф $G' := G/\{w_1, w_2\}$, полученный из G склейкой вершин w_1 и w_2 в вершину w_1. Имеем $w_2 \notin S$, в противном случае цепь $\langle u, w_2, \ldots, v\rangle$ была бы ещё более короткой. Следовательно, в графе G' множество S по-прежнему является наименьшим, разделяющим u и v, и граф G' имеет по крайней мере на одно ребро меньше. По индукционному предположению в G' существуют n вершинно-непересекающихся простых цепей. Но цепи, которые не пересекаются в G', не пересекаются и в G. Таким образом, получаем n вершинно-непересекающихся простых цепей в G. □

8.2.3. Варианты теоремы Менгера

Теорема Менгера представляет собой весьма общий факт, который в разных формулировках встречается в различных областях математики. Комбинируя вершинно- и рёберно-непересекающиеся цепи, разделяя не отдельные вершины, а множества вершин, используя инварианты $\varkappa$ и λ, можно сформулировать и другие утверждения, подобные теореме Менгера. Например:

ТЕОРЕМА *Для любых двух несмежных вершин u и v графа G наибольшее число рёберно-непересекающихся (u,v)-цепей равно наименьшему числу рёбер в (u,v)-разрезе.*

ТЕОРЕМА *Чтобы граф G был n-связным, необходимо и достаточно, чтобы любые две несмежные вершины были соединены не менее чем n вершинно-непересекающимися простыми цепями.*

Другими словами, в любом графе G любые две несмежные вершины соединены не менее чем $\varkappa(G)$ вершинно-непересекающимися простыми цепями.

Доказательство подобных утверждений требует места, и мы оставляем их за пределами книги. В последующих разделах приведены и некоторые другие результаты, в которых проявляется теорема Менгера.

8.3. Теорема Холла

Теорема Холла[1], устанавливаемая в этом разделе, имеет множество различных применений и интерпретаций, с рассмотрения которых мы и начинаем её изложение.

8.3.1. Задача о свадьбах

Пусть имеется конечное множество юношей, каждый из которых знаком с некоторым подмножеством конечного множества девушек. В каком случае всех юношей можно женить так, чтобы каждый женился на знакомой девушке?

8.3.2. Трансверсаль

Пусть $S = \{S_1, \ldots, S_m\}$ — семейство подмножеств конечного множества E. Подмножества S_k не обязательно различны и могут пересекаться. *Системой различных представителей* в семействе S (или *трансверсалью* в семействе S) называется любое подмножество $C = \{c_1, \ldots, c_m\}$ из m элементов множества E, таких, что $\forall\, k \in 1..m\ \ (c_k \in S_k)$. В каком случае существует трансверсаль?

ЗАМЕЧАНИЕ

Все элементы множества C различны, откуда и происходит название «система различных представителей».

[1] Маршалл Холл (1910–1990).

8.3.3. Совершенное паросочетание

Паросочетанием (или *независимым множеством рёбер*) называется множество рёбер, в котором никакие два ребра не смежны. Паросочетание называется *максимальным*, если никакое его надмножество не является независимым.

Пусть $G(V_1, V_2, E)$ — двудольный граф. *Совершенным паросочетанием* из V_1 в V_2 называется паросочетание, покрывающее вершины V_1. В каком случае существует совершенное паросочетание из V_1 в V_2?

ЗАМЕЧАНИЕ

Совершенное паросочетание является максимальным.

8.3.4. Теорема Холла — формулировка и доказательство

Вообще говоря, задачи 8.3.1, 8.3.2 и 8.3.3 — это одна и та же задача. Действительно, задача 8.3.1 сводится к задаче 8.3.3 следующим образом. V_1 — множество юношей, V_2 — множество девушек, рёбра — знакомства юношей с девушками. В таком случае совершенное паросочетание — искомый набор свадеб. Задача 8.3.2 сводится к задаче 8.3.3 следующим образом. Положим $V_1 := S$, $V_2 := E$, ребро (S_k, e_i) существует, если $e_i \in S_k$. В таком случае совершенное паросочетание — искомая трансверсаль. Таким образом, задачи 8.3.1, 8.3.2 и 8.3.3 имеют общий ответ: в том и только том случае, когда

$$\text{любые } k \begin{pmatrix} \text{юношей} \\ \text{подмножеств} \\ \text{вершин из } V_1 \end{pmatrix} \text{ в совокупности } \begin{pmatrix} \text{знакомы с} \\ \text{содержат} \\ \text{смежны с} \end{pmatrix}$$

$$\text{не менее чем } k \begin{pmatrix} \text{девушками} \\ \text{элементов} \\ \text{вершинами из } V_2 \end{pmatrix},$$

что устанавливается следующей теоремой.

ТЕОРЕМА (Холла) *Пусть $G(V_1, V_2, E)$ — двудольный граф. Совершенное паросочетание из V_1 в V_2 существует тогда и только тогда, когда $\forall A \subset V_1$ $(|A| \leqslant |\Gamma(A)|)$.*

Доказательство

[$\Longrightarrow$] Пусть существует совершенное паросочетание из V_1 в V_2. Тогда в $\Gamma(A)$ входит $|A|$ вершин из V_2, парных к вершинам из множества A, и, возможно, еще что-то. Таким образом, $|A| \leqslant |\Gamma(A)|$.

[$\Longleftarrow$] Добавим в G две новые вершины u и v, так что вершина u смежна со всеми вершинами из V_1, а вершина v смежна со всеми вершинами из V_2. Совершенное паросочетание из V_1 в V_2 существует тогда и только тогда, когда существуют $|V_1|$ вершинно-непересекающихся простых $\langle u, v\rangle$-цепей (рис. 8.6). Ясно, что $|P(u, v)| \leqslant |V_1|$ (так как V_1 разделяет вершины u и v).

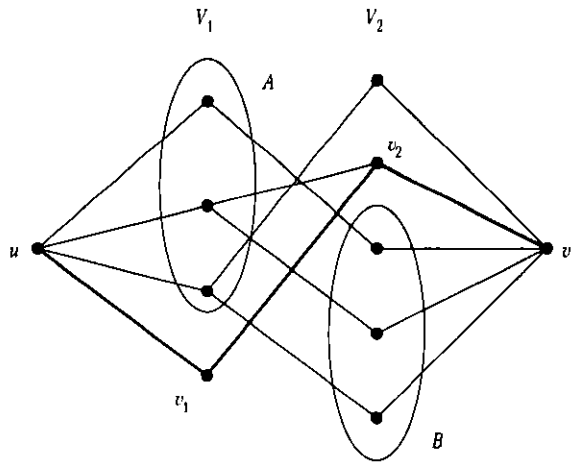

Рис. 8.6. К доказательству теоремы Холла

По теореме Менгера $\max|P(u,v)| = \min|S(u,v)| = |S|$, где S — наименьшее множество, разделяющее вершины u и v. Имеем $|S| \leqslant |V_1|$. Покажем, что $|S| \geqslant |V_1|$. Пусть $S = A \cup B$, $A \subset V_1$, $B \subset V_2$. Тогда $\Gamma(V_1 \setminus A) \subset B$. Действительно, если бы $\Gamma(V_1 \setminus A) \not\subset B$, то существовал бы «обходной» путь $\langle u, v_1, v_2, v\rangle$ (см. рис. 8.6) и S не было бы разделяющим множеством для u и v. Имеем: $|V_1 \setminus A| \leqslant |\Gamma(V_1 \setminus A)| \leqslant |B|$. Следовательно, $|S| = |A| + |B| \geqslant |A| + |V_1 \setminus A| = |V_1|$. □

8.4. Потоки в сетях

Рассмотрим некоторые примеры практических задач.

Примеры

1. Пусть имеется сеть автомобильных дорог, по которым можно проехать из пункта A в пункт B. Дороги могут пересекаться в промежуточных пунктах. Количество автомобилей, которые могут проехать по каждому отрезку дороги за единицу времени, не безгранично: оно определяется такими факторами, как ширина проезжей части, качество дорожного покрытия, действующие ограничения скорости движения и т. д. (обычно это называют «пропускной способностью» дороги). Каково максимальное количество автомобилей, которые могут проехать из пункта A в пункт B за единицу времени без образования пробок на дорогах (обычно это называют «автомобильным потоком»)? Можно поставить и другой вопрос: какие дороги и насколько нужно расширить или улучшить, чтобы увеличить максимальный автомобильный поток на заданную величину?
2. Пусть имеется сеть трубопроводов, соединяющих пункт A (скажем, нефтепромысел) с пунктом B (скажем, нефтеперерабатывающим заводом). Трубопроводы могут соединяться и разветвляться в промежуточных пунктах. Количество

нефти, которое может быть перекачано по каждому отрезку трубопровода в единицу времени, также не безгранично и определяется такими факторами, как диаметр трубы, мощность нагнетающего насоса и т. д. (обычно это называют «пропускной способностью» или «максимальным расходом» трубопровода). Сколько нефти можно прокачать через такую сеть за единицу времени?

3. Пусть имеется система машин для производства готовых изделий из сырья и последовательность технологических операций может быть различной (то есть операции могут выполняться на разном оборудовании или в разной последовательности), но все допустимые варианты заранее строго определены. Максимальная производительность каждой единицы оборудования, естественно, также заранее известна и постоянна. Какова максимально возможная производительность всей системы в целом, и как нужно организовать производство для достижения максимальной производительности?

Изучение этих и многочисленных подобных им практических задач приводит к теории потоков в сетях. В данном разделе рассматривается решение только одной (но самой существенной) задачи этой теории, а именно: задачи определения максимального потока. Описание других родственных задач, например, задачи определения критического пути, можно найти в литературе, перечисленной в конце главы.

8.4.1. Определение потока

Пусть $G(V, E)$ — сеть, s и t — соответственно, источник и сток сети. Дуги сети нагружены неотрицательными вещественными числами, c: $E \to \mathbb{R}^+$. Если u и v — узлы сети, то число $c(u, v)$ — называется *пропускной способностью* дуги (u, v).

ЗАМЕЧАНИЕ

Матрица пропускных способностей C : **array** $[1..p, 1..p]$ **of real** является представлением сети, аналогичным матрице смежности. Нулевое значение элемента $C[u, v]$ соответствует дуге с нулевой пропускной способностью, то есть отсутствию дуги, а положительное значение элемента $C[u, v]$ соответствует дуге с ненулевой пропускной способностью, то есть дуга присутствует.

Пусть задана функция f: $E \to \mathbb{R}$. *Дивергенцией* функции f в узле v называется число $\mathrm{div}(f, v)$, которое определяется следующим образом:

$$\mathrm{div}(f, u) \stackrel{\mathrm{Def}}{=} \sum_{\{v|(u,v)\in E\}} f(u, v) - \sum_{\{v|(v,u)\in E\}} f(v, u).$$

ЗАМЕЧАНИЕ

В физике дивергенция обычно определяется наоборот: то, что пришло, минус то, что ушло. Но в данном случае удобнее, чтобы дивергенция источника была положительна.

Функция $f\colon E \to \mathbb{R}$ называется *потоком* в сети G, если:

1. $\forall (u,v) \in E \ (0 \leqslant f(u,v) \leqslant c(u,v))$, то есть поток через дугу неотрицателен и не превосходит пропускной способности дуги;
2. $\forall u \in V \setminus \{s,t\} \ (\mathrm{div}(f,u) = 0)$, то есть дивергенция потока равна нулю во всех узлах, кроме источника и стока.

Число $w(f) \stackrel{\text{Def}}{=} \mathrm{div}(f,s)$ называется *величиной* потока f. Если $f(u,v) = c(u,v)$, то дуга (u,v) называется *насыщенной*.

8.4.2. Разрезы

Пусть P — (s,t)-разрез, $P \subset E$. Всякий разрез определяется разбиением множества вершин V на два подмножества S и T так, что $S \subset V$, $T \subset V$, $S \cup T = V$, $S \cap T = \varnothing$, $s \in S$, $t \in T$, а в P попадают все дуги, соединяющие S и T. Тогда $P = P^+ \cup P^-$, где P^+ — дуги от S к T, P^- — дуги от T к S. Сумма потоков через дуги разреза P обозначается $F(P)$. Сумма пропускных способностей дуг разреза P называется *пропускной способностью* разреза и обозначается $C(P)$:

$$F(P) \stackrel{\text{Def}}{=} \sum_{e \in P} f(e), \qquad C(P) \stackrel{\text{Def}}{=} \sum_{e \in P} c(e).$$

Аналогично, $F(P^+)$ и $F(P^-)$ — суммы потоков через соответствующие части разрезов.

8.4.3. Теорема Форда–Фалкерсона

В этом подразделе устанавливается количественная связь между величиной максимального потока и пропускными способностями дуг сети.

ЛЕММА 1 $w(f) = F(P^+) - F(P^-)$.

Доказательство Рассмотрим сумму $W := \sum_{v \in S} \mathrm{div}(f,v)$. Пусть дуга $(u,v) \in E$. Если $u,v \in S$, то в сумму W попадают два слагаемых для этой дуги: $+f(u,v)$ при вычислении $\mathrm{div}(f,u)$ и $-f(u,v)$ при вычислении $\mathrm{div}(f,v)$, итого 0. Если $u \in S$, $v \in T$, то в сумму W попадает одно слагаемое $+f(u,v)$, все такие слагаемые дают $F(P^+)$. Если $u \in T$, $v \in S$, то в сумму W попадает одно слагаемое $-f(u,v)$, все такие слагаемые дают $F(P^-)$. Таким образом, $W = F(P^+) - F(P^-)$. С другой стороны, $W = \sum_{v \in S} \mathrm{div}(f,v) = \mathrm{div}(f,s) = w(f)$. □

ЛЕММА 2 $\mathrm{div}(f,s) = -\mathrm{div}(f,t)$.

Доказательство Рассмотрим разрез $P := (S,T)$, где $S := V - t$, а $T := \{t\}$. Имеем:

$$\mathrm{div}(f,s) = w(f) = F(P^+) - F(P^-) = F(P^+) = \sum_v f(v,t) = -\mathrm{div}(f,t). \qquad \square$$

ЛЕММА 3 $w(f) \leqslant F(P)$.

Доказательство $w(f) = F(P^+) - F(P^-) \leqslant F(P^+) \leqslant F(P)$. □

ЛЕММА 4 $\max\limits_f w(f) \leqslant \min\limits_P C(P)$.

Доказательство По предыдущей лемме $w(f) \leqslant F(P)$, следовательно, $\max\limits_f w(f) \leqslant$ $\leqslant \min\limits_P F(P)$. По определению $F(P) \leqslant C(P)$, следовательно, $\min\limits_P F(P) \leqslant \min\limits_P C(P)$. Имеем: $\max\limits_f w(f) \leqslant \min\limits_P C(P)$. □

ТЕОРЕМА (Форда[1]—Фалкерсона[2]) *Максимальный поток в сети равен минимальной пропускной способности разреза, то есть существует поток f^*, такой, что*

$$w(f^*) = \max_f w(f) = \min_P C(P).$$

Доказательство Пусть f — некоторый максимальный поток. Покажем, что существует разрез P, такой, что $w(f) = C(P)$. Рассмотрим граф G', полученный из сети G забыванием ориентации дуг. Построим множество вершин S следующим образом:

$$\begin{aligned} S \stackrel{\text{Def}}{=} \{u \in V \mid \exists\, \langle s, u\rangle \in G'\ \forall (u_i, u_{i+1}) \in \langle s, u\rangle \in G' \\ ((u_i, u_{i+1}) \in E \Longrightarrow f(u_i, u_{i+1}) < C(u_i, u_{i+1})\ \& \\ \&\,(u_{i+1}, u_i) \in E \Longrightarrow f(u_{i+1}, u_i) > 0)\}, \end{aligned}$$

то есть вдоль пути $\langle s, u\rangle$ дуги в направлении пути не насыщены, а дуги против направления пути имеют положительный поток. Такая цепь $\langle s, u\rangle$ называется *аугментальной*. Имеем $s \in S$ по построению. Следовательно, $S \neq \varnothing$. Положим $T := V \setminus S$. Покажем, что $t \in T$. Действительно, пусть $t \in S$. Тогда существует аугментальная цепь $\langle s, t\rangle$, обозначим её R. Но тогда можно найти число $\delta > 0$:

$$\delta := \min_{e \in R} \Delta(e), \text{ где } \Delta(e) := \begin{cases} c(e) - f(e) > 0, & \text{если } e \text{ ориентировано вдоль } R, \\ f(e) > 0, & \text{если } e \text{ ориентировано против } R. \end{cases}$$

В этом случае можно увеличить величину потока на δ, изменив поток для всех дуг аугментальной цепи:

$$f(e) := \begin{cases} f(e) + \delta, & \text{если } e \text{ ориентировано вдоль } R, \\ f(e) - \delta, & \text{если } e \text{ ориентировано против } R. \end{cases}$$

При этом условия потока выполняются: $0 \leqslant f(e) \leqslant C(e)$, $\operatorname{div}(v) = 0$.

---

[1] Лестер Рендальф Форд мл. (р. 1927).

[2] Дальберт Рей Фалкерсон (1924–1976).

Таким образом, поток f увеличен на величину δ, что противоречит максимальности потока f. Имеем $t \in T$ и $T \neq \varnothing$. Следовательно, S и T определяют разрез $P = P^+ \cup P^-$. В этом разрезе все дуги e^+ насыщены ($f(e^+) = c(e^+)$), а все дуги e^- не нагружены ($f(e^-) = 0$), иначе S можно было бы расширить. Имеем: $w(f) = F(P^+) - F(P^-) = C(P^+)$, таким образом, P^+ — искомый разрез. □

8.4.4. Алгоритм нахождения максимального потока

Следующий алгоритм определяет максимальный поток в сети, заданной матрицей пропускных способностей дуг. Этот алгоритм использует идею доказательства теоремы Форда–Фалкерсона, а именно, задавшись начальным приближением потока, определяем множество вершин S, которые соединены аугментальными цепями с источником s. Если оказывается, что $t \in S$, то это означает, что поток не максимальный и его можно увеличить на величину δ. Для определения аугментальных цепей и одновременного подсчета величины δ в алгоритме использована вспомогательная структура данных:

```
P : array [1..p] of record
  s : enum (−,+) { «знак», то есть направление дуги }
  n : 1..p { предшествующая вершина в аугментальной цепи }
  δ : real { величина возможного увеличения потока }
end record
N : array [1..p] of 0..1 { отметка узла }
S : array [1..p] of 0..1 { признак принадлежности вершины множеству S}
```

Алгоритм 8.1 Нахождение максимального потока

Вход: сеть $G(V, E)$ с источником s и стоком t, заданная матрица пропускных способностей C : **array** $[1..p, 1..p]$ **of real**.
Выход: матрица максимального потока F : **array** $[1..p, 1..p]$ **of real**.

```
for u, v ∈ V do
  F[u, v] := 0 { вначале поток нулевой }
end for
M : { итерация увеличения потока }
for v ∈ V do
  S[v] := 0; N[v] := 0; P[v] :=(+, nil, 0) { инициализация }
end for
S[s] := 1; P[s] :=(+, nil, ∞) { так как s ∈ S }
repeat
  a := 0 { признак расширения S }
  for v ∈ V do
    if S[v] = 1 & N[v] = 0 then
      for u ∈ Γ(v) do
        if S[u] = 0 & F[v, u] < C[v, u] then
          S[u] := 1; P[u] :=(+, v, min(P[v].δ, C[v, u] − F[v, u])); a := 1
        end if
```

```
        end for
        for u ∈ Γ^{-1}(v) do
          if S[u] = 0 & F[u,v] > 0 then
            S[u] := 1; P[u] := (−, v, min(P[v].δ, F[u,v])); a := 1
          end if
        end for
        N[v] := 1 { узел v отмечен }
      end if
    end for
    if S[t] then
      x := t { текущий узел аугментальной цепи }
      δ := P[t].δ { величина изменения потока }
      while x ≠ s do
        if P[x].s = + then
          F[P[x].n, x] := F[P[x].n, x] + δ { увеличение потока }
        else
          F[x, P[x].n] := F[x, P[x].n] − δ { увеличение потока }
        end if
        x := P[x].n { предыдущий узел аугментальной цепи }
      end while
      goto M
    end if
  until a = 0
```

Обоснование В качестве первого приближения берётся нулевой поток, который по определению является допустимым. В основном цикле, помеченном меткой M, делается попытка увеличить поток. Для этого в цикле **repeat** расширяется, пока это возможно, множество S узлов, достижимых из источника s по аугментальным цепям. При этом, если в множество S попадает сток t, то поток вдоль найденной аугментальной цепи $\langle s, t\rangle$ немедленно увеличивается на величину δ и начинается новая итерация с целью увеличить поток. Процесс расширения множества S в цикле **repeat** заканчивается, потому что множество узлов конечно, а отмеченные в массиве N узлы повторно не рассматриваются. Если процесс расширения множества S заканчивается и при этом сток t не попадает в множество S, то по теореме Форда–Фалкерсона найденный поток F является максимальным и работа алгоритма завершается. □

ЗАМЕЧАНИЕ

Приведённый алгоритм не является самым эффективным. Более подробное изложение известных методов можно найти, например, в [8] или в [21].

8.4.5. Связь между теоремой Менгера и теоремой Форда–Фалкерсона

Теорема Менгера является фундаментальным результатом, который проявляется в различных формах (см., например, задачи 8.3.1, 8.3.2, 8.3.3). Теорема Форда–Фалкерсона также может быть получена из теоремы Менгера. Далее приведена схема доказательства теоремы Форда–Фалкерсона на основе теоремы Менгера. Сначала нужно получить вариант теоремы Менгера в ориентированной рёберной форме: наибольшее число $\langle s,t\rangle$-путей, не пересекающихся по дугам, равно наименьшему числу дуг в (s,t)-разрезе. Это теорема Форда–Фалкерсона в том случае, когда $\forall e \in E\ (c(e)=1)$. Действительно, пусть Q — множество дуг из максимального набора непересекающихся $\langle s,t\rangle$-путей. Назначим поток следующим образом: $f(e):=1$, если $e \in Q$, и $f(e):=0$, если $e \notin Q$. Это поток, так как дивергенция в любом узле равна 0 и поток через дугу не превосходит пропускной способности. Величина этого потока равна $k \leqslant d^+(s)$, где k — число дуг, выходящих из s, которые начинают пути из Q. Этот поток максимальный. Действительно, если положить $f(e):=a>0$ для некоторой дуги $e \notin Q$, то:

1) если дуга e входит в узел, входящий в пути из Q, или выходит из такого узла, то нарушается условие потока (дивергенция $-a$ или $+a$ соответственно);
2) если вновь назначенные дуги с ненулевыми потоками образуют новый $\langle s,t\rangle$-путь, то это противоречит выбору Q.

Пусть теперь пропускные способности суть натуральные числа. Тогда можно провести аналогичные рассуждения для мультиграфов, считая, что вместо одной дуги с пропускной способностью n имеется n дуг с пропускной способностью 1. Если пропускные способности — рациональные числа, то их можно привести к общему знаменателю и свести задачу к предыдущему случаю.

Для вещественных пропускных способностей заключение теоремы можно получить путем перехода к пределу в условиях предыдущего случая.

ОТСТУПЛЕНИЕ

Приведенное в разделе 8.4.3 доказательство теоремы Форда–Фалкерсона *конструктивно*, из него не только следует заключение о величине максимального потока, но и извлекается способ нахождения максимального потока. Набросок доказательства в этом подразделе *неконструктивен* — из него нельзя извлечь алгоритм.

8.5. Связность в орграфах

Связность является одним из немногих понятий, которые не распространяются непосредственно с графов на другие родственные объекты и требуют отдельного определения и рассмотрения в каждом случае. В данном разделе рассматривается связность в орграфах, поскольку именно это понятие чаще всего применяется в программировании.

8.5.1. Сильная, односторонняя и слабая связность

В неориентированном графе две вершины либо связаны (если существует соединяющая их цепь), либо не связаны. В ориентированном графе отношение связанности узлов несимметрично, а потому определение связности отличается.

Пусть $G(V, E)$ — орграф, v_1 и v_2 — его узлы. Говорят, что два узла v_1 и v_2 *сильно связаны* в орграфе G, если существует путь (ориентированная цепь) как из v_1 в v_2, так и из v_2 в v_1. Говорят, что два узла v_1 и v_2 *односторонне связаны* в орграфе G, если существует путь либо из v_1 в v_2, либо из v_2 в v_1. Говорят, что два узла v_1 и v_2 *слабо связаны* в орграфе G, если они связаны в графе G', полученном из G забыванием ориентации дуг. Если все вершины в орграфе сильно (односторонне, слабо) связаны, то орграф называется *сильно (односторонне, слабо) связным*. Сильная связность влечет одностороннюю связность, которая влечет слабую связность. Обратное, разумеется, неверно.

Пример На рис. 8.7 показаны диаграммы сильно, односторонне и слабо связных орграфов.

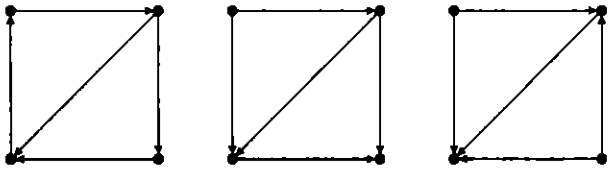

Рис. 8.7. Сильная (*слева*), односторонняя (*в центре*) и слабая (*справа*) связность

8.5.2. Компоненты сильной связности

Компонента сильной связности (употребляется аббревиатура КСС) орграфа G — это его максимальный сильно связный подграф.

Каждый узел орграфа принадлежит только одной КСС. Если узел не связан сильно с другими, то считаем, что он сам образует КСС. *Конденсацией* G^* орграфа G (или *графом Гёрца*, или *фактор-графом*) называется орграф, который получается стягиванием в один узел каждой КСС орграфа G.

Пример На рис. 8.8 показаны диаграммы орграфа и его конденсации.

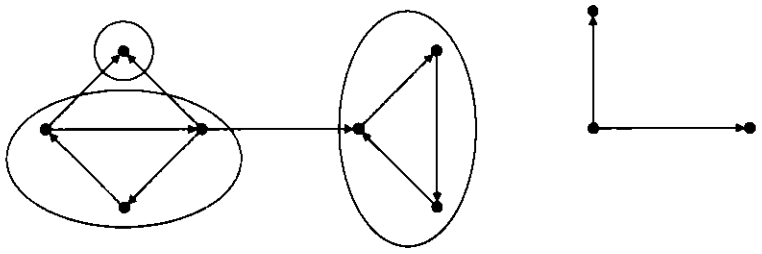

Рис. 8.8. Орграф (*слева*) и его фактор-граф (*справа*)

ЗАМЕЧАНИЕ

Фактор-граф не содержит контуров.

8.5.3. Выделение компонент сильной связности

Следующий алгоритм, основанный на методе поиска в глубину, находит все компоненты сильной связности орграфа.

Алгоритм 8.2 Выделение компонент сильной связности

Вход: орграф $G(V, E)$, заданный списками смежности $\Gamma(v)$.
Выход: список C компонент сильной связности, каждый элемент которого есть список узлов, входящих в компоненту сильной связности.

```
C := ∅
for v ∈ V do
  M[v] := {v} { M[v] список узлов, входящих в ту же КСС, что и v }
  e[v] := 0 { все узлы не рассмотрены }
end for
while V ≠ ∅ do
  select v ∈ V { взять v из V }
  v → T { положить v в стек }
  e[v] := 1 { отметить v }
  КСС { вызов процедуры КСС }
end while
```

Основная работа выполняется рекурсивной процедурой без параметров КСС, которая использует стек T для хранения просматриваемых узлов. Процедура КСС выделяет все компоненты сильной связности, достижимые из узла, выбранного в основном алгоритме.

```
if T = ∅ then
  return { негде выделять }
end if
v := top T { v — верхний элемент стека}
if Γ[v] ∩ V = ∅ then
  C := C ∪ M[v] { это КСС }
  V := V − v { удалить узел }
  v ← T { снять узел v со стека }
  КСС { возврат из тупика }
else
  for u ∈ Γ[v] do
    if e[u] = 0 then
      u → T { положить узел u в стек }
      e[u] := 1 { отметить узел u }
    else
      repeat
        w ← T { w — склеиваемый узел }
        V := V − w { удалить узел }
```

```
      Γ[u] := Γ[u] ∪ Γ[w] { запомнить смежность }
      M[u] := M[u] ∪ M[w] { склеивание узлов }
    until u = w
    w → T { чтобы не убрать тот узел, }
    V := V + w { с которого начали }
  end if
  КСС { поиск в глубину }
 end for
end if
```

Обоснование Достаточно заметить, что любой контур принадлежит ровно одной КСС, и более того, если в КСС входит несколько узлов, то они обязательно принадлежат некоторым контурам. Поэтому, если при обходе в глубину мы попадаем в уже отмеченный узел u, то это означает, что обнаружен контур, причём предшествующие узлы найденного контура находятся на стеке, начиная от вершины стека и до рассматриваемого отмеченного узла u, который также присутствует в стеке. Все узлы контура найденного контура можно «склеить». При склеивании узла w его список смежности и список уже найденных узлов той КСС, которой принадлежит узел w, объединяются с соответствующими списками узла u. После этого можно продолжить поиск в глубину от узла u, который для этой цели следует оставить в стеке. □

8.6. Кратчайшие пути

Задача нахождения кратчайшего пути в графе имеет столько практических применений и интерпретаций, что читатель, без сомнения, может сам легко привести множество примеров. Здесь рассматриваются четыре классических алгоритма, которые обязан знать каждый программист.

8.6.1. Длина дуг

Алгоритм Уоршалла (алгоритм 1.11) позволяет ответить на вопрос, достижима ли вершина v из вершины u, то есть существует ли цепь $\langle u, v\rangle$. Часто нужно не только определить, существует ли цепь, но и найти эту цепь.

Если задан орграф $G(V, E)$, в котором дуги нагружены числами (эти числа обычно называют *весами*, или *длинами*, дуг), то этот орграф можно представить в виде матрицы весов (длин) C:

$$C[i,j] = \begin{cases} 0, & \text{для } i = j, \\ c_{ij}, & \text{конечная величина, если есть дуга из узла } i \text{ в узел } j, \\ \infty, & \text{если нет дуги из узла } i \text{ в узел } j. \end{cases}$$

Длиной пути называется сумма длин дуг, входящих в этот путь. Наиболее часто на практике встречается задача отыскания *кратчайшего* пути.

ЗАМЕЧАНИЕ

Данное представление, равно как и последующие алгоритмы, применимы как к графам, так и к орграфам.

8.6.2. Алгоритм Флойда

Алгоритм Флойда[1] находит кратчайшие пути между всеми парами вершин (узлов) в (ор)графе. В этом алгоритме для хранения информации о путях используется матрица H: **array** $[1..p, 1..p]$ **of** $1..p$, где

$$H[i,j] = \begin{cases} k, \text{ если } k - \text{ первая вершина, достигаемая на кратчайшем пути из } i \text{ в } j; \\ 0, \text{ если из вершины } i \text{ в вершину } j \text{ нет пути.} \end{cases}$$

Алгоритм 8.3 Алгоритм Флойда

Вход: матрица $C[1..p, 1..p]$ длин дуг.
Выход: матрица $T[1..p, 1..p]$ длин путей и матрица $H[1..p, 1..p]$ самих путей.

```
for i from 1 to p do
  for j from 1 to p do
    T[i,j] := C[i,j] { инициализация }
    if C[i,j] = ∞ then
      H[i,j] := 0 { нет дуги из i в j }
    else
      H[i,j] := j { есть дуга из i в j }
    end if
  end for
end for
for i from 1 to p do
  for j from 1 to p do
    for k from 1 to p do
      if i ≠ j & T[j,i] ≠ ∞ & i ≠ k & T[i,k] ≠ ∞ & (T[j,k] = ∞ ∨ T[j,k] >
      > T[j,i] + T[i,k]) then
        H[j,k] := H[j,i] { запомнить новый путь }
        T[j,k] := T[j,i] + T[i,k] { и его длину }
      end if
    end for
  end for
  for j from 1 to p do
    if T[j,j] < 0 then
      stop { нет решения: вершина j входит в цикл отрицательной длины }
    end if
  end for
end for
```

ОТСТУПЛЕНИЕ

Матрица H размера $O(p^2)$ хранит информацию обо всех (кратчайших) путях в графе. Заметим, что всего в графе $O(p^2)$ путей, состоящих из $O(p)$ вершин. Таким образом, непосредственное представление всех путей потребовало бы памяти объема $O(p^3)$. Экономия памяти достигается за счет *интерпретации представления*, то есть динамического вычисления некоторой части информации вместо её хранения в памяти. В данном случае любой конкретный путь $\langle u, v \rangle$ легко извлекается из матрицы с помощью следующего алгоритма.

[1] Роберт Флойд (1936–2001).

```
w := u; yield w { первая вершина }
while w ≠ v do
  w := H[w, v]; yield w { следующая вершина }
end while
```

ЗАМЕЧАНИЕ

Если в G есть цикл с отрицательным весом, то решения поставленной задачи не существует, так как на этом цикле можно «накручивать» сколь угодно короткий путь.

Обоснование Алгоритм Флойда имеет много общего с алгоритмом Уоршалла (алгоритм 1.11). Покажем по индукции, что после выполнения i-го шага основного цикла по i элементы матриц $T[j,k]$ и $H[j,k]$ содержат, соответственно, длину кратчайшего пути и первую вершину на кратчайшем пути из вершины j в вершину k, проходящем через промежуточные вершины из диапазона $1..i$. База: $i = 0$, то есть до начала цикла элементы матриц T и H содержат информацию о кратчайших путях (если таковые есть), не проходящих ни через какие промежуточные вершины. Пусть теперь перед началом выполнения тела цикла на i-м шаге $T[j,k]$ содержит длину кратчайшего пути от j к k, а $H[j,k]$ содержит первую вершину на кратчайшем пути из вершины j в вершину k (если таковой есть). В таком случае, если в результате добавления вершины i к диапазону промежуточных вершин находится более короткий путь (в частности, если это вообще первый найденный путь), то он записывается. Таким образом, после окончания цикла, когда $i = p$, матрицы содержат кратчайшие пути, проходящие через промежуточные вершины $1..p$, то есть искомые кратчайшие пути. Алгоритм не всегда выдаёт решение, поскольку оно не всегда существует. Дополнительный цикл по j служит для прекращения работы в случае обнаружения в графе цикла с отрицательным весом. □

8.6.3. Алгоритм Дейкстры

Алгоритм Дейкстры[1] находит кратчайший путь между двумя данными вершинами (узлами) в (ор)графе, если длины дуг неотрицательны.

Алгоритм 8.4 Алгоритм Дейкстры

Вход: орграф $G(V,E)$, заданный матрицей длин дуг C : **array** $[1..p, 1..p]$ **of real**; s и t — вершины графа.

Выход: векторы T : **array** $[1..p]$ **of real**; и H : **array** $[1..p]$ **of** $0..p$. Если вершина v лежит на кратчайшем пути от s к t, то $T[v]$ — длина кратчайшего пути от s к v; $H[v]$ — вершина, непосредственно предшествующая v на кратчайшем пути.

```
for v from 1 to p do
  T[v] := ∞ { кратчайший путь неизвестен }
  X[v] := 0 { все вершины не отмечены }
end for
```

[1] Эдсгер Дейкстра (1930–2002).

$H[s] := 0$ { s ничего не предшествует }
$T[s] := 0$ { кратчайший путь имеет длину 0... }
$X[s] := 1$ { ... и он известен }
$v := s$ { текущая вершина }
M: { обновление пометок }
for $u \in \Gamma(v)$ **do**
  if $X[u] = 0$ & $T[u] > T[v] + C[v,u]$ **then**
    $T[u] := T[v] + C[v,u]$ { найден более короткий путь из s в u через v }
    $H[u] := v$ { запоминаем его }
  end if
end for
$m := \infty; v := 0$
{ поиск конца кратчайшего пути }
for u **from** 1 **to** p **do**
  if $X[u] = 0$ & $T[u] < m$ **then**
    $v := u; m := T[u]$ { вершина v заканчивает кратчайший путь из s }
  end if
end for
if $v = 0$ **then**
  stop { нет пути из s в t }
end if
if $v = t$ **then**
  stop { найден кратчайший путь из s в t }
end if
$X[v] := 1$ { найден кратчайший путь из s в v }
goto M

ЗАМЕЧАНИЕ

Для применимости алгоритма Дейкстры достаточно выполнения *неравенства треугольника*:

$$\forall u, v, w \in V \ \ (d(u,v) \leqslant d(u,w) + d(w,v)),$$

которое, очевидно, выполняется, если длины дуг неотрицательны. Если же допускаются отрицательные длины дуг, то алгоритм Дейкстры может оказаться неприменимым. Например, в графе с тремя узлами s, t и u, если $C[s,t] = 2$, $C[s,u] = 3$ и $C[u,t] = -2$, алгоритм 8.4 не найдет кратчайшего пути длиной 1 и выдаст путь длиной 2.

Обоснование Для доказательства корректности алгоритма Дейкстры достаточно заметить, что при каждом выполнении тела цикла, начинающегося меткой M, в качестве v используется вершина, для которой известен кратчайший путь из вершины s. Другими словами, если $X[v] = 1$, то $T[v] = d(s,v)$, и все вершины на пути $\langle s, v \rangle$, определяемом вектором H, обладают тем же свойством, то есть

$$\forall u \ \ (X[u] = 1 \Longrightarrow T[u] = d(s,u) \ \& \ X[H[u]] = 1).$$

Действительно (по индукции), первый раз в качестве v используется вершина s, для которой кратчайший путь пустой и имеет длину 0 (непустые пути не могут быть короче, потому что длины дуг неотрицательны). Пусть $T[u] = d(s,u)$ для всех ранее помеченных вершин u. Рассмотрим вновь помеченную вершину v, которая выбрана из условия $T[v] = \min\limits_{X[u]=0} T[u]$. Заметим, что если известен путь, проходящий через помеченные вершины, то тем самым известен кратчайший путь. Допустим (от противного), что $T[v] > d(s,v)$, то есть найденный путь, ведущий из s в v, не является кратчайшим. Тогда на этом пути должны быть непомеченные вершины. Рассмотрим первую вершину w на этом пути, такую, что $X[w] = 0$. Имеем: $T[w] = d(s,w) \leqslant d(s,v) < T[v]$, что противоречит выбору вершины v. □

ЗАМЕЧАНИЕ

Известно, что применение алгоритма Флойда в среднем примерно вдвое менее трудоёмко, чем применение алгоритма Дейкстры для всех пар вершин.

8.6.4. Дерево кратчайших путей

Алгоритм Дейкстры можно использовать и в том случае, когда нужно найти кратчайшие цепи (пути) от вершины s до всех остальных вершин.

ЗАМЕЧАНИЕ

В предыдущем подразделе мы отметили, что алгоритм Дейкстры применим как к графам, так и к орграфам в равной мере. Действительно, в этом алгоритме достаточно знать, какие вершины (узлы) являются соседними и какова цена (длина) перехода. Для графа матрица длин C и отношение смежности Γ симметричны, для орграфа это не так, но для работы алгоритма Дейкстры данное обстоятельство не имеет значения. Таким образом, алгоритм 8.5, равно как и предшествующий, применим как к графам, так и к орграфам. Именно поэтому в этом разделе мы позволяем себе вольности, используя в одном контексте термины, относящиеся к графам («вершина») и к орграфам («путь»).

Если достаточно знать только длины путей и все вершины достижимы из s, а длины дуг положительны, то алгоритм можно записать в следующей компактной и наглядной форме.

Алгоритм 8.5 Алгоритм Дейкстры для вычисления дерева кратчайших путей

Вход: орграф $G(V,E)$, заданный матрицей длин дуг C : **array** $[1..p, 1..p]$ **of real** и списками смежности Γ; s — исходная вершина графа.
Выход: вектор T : **array** $[1..p]$ **of real** длин кратчайших путей от s.

```
for u ∈ V do
    T[u] := C[s,u] { начальное приближение определяется матрицей }
    X[u] := 0 { все вершины не отмечены }
```

```
end for
for i from 1 to p do
  m := ∞ { поиск конца нового кратчайшего пути }
  for u ∈ V do
    if X[u] = 0 & T[u] < m then
      v := u, m := T[u] { вершина u заканчивает новый кратчайший путь из s }
    end if
  end for
  for u ∈ Γ(v) do
    T[u] := min(T[u], T[v] + C[v, u])
                        { пересчет оценки длины пути из s в u через v }
  end for
  X[v] := 1 { найден кратчайший путь из s в v}
end for
```

Обоснование Приведённый вариант алгоритма основан на той же идее, что и предшествующий: на очередном шаге нужно выбрать вершину v, до которой мы точно знаем кратчайший путь от s, и пересчитать оценки путей для вершин, смежных с v. Такой вершиной на первом шаге окажется вершина s (для нее $T[s] = 0$). На следующих шагах новой вершиной v является вершина, которая еще не рассмотрена ($X[u] = 0$) и для которой оценка пути минимальна. Действительно, даже если в будущем мы обнаружим другой путь, ведущий в v, его длина может быть разве что больше. □

Таким образом, грубая оценка трудоёмкости алгоритма Дейкстры составляет $O(p^2)$, поскольку вложенные циклы выполняются $O(p)$ раз. Заметим, что во внутреннем цикле явно совершается излишняя работа: вершины, далекие от s, не могут повлиять на выбор v на ранних шагах и их не стоит рассматривать раньше времени. Известно, что, более тщательно подбирая структуры данных для представления графа, можно уменьшить трудоёмкость алгоритма Дейкстры до $O(q \log_2 p)$.

ЗАМЕЧАНИЕ

Последнее замечание этого подраздела относится к его названию. Совокупность путей от вершины s к другим вершинам образует корневое дерево в смысле подраздела 9.2.1. Действительно, никакой кратчайший путь, очевидно, не содержит контуров (напомним, что длины дуг положительны). Всякий узел достижим из корня по построению. Более того, этот путь единственный, поскольку даже если в вершину ведет несколько кратчайших путей одинаковой длины, алгоритм выберет из них только один. Отсюда и из определений подраздела 9.2.1 немедленно следует, что традиционное название, использованное для данного подраздела, оправданно.

8.6.5. Кратчайшие пути в бесконтурном орграфе

Алгоритм Дейкстры применим, если длины дуг неотрицательны. Существует эффективный алгоритм, который позволяет найти дерево кратчайших путей в

орграфе, даже если длины дуг могут быть отрицательными, но известно, что орграф не содержит контуров.

ЛЕММА *В произвольном бесконтурном орграфе $G(V, E)$ узлы можно перенумеровать так, что* $\forall (v_i, v_j) \in E \ (i < j)$.

Доказательство По теореме 7.5.2 отношение достижимости в бесконтурном графе есть строгое частичное упорядочение на конечном множестве. Применяя алгоритм топологической сортировки 1.12, получаем требуемую нумерацию узлов. □

Допустим теперь, что узлы в бесконтурном орграфе перенумерованы так, что каждая дуга ведёт из узла с меньшим номером в узел с бóльшим номером, а источник этого орграфа, из которого нужно построить дерево кратчайших путей, имеет номер 1. Тогда алгоритм 8.6 находит кратчайшие пути от узла 1 до всех достижимых узлов.

Алгоритм 8.6 Определение расстояний от источника в бесконтурном графе

Вход: орграф $G(V, E)$, заданный матрицей длин дуг C : **array** $[1..p, 1..p]$ **of real** и списками предшествующих узлов Γ^{-1}; источник имеет номер 1.
Выход: вектор T : **array** $[1..p]$ **of real** длин кратчайших путей от источника.

```
for i from 1 to p do
  T[i] := C[1, i] { начальное приближение определяется матрицей }
end for
for i from 2 to p do
  for j ∈ Γ⁻¹(i) do
    T[i] := min(T[i], T[j] + C[j, i]) { пересчёт оценки длины пути }
  end for
end for
```

Обоснование Докажем индукцией по i, что основной цикл имеет инвариант $\forall j \in 1..i \ (T[j] = d(1, j))$. База: если $j = 1$, то $T[1] = 0 = d(1, 1)$. Пусть теперь $\forall j < i \ (T[j] = d(1, j))$. В кратчайшем пути $\langle 1, i \rangle$ все промежуточные узлы имеют номера больше 1 и меньше i по построению орграфа, в частности, вершина j, предшествующая i на кратчайшем пути, будет выбрана во внутреннем цикле, для неё по индукционному предположению $T[j] = d(1, j)$, и значит, $T[i] = d(1, i)$. □

ЗАМЕЧАНИЕ

В алгоритме используется множество «предшествования»

$$\Gamma^{-1}(v) \stackrel{\text{Def}}{=} \{u \mid v \in \Gamma(u)\},$$

которое совпадает со списками смежности $\Gamma(v)$ для графов и не пересекается с $\Gamma(v)$ для направленных орграфов.

Комментарии

Изложение центрального результата этой главы — теоремы Менгера (8.2.2) и сопутствующего материала — в основном следует [28]. Алгоритм нахождения максимального потока в сети заимствован из [21] с небольшими модификациями. Алгоритм выделения компонент сильной связности приведен в [4] и [20], где имеется весьма полный обзор различных алгоритмов обхода и анализа графов, применяемых в программировании. Алгоритмы Флойда и Дейкстры нахождения кратчайших путей принадлежит к числу классических общеизвестных алгоритмов, описание которых можно найти в большинстве учебников по дискретной математике, в частности, в [8], [27], [9].

Упражнения

8.1 а. Доказать, что если $\delta(G) > (p-1)/2$, то граф G связен.

8.1 б. Доказать вторую теорему подраздела 8.1.2.

8.2. Доказать, что наибольшее число непересекающихся множеств вершин, разделяющих вершины u и v, равно $d(u,v)-1$.

8.3. «Задача о гаремах». Имеется конечное множество юношей, каждый из которых знаком с некоторым конечным подмножеством множества девушек. Каждый юноша желает взять в жены *более чем одну* знакомую девушку. Найти необходимое и достаточное условие решения этой задачи.

8.4. Пусть в сети $G(V,E)$ помимо пропускной способности дуг заданы пропускные способности узлов, то есть задана нагрузка на вершины $D\colon V \to \mathbb{R}^+$. Для допустимого потока сумма потоков через входящие дуги не должна превышать пропускной способности вершины:

$$\forall v \in V \left(\sum_{\{u|(u,v)\in E\}} f(u,v) \leqslant D(v) \right).$$

Найти максимальный поток в такой сети.

8.5. Как может измениться количество компонент сильной связности орграфа при добавлении к нему одной дуги?

Сравните ответ с ответом на аналогичный вопрос для графов.

8.6. Пусть в графе $G(V,E)$ заданы вероятности успешного прохождения дуг, $\forall e \in E\ (0 \leqslant P[e] \leqslant 1)$. Вероятность успешного прохождения пути определяется как произведение вероятностей составляющих его дуг. Построить алгоритм нахождения наиболее надёжного пути (то есть имеющего наибольшую вероятность прохождения) от вершины s к вершине t.

Глава 9 Деревья

Деревья заслуживают отдельного и подробного рассмотрения по двум причинам:

- Деревья являются в некотором смысле простейшим классом графов. Для них выполняются многие интересные утверждения, которые не всегда выполняются для графов в общем случае. Применительно к деревьям многие доказательства и рассуждения оказываются намного проще. Выдвигая какие-то гипотезы при решении задач теории графов, целесообразно сначала их проверять на деревьях.
- Деревья являются самым распространённым классом графов, применяемых в программировании, причём в самых разных ситуациях. Более половины объёма этой главы посвящено рассмотрению конкретных применений деревьев в программировании.

9.1. Свободные деревья

Изучение деревьев целесообразно начать с самых общих определений и установления основных свойств.

9.1.1. Определения

Граф без циклов называется *ациклическим*, или *лесом*. Связный ациклический граф называется (*свободным*) *деревом*. Таким образом, компонентами связности леса являются деревья.

ЗАМЕЧАНИЕ

Прилагательное «свободное» употребляется в том случае, когда нужно подчеркнуть отличие деревьев от других объектов, родственных деревьям: ориентированных деревьев, упорядоченных деревьев и т. д.

В связном графе G выполняется неравенство $q(G) \geqslant p(G) - 1$ (см. 8.1.4). Граф G (не обязательно связный!), в котором $q(G) = p(G) - 1$, называется *древочисленным*.

В ациклическом графе G $z(G) = 0$. Пусть u, v — несмежные вершины графа G, $x = (u, v) \notin E$. Если граф $G + x$ имеет ровно один простой цикл, $z(G + x) = 1$, то граф G (не обязательно ациклический!) называется *субциклическим*.

ЗАМЕЧАНИЕ

Графы $K_3 \cup K_1$ и $K_3 \cup K_2$ являются древочисленными и субциклическими и в то же время не являются ни связными, ни ациклическими.

Пример На рис. 9.1 показаны диаграммы всех различных (свободных) деревьев с 5 вершинами, а на рис. 9.2 — диаграммы всех различных (свободных) деревьев с 6 вершинами.

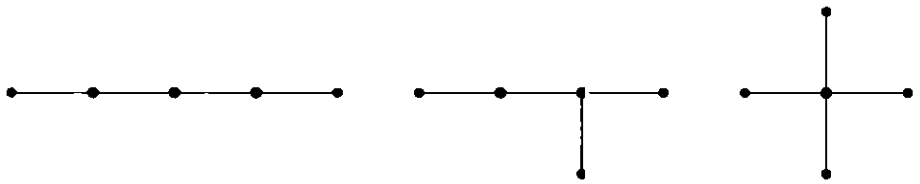

Рис. 9.1. Свободные деревья с 5 вершинами

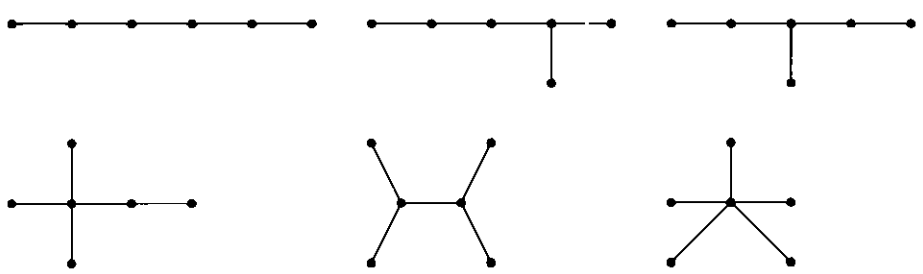

Рис. 9.2. Свободные деревья с 6 вершинами

9.1.2. Основные свойства деревьев

Следующая теорема устанавливает, что два из четырех свойств — связность, ацикличность, древочисленность и субцикличность — характеризуют граф как дерево.

ТЕОРЕМА *Пусть $G(V, E)$ — граф с p вершинами, q ребрами, k компонентами связности и z простыми циклами. Пусть далее x — ребро, соединяющее любую пару несмежных вершин в G. Тогда следующие утверждения эквивалентны:*

1. G — дерево, то есть связный граф без циклов,

$$k(G) = 1 \ \& \ z(G) = 0.$$

2. Любые две вершины соединены в G единственной простой цепью,
$$\forall u, v \ (|P(u,v)| = 1).$$

3. G — связный граф, и любое ребро есть мост,
$$k(G) = 1 \ \& \ \forall e \in E \ (k(G - e) > 1).$$

4. G — связный и древочисленный,
$$k(G) = 1 \ \& \ q(G) = p(G) - 1.$$

5. G — ациклический и древочисленный,
$$z(G) = 0 \ \& \ q(G) = p(G) - 1.$$

6. G — ациклический и субциклический,
$$z(G) = 0 \ \& \ z(G + x) = 1.$$

7. G — связный, субциклический и неполный,
$$k(G) = 1 \ \& \ G \neq K_p \ \& \ p \geqslant 3 \ \& \ z(G + x) = 1.$$

8. G — древочисленный и субциклический (за двумя исключениями),
$$q(G) = p(G) - 1 \ \& \ G \neq K_1 \cup K_3 \ \& \ G \neq K_2 \cup K_3 \ \& \ z(G + x) = 1.$$

Доказательство

[1⟹2] От противного. Пусть существуют две цепи $\langle u, v\rangle$. Некоторые вершины этих цепей различны. Обозначим через w_1 первую вершину при перечислении вершин от u к v, такую, что следующие вершины в цепях различны, а через w_2 обозначим первую вершину при перечислении вершин от v к u, такую, что следующие вершины в цепях различны (рис. 9.3, *слева*). Рассмотрим отрезок $\langle w_1, w_2\rangle$ первой цепи при перечислении вершин от u к v и отрезок $\langle w_2, w_1\rangle$ второй цепи при перечислении вершин от v к u. Тогда $\langle w_1, w_2\rangle + \langle w_2, w_1\rangle$ — цикл, что противоречит ацикличности графа G.

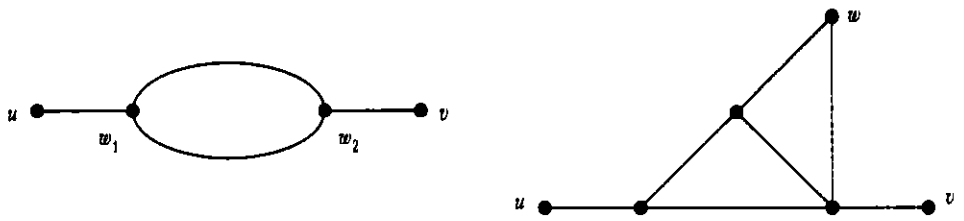

Рис. 9.3. К доказательству теоремы о свойствах деревьев

[2⟹3] Любые две вершины соединены цепью (единственной), следовательно, $k(G) = 1$. Далее от противного. Пусть ребро x — не мост. Тогда в $G - x$ концы этого ребра связаны цепью. Само ребро x — вторая цепь.

[3⟹4] Индукция по p. База: $p = 1 \Longrightarrow q = 0$. Пусть $q(G) = p(G) - 1$ для всех связных графов G с числом вершин меньше p, у которых любое ребро является мостом. Тогда удалим из графа G некоторое ребро x (которое является мостом). Получим две компоненты G' и G'', удовлетворяющие индукционному предположению. Имеем:

$$q' = p' - 1,\ q'' = p'' - 1,\ q = q' + q'' + 1 = p' - 1 + p'' - 1 + 1 = p - 1.$$

[4⟹5] От противного. Пусть есть цикл с n вершинами и n рёбрами. Остальные $p - n$ вершин имеют инцидентные им рёбра, которые связывают их с циклом. Следовательно, $q \geqslant p$, что противоречит условию $q = p - 1$.

[5⟹1] Граф без циклов, следовательно, его компоненты — деревья. Пусть их k. Имеем:

$$q = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} (p_i - 1) = \sum_{i=1}^{k} p_i - k = p - k.$$

Но $q = p - 1$, следовательно, $k = 1$.

[5⟹6] По ранее доказанному $5 \Longrightarrow 1 \Longrightarrow 2$, то есть любые две несмежные вершины соединены единственной простой цепью. Соединив две несмежные вершины ребром, получим единственный простой цикл.

[6⟹7] При $p \geqslant 3$ граф K_p содержит цикл, следовательно, $G \neq K_p$. Далее от противного. Пусть G несвязен, тогда при соединении ребром двух компонент связности цикл не возникнет.

[7⟹2] Имеем $k(G) = 1$, значит любые две несмежные вершины соединены простой цепью. Пусть цепь не единственная. Тогда существует цикл Z, причём $Z = K_3 = C_3$. Действительно, пусть $Z > C_3$, тогда, соединив две несмежные вершины этого цикла, получим два цикла. Но G связен и $G \neq K_3$, следовательно, существует другая вершина w, смежная с $Z = K_3$ (см. рис. 9.3, *справа*). Если w смежна более чем с одной вершиной Z, то имеем больше одного цикла. Если w смежна только с одной вершиной Z, то, соединив её с другой вершиной, получим два цикла.

[7⟹8] Имеем $k(G) = 1$, следовательно, $G \neq K_2 \cup K_3$, $G \neq K_1 \cup K_3$. Имеем по доказанному: $7 \Longrightarrow 2 \Longrightarrow 3 \Longrightarrow 4$, то есть $q = p - 1$.

[8⟹5] От противного. Пусть в G есть цикл $Z = C_n$. Если $n > 3$, то если внутри Z уже есть смежные вершины, имеем два цикла. Если в Z нет смежных вершин, то, соединив несмежные вершины в Z, получим два цикла. Следовательно, $Z = K_3$. Этот цикл Z является компонентой связности G. Действительно, пусть это не так. Тогда существует вершина w, смежная с Z. Если w смежна более чем с одной вершиной Z, то имеем больше одного цикла. Если w смежна только с одной вершиной Z, то, соединив её с другой вершиной, получим два цикла. Рассмотрим $G' := G - Z$. Имеем: $p = p' + 3$, $q = q' + 3$. Но $q = p - 1$, следовательно, $q' = p' - 1$. Отсюда $z(G') = 0$, так как один цикл уже есть. Следовательно, компоненты G' — деревья. Пусть их k. Имеем:

$$q' = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} (p_i - 1) = \sum_{i=1}^{k} p_i - k = p' - k,$$

по $q' = p' - 1$, следовательно, $k = 1$, то есть дерево одно. Если в этом дереве соединить несмежные вершины, то получим второй цикл. Два исключения: деревья, которые не имеют несмежных вершин, — это K_1 и K_2. Общая схема доказательства представлена на рис. 9.4. Граф доказательства сильно связен, следовательно, теорема доказана. □

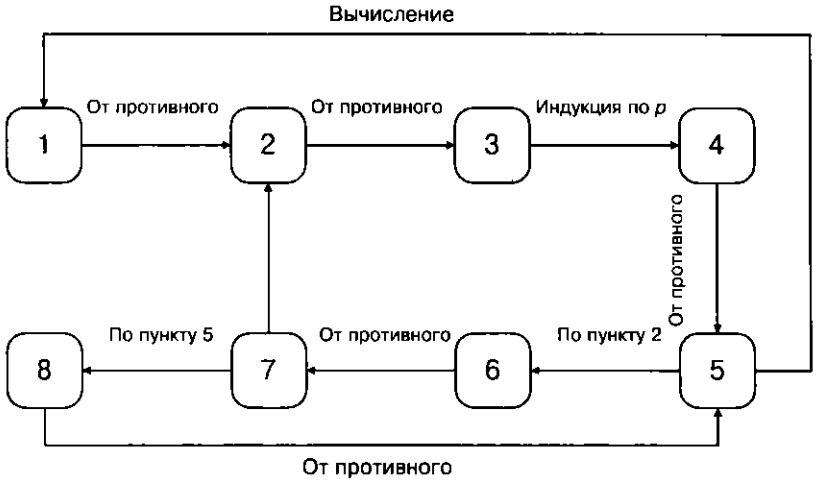

Рис. 9.4. Схема доказательства теоремы о свойствах деревьев

СЛЕДСТВИЕ 1 *В любом нетривиальном дереве имеются по крайней мере две висячие вершины.*

Доказательство Рассмотрим дерево $G(V, E)$. Дерево — связный граф, следовательно, $\forall v_i \in V \ \ (d(v_i) \geqslant 1)$. Далее от противного. Пусть $\forall i \in 1..p-1 \ \ (d(v_i) > 1)$. Тогда

$$2q = \sum_{i=1}^{p} d(v_i) > 2(p-1) + 1 = 2p - 1.$$

Но $q = p - 1$, то есть $2q = 2p - 2$. Имеем противоречие: $2p - 2 > 2p - 1$. □

ЗАМЕЧАНИЕ

Легко видеть, в частности, что висячими вершинами в дереве являются концы любого диаметра.

СЛЕДСТВИЕ 2 *Каждая не висячая вершина свободного дерева является точкой сочленения.*

Доказательство Пусть $G(V,E)$ — дерево, $v \in V$ и $d(v) > 1$. Тогда, по определению, $\exists u, w \in V \ (u \neq w \ \& \ (u,v) \in E \ \& \ (v,w) \in E)$. Граф G связен, поэтому существует цепь $\langle u, w\rangle$. Если $v \notin \langle u, w\rangle$, то имеем цикл $v, \langle u, w\rangle, v$, что противоречит тому, что G — дерево. Следовательно, $\exists u, w \in V \ (\forall \langle u, w\rangle \ (v \in \langle u, w\rangle))$, и по теореме 8.1.2 вершина v — точка сочленения. □

СЛЕДСТВИЕ 3 *Если в связном графе нет висячих вершин, то в нём есть цикл.*

Доказательство От противного. Если связный граф не имеет циклов, то он является деревом, и по следствию 1 обязан иметь висячие вершины. □

9.1.3. Центр дерева

Свободные деревья выделяются из других графов тем, что их центр всегда оправдывает своё название.

ТЕОРЕМА *Центр свободного дерева состоит из одной вершины или из двух смежных вершин:*

$$(z(G) = 0 \ \& \ k(G) = 1) \Longrightarrow (C(G) = K_1 \vee C(G) = K_2).$$

Доказательство Для деревьев K_1 и K_2 утверждение теоремы очевидно. Пусть теперь $G(V,E)$ — некоторое свободное дерево, отличное от K_1 и K_2. Рассмотрим граф $G'(V',E')$, полученный из G удалением всех висячих вершин. Заметим, что G' — дерево, поскольку ацикличность и связность при удалении висячих вершин сохраняется. Далее, если эксцентриситет $e_G(v) = d(v,u)$, то u — висячая вершина в дереве G (иначе можно было бы продолжить цепь «за» вершину u). Поэтому $\forall v \in V' \ (e_G(v) = e_{G'}(v) + 1)$, и при удалении висячих вершин эксцентриситеты оставшихся уменьшаются на 1. Следовательно, при удалении висячих вершин центр не меняется, $C(G) = C(G')$. Поскольку дерево G конечно, то, удаляя на каждом шаге все висячие вершины, в конце концов за несколько шагов придём к K_1 или K_2. □

9.2. Ориентированные, упорядоченные и бинарные деревья

Ориентированные (упорядоченные) деревья являются абстракцией иерархических отношений, которые очень часто встречаются как в практической жизни, так и в математике и программировании. Дерево (ориентированное) и иерархия — это равнообъёмные понятия.

9.2.1. Ориентированные деревья

Ориентированным деревом (или *ордеревом*, или *корневым* деревом) называется орграф со следующими свойствами:

1. Существует единственный узел r, полустепень захода которого равна 0, $d^+(r) = 0$. Он называется *корнем* ордерева.
2. Полустепень захода всех остальных узлов равна 1, $\forall v \in V - r \ (d^+(v) = 1)$.
3. Каждый узел достижим из корня, $\forall v \in V - r \ \left(\exists \overrightarrow{\langle r, v\rangle}\right)$.

Пример На рис. 9.5 приведены диаграммы всех различных ориентированных деревьев с 3 узлами, а на рис. 9.6 показаны диаграммы всех различных ориентированных деревьев с 4 узлами.

ТЕОРЕМА *Ордерево обладает следующими свойствами:*

1. $q = p - 1$.
2. *Если в ордереве забыть ориентацию дуг, то получится свободное дерево.*
3. *В ордереве нет контуров.*
4. *Для каждого узла существует единственный путь, ведущий в этот узел из корня.*
5. *Подграф, определяемый множеством узлов, достижимых из узла v, является ордеревом с корнем v (это ордерево называется поддеревом узла v).*
6. *Если в свободном дереве любую вершину назначить корнем, то получится ордерево.*

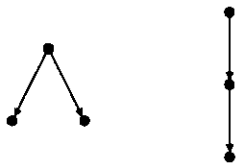

Рис. 9.5. Ориентированные деревья с 3 узлами

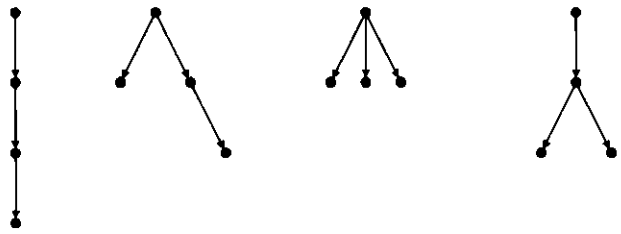

Рис. 9.6. Ориентированные деревья с 4 узлами

Доказательство

[1] Каждая дуга входит в какой-то узел. Из п. 2 определения 9.2.1 имеем: $\forall v \in V - r \ (d^+(v) = 1)$, где r — корень. Следовательно, $q = p - 1$.

[2] Пусть G — ордерево, граф G' получен из G забыванием ориентации рёбер, r — корень. Тогда $\forall v_1, v_2 \in V \ (\exists \langle v_1, r\rangle \in G' \ \& \ \exists \langle r, v_2\rangle \in G')$, следовательно, $\forall v_1, v_2 \ (\exists \langle v_1, v_2\rangle)$ и граф G' связен. Таким образом, учитывая п. 4. теоремы 9.1.2, G' — дерево.

[3] Следует из п. 2.

[4] От противного. Если бы в G существовали два пути из r в v, то в G' имелся бы цикл.

[5] Пусть G_v — правильный подграф, определяемый множеством узлов, достижимых из v. Тогда $d^+_{G_v}(v) = 0$, иначе узел v был бы достижим из какого-то узла $v' \in G_v$ и, таким образом, в G_v, а значит, и в G имелся бы контур, что противоречит п. 3. Далее имеем: $\forall v' \in G_v - v \ (d^+(v') = 1)$, так как $G_v \subset G$. Все узлы G_v достижимы из v по построению. По определению 9.2.1 получаем, что G_v — ордерево.

[6] Пусть вершина r назначена корнем и дуги последовательно ориентированы «от корня» обходом в глубину. Тогда $d^+(r) = 0$ по построению; $\forall v \in V - r \ (d^+(v) = 1)$, так как входящая дуга появляется при первом посещении узла; все узлы достижимы из корня, так как обход в глубину посещает все вершины связного графа. Таким образом, по определению 9.2.1 получаем ордерево. □

СЛЕДСТВИЕ *Алгоритм поиска в глубину строит ордерево с корнем в начальном узле.*

ЗАМЕЧАНИЕ

Каждую вершину в свободном дереве с p вершинами можно назначить корнем и получить ордерево. Некоторые из полученных ордеревьев могут оказаться изоморфными. Следовательно, свободное дерево определяет не более p различных ориентированных деревьев. Таким образом, общее число различных ордеревьев с p узлами не более чем в p раз превосходит общее число различных свободных деревьев с p вершинами.

Концевая вершина ордерева называется *листом*. Множество листьев называется *кроной*. Путь из корня в лист называется *ветвью*. Длина наибольшей ветви ордерева называется его *высотой*. *Уровень* узла ордерева — это расстояние от корня до узла. Сам корень имеет уровень 0. Узлы одного уровня образуют *ярус* ордерева.

ЗАМЕЧАНИЕ

Наряду с «растительной» применяется еще и «генеалогическая» терминология. Узлы, достижимые из узла u, называются *потомками* узла u (потомки одного узла образуют поддерево). Если узел v является потомком узла u, то узел u называется *предком* узла v. Если в дереве существует дуга (u, v), то узел u называется *отцом* (или *родителем*) узла v, а узел v называется *сыном* узла u. Сыновья одного отца называются *братьями*.

9.2.2. Эквивалентное определение ордерева

Ордерево T — это непустое конечное множество узлов, на котором определено разбиение, обладающее следующими свойствами:

1. Имеется один выделенный одноэлементный блок $\{r\}$, называемый корнем данного ордерева.
2. Остальные узлы (исключая корень) содержатся в k ($k \geqslant 0$) блоках $T_1, \ldots, T_k$, каждый из которых, в свою очередь, является ордеревом и называется поддеревом.

$$T \stackrel{\text{Def}}{=} \{\{r\}, T_1, \ldots, T_k\}.$$

Нетрудно видеть, что данное определение эквивалентно определению 9.2.1. Достаточно построить орграф, проводя дуги от заданного корня ордерева r к корням поддеревьев $T_1, \ldots, T_k$ и далее повторяя рекурсивно этот процесс для каждого из поддеревьев.

9.2.3. Упорядоченные деревья

Если относительный порядок поддеревьев $T_1, \ldots, T_k$ в эквивалентном определении ордерева фиксирован, то ордерево называется *упорядоченным*.

Примеры

Ориентированные и упорядоченные ориентированные деревья интенсивно используются в программировании:

1. Выражения. Для представления выражений языков программирования, как правило, используются ориентированные упорядоченные деревья. Пример представления выражения $a + b * c$ показан на рис. 9.7, *а*.
2. Для представления блочной структуры программы и связанной с ней структуры областей определения идентификаторов часто используется ориентированное дерево (может быть, неупорядоченное, так как порядок определения переменных в блоке в большинстве языков программирования считается несущественным). На рис. 9.7, *б* показана структура областей определения идентификаторов a, b, c, d, e, причём для отображения иерархии использованы вложенные области.
3. Для представления иерархической структуры вложенности элементов данных и/или операторов управления часто используется техника отступов, показанная на рис. 9.7, *в*.
4. Структура вложенности каталогов и файлов в современных операционных системах является упорядоченным ориентированным деревом. Обычно для изображения таких деревьев применяется способ, показанный на рис. 9.7, *г*.
5. Различные «правильные скобочные структуры» (например $(a(b)(c(d)(e)))$) являются ориентированными упорядоченными деревьями.

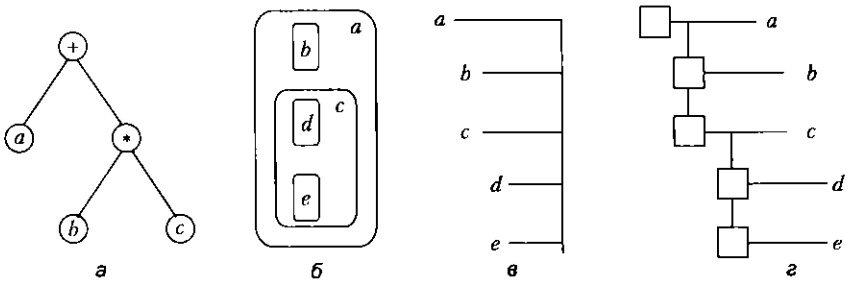

Рис. 9.7. Примеры изображения деревьев в программировании

ОТСТУПЛЕНИЕ

Тот факт, что большинство систем управления файлами использует ориентированные деревья, отражается даже в терминологии, например: «корневой каталог диска».

ЗАМЕЧАНИЕ

Общепринятой практикой при изображении деревьев является соглашение о том, что корень находится наверху и все дуги ориентированы сверху вниз, поэтому стрелки можно не изображать. Таким образом, диаграммы свободных, ориентированных и упорядоченных деревьев оказываются графически неразличимыми, и требуется дополнительное уточнение, дерево какого класса изображено на диаграмме. В большинстве случаев это ясно из контекста.

Пример На рис. 9.8 приведены три диаграммы деревьев, которые внешне выглядят различными. Как упорядоченные деревья они действительно все различны: (1) $\neq$ (2), (2) $\neq$ (3), (3) $\neq$ (1). Как ориентированные деревья (1) = (2), но (2) $\neq$ (3). Как свободные деревья они все изоморфны: (1) = (2) = (3).

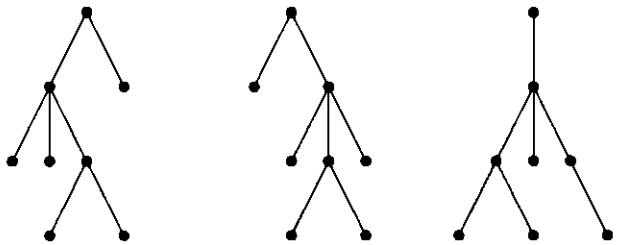

Рис. 9.8. Диаграммы деревьев

Указанное в подразделе 9.2.2 построение позволяет определить на упорядоченном ордереве единый линейный порядок узлов, согласованный с уже заданными в дереве упорядочениями.

ТЕОРЕМА *В упорядоченном ордереве с p узлами существует такая нумерация узлов числами из диапазона $1..p$, что номера потомков больше номеров предков и номера старших братьев больше номеров младших братьев.*

Доказательство Применим к упорядоченному ордереву алгоритм обхода в глубину (алгоритм 7.1), начиная с корня, причём узлы, смежные с данным, посещаются в порядке, определяемым порядком поддеревьев (см. 9.2.2). Присвоим узлам номера в порядке первого посещения. По теореме 7.4.6 все узлы получат уникальные номера из диапазона $1..p$ и корень получит номер 1. Далее, старшие братья получат номера, бóльшие, чем номера младших братьев по условию обхода, а потомки получат номера бóльшие, чем номера предков, поскольку алгоритм обхода в глубину посещает предков раньше, чем потомков. □

ЗАМЕЧАНИЕ

Указанный порядок обхода часто называют *прямым*.

Пример Узлы упорядоченного ордерева на рис. 9.11 слева при прямом обходе получат следующие номера: $a \mapsto 1$, $b \mapsto 2$, $c \mapsto 5$, $d \mapsto 3$, $e \mapsto 4$, $f \mapsto 6$, $g \mapsto 7$, $h \mapsto 8$, $i \mapsto 9$.

ЗАМЕЧАНИЕ

Построенная в доказательстве теоремы нумерация не является единственной нумерацией, обладающей указанными свойствами.

Пример Можно обойти упорядоченное ордерево по ярусам. При этом узлы упорядоченного ордерева на рис. 9.11, *слева*, получат следующие номера: $a \mapsto 1$, $b \mapsto 2$, $c \mapsto 3$, $d \mapsto 4$, $e \mapsto 5$, $f \mapsto 6$, $g \mapsto 7$, $h \mapsto 8$, $i \mapsto 9$.

9.2.4. Бинарные деревья

Бинарное (или *двоичное*) дерево — это непустое конечное множество узлов, на котором определена структура, обладающая следующими свойствами:

1. Имеется один выделенный узел r, называемый корнем данного бинарного дерева.
2. Остальные узлы (исключая корень) содержатся в двух непересекающихся множествах (поддеревьях) — левом и правом, каждое из которых, в свою очередь, либо пусто, либо является бинарным деревом.

На первый взгляд может показаться, что бинарное дерево — это частный случай упорядоченного ориентированного дерева, в котором у каждого узла не более двух смежных. Но это не так, бинарное дерево *не является* упорядоченным ордеревом. Дело в том, что даже если у некоторого узла бинарного дерева имеется только одно непустое поддерево, то всё равно известно, какое именно это поддерево: левое или правое.

Пример На рис. 9.9 приведены две диаграммы деревьев, которые изоморфны как упорядоченные, ориентированные и свободные деревья, но не изоморфны как бинарные деревья.

Рис. 9.9. Два различных бинарных дерева

ЗАМЕЧАНИЕ

Понятие двоичного дерева допускает обобщение. *m-ичным деревом* называется непустое конечное множество узлов, которое состоит из корня и m непересекающихся подмножеств, имеющих номера $1,\ldots,m$, каждое из которых в свою очередь, либо пусто, либо является m-ичным деревом. Большая часть утверждений и алгоритмов для двоичных деревьев может быть сравнительно легко распространена и на m-ичные деревья (с соответствующими модификациями). Поэтому хотя на практике m-ичные деревья и используются достаточно часто, здесь мы ограничиваемся только двоичными деревьями.

9.3. Представление деревьев в программах

Обсуждению представлений деревьев можно предпослать в точности те же рассуждения, что были предпосланы обсуждению представлений графов (см. 7.4). Кроме того, следует подчеркнуть, что задача представления деревьев в программе встречается гораздо чаще, чем задача представления графов общего вида, а потому методы её решения оказывают ещё большее влияние на практику программирования.

9.3.1. Представление свободных деревьев

Для представления деревьев можно использовать те же приёмы, что и для представления графов общего вида — матрицы смежности и инциденций, списки смежности и др. Но используя особенные свойства деревьев, можно предложить существенно более эффективные представления. Рассмотрим следующее представление свободного дерева, известное как *код Прюфера*. Допустим, что вершины дерева $T(V,E)$ занумерованы числами из интервала $1..p$. Построим последовательность A : **array** $[1..p-1]$ **of** $1..p$ в соответствии с алгоритмом 9.1.

Алгоритм 9.1 Построение кода Прюфера свободного дерева

Вход: Дерево $T(V,E)$ в любом представлении, вершины дерева занумерованы числами $1..p$ произвольным образом.
Выход: Массив A : **array** $[1..p-1]$ **of** $1..p$ — код Прюфера дерева T.

```
for i from 1 to p − 1 do
    v := min{k ∈ V | d(k) = 1} { выбираем вершину v — висячую вершину с наименьшим номером }
    A[i] := Γ(v) { заносим в код номер единственной вершины, смежной с v }
    V := V − v { удаляем вершину v из дерева }
end for
```

По построенному коду можно восстановить исходное дерево с помощью алгоритма 9.2.

Алгоритм 9.2 Распаковка кода Прюфера свободного дерева

Вход: Массив A : **array** $[1..p-1]$ **of** $1..p$ — код Прюфера дерева T.
Выход: Дерево $T(V, E)$, заданное множеством рёбер E, вершины дерева занумерованы числами $1..p$.

```
E := ∅ { в начале множество рёбер пусто }
B := 1..p { множество неиспользованных номеров вершин }
for i from 1 to p − 1 do
  v := min {k ∈ B | ∀j ⩾ i (k ≠ A[j])} { выбираем вершину v — неиспользованную вершину с наименьшим номером, который не встречается в остатке кода Прюфера }
  E := E + (v, A[i]) { добавляем ребро (v, A[i]) }
  B := B − v { удаляем вершину v из списка неиспользованных }
end for
```

Пример Для дерева, представленного на рис. 9.10, код Прюфера 7, 9, 1, 7, 2, 2, 7, 1, 2, 5, 12. На этом рисунке числа в вершинах — это их номера, а числа на рёбрах указывают порядок, в котором будут выбираться висячие вершины и удаляться рёбра при построении кода Прюфера.

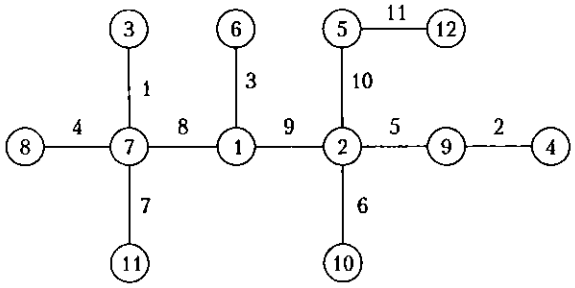

Рис. 9.10. Построение кода Прюфера

Обоснование Код Прюфера действительно является представлением свободного дерева. Чтобы убедиться в этом, покажем, что если T' — дерево, построенное алгоритмом 9.2 по коду A, который построен алгоритмом 9.1 по дереву T, то T' изоморфно T, $T' \sim T$. Для этого установим отображение $f\colon 1..p \to 1..p$ между номерами вершин в деревьях T и T' по порядку выбора вершин в алгоритмах: если вершина v выбрана на i-м шаге алгоритма 9.1, то вершина $f(v)$ выбрана на i-м шаге алгоритма 9.2. Заметим, что $\mathrm{Dom}\, f = 1..p$, поскольку висячие вершины есть в любом дереве и удаление висячей вершины оставляет дерево деревом. Далее, $\mathrm{Im}\, f = 1..p$, поскольку на i-м шаге алгоритма 9.2 использовано $i-1$ число из p чисел и остаётся $p-i+1$ чисел, а в хвосте кода Прюфера занято не более $p-i$ чисел. Более того, $\forall v\ (f(v) = v)$. Действительно, номера вершин, которые являются висячими в исходном дереве, не появляются в коде Прюфера (кроме, может быть, одной висячей вершины с наибольшим номером), а номера вершин,

которые не являются висячими, обязательно появляются. Поскольку при выборе первой вершины v в алгоритме 9.1 все вершины с меньшими номерами не являются висячими, их номера будет присутствовать в коде и, значит, не могут быть использованы на первом шаге алгоритма 9.2. Таким образом, на первом шаге алгоритм 9.2 выберет ту же вершину v. Но после удаления вершины v на втором шаге снова имеется дерево, к которому применимы те же рассуждения. Итак, f — тождественное и, значит, взаимно-однозначное отображение. Заметим теперь, что для определения i-го элемента кода на i-м шаге алгоритма 9.1 используется, а затем удаляется ребро $(v, A[i])$ и в точности то же ребро добавляется в дерево T' на i-м шаге алгоритма 9.2. Следовательно, f — взаимно-однозначное отображение, сохраняющее смежность и $T \sim T'$. □

ЗАМЕЧАНИЕ

Код Прюфера – наиболее экономное по памяти представление дерева. Его можно немного улучшить, если заметить, что существует всего одно дерево с двумя вершинами — K_2, а потому информацию о «последнем ребре» можно не хранить, она восстанавливается однозначно.

9.3.2. Представление упорядоченных ориентированных деревьев

В упорядоченных ордеревьях выделен корень и поддеревья упорядочены. Использование этой информации позволяет построить ещё более компактное представление по сравнению с кодом Прюфера для свободных деревьев. Пусть упорядоченное ордерево задано списками смежности (см. 7.4.4), причём узлы перенумерованы в соответствии с порядком прямого обхода, определённым в доказательстве теоремы 9.2.3. Тогда следующий алгоритм построит представление упорядоченного ордерева с p узлами в виде битовой шкалы (кода) из $2q = 2p - 2$ разрядов.

Алгоритм 9.3 Построение кода упорядоченного ордерева

Вход: Нетривиальное упорядоченное ордерево T, заданное списками смежности Γ : **array** $[1..p]$ **of** $\uparrow N$, где $N =$ **record** $v : 1..p; n : \uparrow N$ **end record**, причём узлы перенумерованы в прямом порядке.

Выход: Массив C : **array** $[1..2q]$ **of** 0..1, являющийся кодом ордерева T.

```
i := 0 { количество заполненных разрядов кода }
TraverseTree(1) { корневой узел имеет номер 1 }
```

Основная работа выполняется рекурсивной процедурой TraverseTree, для которой переменная i и массивы Γ и C глобальны.

Вход: Число n : $1..p$ — номер текущего узла.
Выход: Заполнение двух разрядов кода C.
$i := i + 1$; $C[i] := 1$ { отмечаем вход в узел }
$d := \Gamma[n]$ { указатель на первый элемент списка сыновей }
while $d \neq$ **nil do**
TraverseTree($d.v$) { построение кода поддерева очередного сына }
$d := d.n$ { выбор следующего сына }
end while
$i := i + 1$; $C[i] := 0$ { отмечаем выход из узла }

ЗАМЕЧАНИЕ

Алгоритм 9.3 в сущности строит протокол выполнения алгоритма обхода в глубину упорядоченного ордерева. Запись 1 отмечает вход в узел по единственной входящей дуге, запись 0 отмечает возврат из узла по этой же дуге.

По построенному коду нетрудно восстановить исходное представление упорядоченного ордерева с помощью алгоритма 9.4, в котором используются вспомогательный стек S для хранения номеров узлов и две процедуры:

- NewNode(v : $1..p, n$: $\uparrow N$) : $\uparrow N$ — функция, создающая новый элемент списка смежности с полями v и n и возвращающая указатель на него;
- Append(L, e : $\uparrow N$) — процедура, присоединяющая элемент, указатель на который задан параметром e, к списку, указатель на первый элемент которого задан параметром L.

Алгоритм 9.4 Восстановление упорядоченного ордерева по коду

Вход: Массив C : **array** $[1..2q]$ **of** $0..1$ — код упорядоченного ордерева T.
Выход: Массив Γ : **array** $[1..p]$ **of** $\uparrow N$, где N = **record** v : $1..p$; n : $\uparrow N$ **end record** — списки смежности упорядоченного ордерева T.
$p := 1$ { счётчик узлов }
$\Gamma[1] :=$ **nil** { корень }
$n := 1$ { текущий узел }
for i **from** 1 **to** $2q$ **do**
if $C[i] = 1$ **then**
$p := p + 1$ { номер нового узла }
$\Gamma[p] :=$ **nil** { новый узел пока листовой }
$d :=$ NewNode(p, **nil**) { дуга от текущего узла к новому узлу }
Append($\Gamma[n], d$) { добавить узел в список смежности }
$n \rightarrow S$ { положить номер текущего узла на стек }
$n := p$ { перейти в новый узел }
else
$n \leftarrow S$ { снять со стека и перейти в новый узел }
end if
end for

Обоснование Массив C является представлением упорядоченного ордерева T, то есть если к T применить алгоритм 9.3, а затем к результату применить алгоритм 9.4, то получится то же самое упорядоченное ордерево T. Действительно, алгоритм 9.4 интерпретирует протокол работы алгоритма 9.3. Каждый раз, когда алгоритм 9.3 входит по дуге в некоторый узел первый раз, он записывает в протокол 1, и в точности в том же порядке алгоритм 9.4 создаёт узлы. Каждый раз, когда алгоритм 9.3 возвращается на предыдущий уровень и записывает в протокол 0, алгоритм 9.4 возвращается к родительскому узлу, снимая его номер со стека. □

Пример Применение алгоритма 9.3 к упорядоченному ордереву на рис. 9.11, *слева*, даст код 1101001101011000.

ЗАМЕЧАНИЕ

Наложенное здесь требование прямого порядка нумерации узлов не является ограничением и используется только для упрощения изложения. Можно показать, что при *любой* нумерации узлов ордерева алгоритм 9.3 построит код, а алгоритм 9.4 восстановит по этому коду изоморфное ордерево, отличающееся разве что нумерацией узлов.

9.3.3. Число упорядоченных ориентированных деревьев

Представление упорядоченных ордеревьев, построенное в предыдущем подразделе, обладает замечательным характеристическим свойством, позволяющим получить явную формулу для числа упорядоченных ордеревьев. Введём обозначения. Пусть b : **array** $[1..n]$ **of** $0..1$ — любая битовая шкала. Обозначим через $N_0(b,k)$ — количество нулей в отрезке шкалы $b[1..k], k \leqslant n$, и через $N_1(b,k)$ — количество единиц в отрезке шкалы $b[1..k], k \leqslant n$. Пусть теперь C : **array** $[1..2q]$ **of** $0..1$ — битовая шкала, построенная по упорядоченному ордереву T алгоритмом 9.3. Тогда по построению алгоритма имеем:

$$N_0(C,2q) = N_1(C,2q)\ \&\ \forall\, k < 2q\ \ (N_0(C,k) \leqslant N_1(C,k))\,. \qquad (*)$$

Из алгоритма 9.4 следует, что данное свойство является характеристическим, то есть *любая* шкала, обладающая свойством $(*)$, является кодом некоторого дерева, причём по теореме 9.2.3 соответствие между деревьями и кодами взаимнооднозначно. Пусть S_n — множество битовых шкал длины $2n$:

$$S_n \stackrel{\text{Def}}{=} \{c = (a_1, \ldots, a_{2n}) \mid \forall\, i \in 1..2n\ \ (a_i \in \{0,1\})\}\,.$$

Обозначим число тех битовых шкал $c \in S_n$, которые обладают свойством $(*)$, через $C(n)$:

$$C(n) \stackrel{\text{Def}}{=} |\{c \in S_n \mid N_0(c,2n) = N_1(c,2n)\ \&\ \forall\, k < 2n\ \ (N_0(c,k) \leqslant N_1(c,k))\}|\,.$$

По определению $C(0) \stackrel{\text{Def}}{=} 1$.

Пример Ясно, что $C(1) = |\{(1,0)\}| = 1$, $C(2) = 2$ — см. рис. 9.5, $C(3) = 4$ — см. рис. 9.6.

ЛЕММА *Для числа $C(n)$ выполняется следующее равенство:*

$$C(n) = \sum_{k=0}^{n-1} C(k)C(n-k-1).$$

Доказательство Рассмотрим подмножество кодов $c \in S_n$, удовлетворяющих более сильному условию

$$N_0(C,2q) = N_1(C,2q)\ \&\ \forall k < 2q\ \ (N_0(C,k) < N_1(C,k))\,, \qquad (*')$$

и обозначим число таких кодов через $C'(n)$. Ясно, что в *любом* коде, удовлетворяющем условию $(*')$, первый бит равен 1, а последний — 0. Если их отбросить, то оставшаяся часть кода будет удовлетворять условию $(*)$, и поэтому $C'(n) = C(n-1)$, причём по определению $C'(1) = C(0) = 1$. Пусть теперь s — наименьшее число, такое, что код $c[1..n]$ удовлетворяет условию $(*')$. Сгруппируем все коды, удовлетворяющие условию $(*)$, по значениям числа s. В группе, в которой $s = n$, имеется $C'(n)$ кодов, а в группах, в которых $1 \leqslant s \leqslant n-1$, имеется $C'(s)C(n-s)$ кодов. Имеем

$$C(n) = C'(n) + \sum_{s=1}^{n-1} C'(s)C(n-s) = C(n-1)\cdot 1 + \sum_{s=1}^{n-1} C(s-1)C(n-s) =$$
$$= C(n-1)C(0) + \sum_{k=0}^{n-2} C(k)C(n-k-1) = \sum_{k=0}^{n-1} C(k)C(n-k-1),$$

где $k := s-1$. □

Таким образом, для числа $C(n)$ битовых шкал, удовлетворяющих условию $(*)$, выполняется характеристическое рекуррентное соотношение чисел Каталана (см. 5.7.4), откуда немедленно следует теорема.

ТЕОРЕМА *Число ориентированных упорядоченных деревьев с q дугами равно*

$$\frac{C(2q,q)}{q+1}.$$

9.3.4. Проверка правильности скобочной структуры

Из характеристического свойства $(*)$ рассматриваемого представления упорядоченных ордеревьев в качестве побочного, но полезного наблюдения можно извлечь эффективный алгоритм проверки правильности скобочной структуры (см. пример 5 в первой группе примеров подраздела 9.2.3).

Алгоритм 9.5 Проверка правильности скобочной структуры

Вход: Строка s : **array** $[1..n]$ **of char**, возможно, содержащая скобки «(» и «)».
Выход: Число 0, если скобочная структура правильна, или число в диапазоне $1..(n+1)$, указывающее на позицию в строке, где скобочная структура нарушена.

```
p := 0 { число прочитанных открывающих минус число закрывающих скобок }
for i from 1 to n do
  if s[i] = "(" then
    p := p + 1 { прочли открывающую скобку }
  end if
  if s[i] = ")" then
    p := p - 1 { прочли закрывающую скобку }
  end if
  if p < 0 then
    return i { лишняя закрывающая скобка }
  end if
end for
if p = 0 then
  return 0 { скобочная структура правильна }
else
  return n + 1 { не хватает закрывающих скобок }
end if
```

Обоснование Всякая правильная скобочная структура взаимно-однозначно соответствует упорядоченному ордереву. Если трактовать открывающую скобку как 1, а закрывающую — как 0, то последовательность скобок, содержащихся в строке, образует код дерева. Алгоритм очевидным образом проверяет выполнение условия $(*)$, то есть проверяет допустимость этого кода. □

Пример Строка $a(b(d)(e))(c(f)(g)(h(i)))$ является естественной записью с помощью правильной скобочной структуры дерева на рис. 9.11, *слева*.

9.3.5. Представление бинарных деревьев

Всякое свободное дерево можно ориентировать, назначив один из узлов корнем. Всякое ордерево можно произвольно упорядочить. Для потомков одного узла (братьев) упорядоченного ордерева определено отношение старше–младше (левее–правее). Всякое упорядоченное дерево можно представить бинарным деревом, например, проведя правую связь к старшему брату, а левую — к младшему сыну. Таким образом, достаточно рассмотреть представление в программе бинарных деревьев. Это наблюдение объясняет, почему представлению бинарных деревьев в программах традиционно уделяется особое внимание при обучении программированию.

Пример На рис. 9.11 приведены диаграммы упорядоченного и соответствующего ему бинарного деревьев.

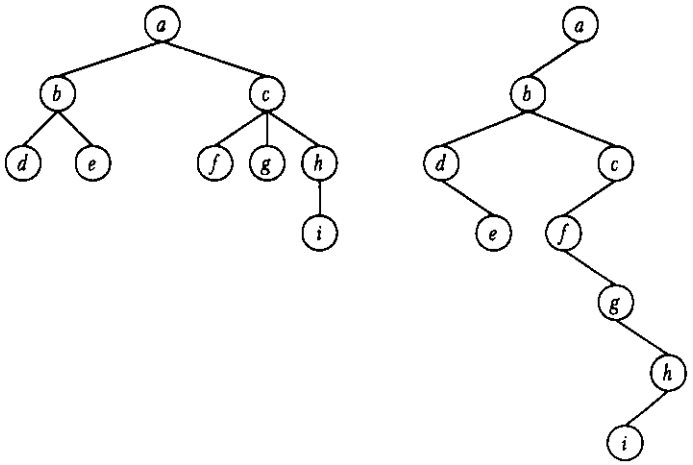

Рис. 9.11. Упорядоченное и бинарное деревья

ЗАМЕЧАНИЕ

Из данного представления следует, что множество бинарных деревьев взаимно-однозначно соответствует множеству упорядоченных лесов упорядоченных ордеревьев. Действительно, в указанном представлении одиночному упорядоченному ордереву всегда соответствует бинарное дерево, у которого правая связь корня пуста, а упорядоченному лесу — бинарное дерево, у которого правая связь корня не пуста.

Обозначим через $n(p)$ объем памяти, занимаемой представлением бинарного дерева, где p — число узлов. Наиболее часто используются следующие представления бинарных деревьев:

1. **Списочные структуры**: каждый узел представляется записью типа N, содержащей два поля (l и r) с указателями на левый и правый узлы и еще одно поле i для хранения указателя на информацию об узле. Дерево представляется указателем на корень. Тип N обычно определяется следующим образом: $N =$ **record** $i : info;\ l, r : \uparrow N$ **end record**, где тип $info$ считается заданным. Для этого представления $n(p) = 3p$.

ЗАМЕЧАНИЕ

Поскольку в бинарном дереве, как и в любом другом, $q = p - 1$, то из $2p$ указателей, отводимых для хранения дуг, $p + 1$ всегда хранит значение **nil**, то есть половина связей не используется.

2. **Упакованные массивы**: все узлы располагаются в массиве, так что все узлы поддерева данного узла располагаются вслед за этим узлом. Вместе с каждым узлом хранится индекс узла, который является первым узлом правого поддерева данного узла. Дерево T обычно определяется следующим образом: T : **array** $[1..p]$ **of record** $i : info$, $k : 1..p$ **end record**, где тип $info$ считается заданным. Для этого представления $n(p) = 2p$.

3. **Польская запись**: аналогично, но вместо связей фиксируется «размеченная степень» каждого узла (например, 0 означает, что это лист, 1 — есть левая связь, но нет правой, 2 — есть правая связь, но нет левой, 3 — есть обе связи). Дерево T определяется следующим образом: T : **array** $[1..p]$ **of record** $i : info$, $d : 0..3$ **end record**, где тип $info$ считается заданным. Для этого представления $n(p) = 2p$. Если степень узла известна из информации, хранящейся в самом узле, то можно не хранить и степень. Такой способ представления деревьев называется *польской записью* и обычно используется для представления выражений. В этом случае представление дерева оказывается наиболее компактным: объем памяти $n(p) = p$.

Пример Покажем, как выглядят в памяти бинарные деревья в разных представлениях. В приводимых ниже таблицах первая группа столбцов соответствует полям списочных структур, вторая — упакованным массивам и третья — польской записи с явными размеченными степенями. Условные адреса — это целые числа, а пустой указатель — 0.

1. Бинарное дерево на рис. 9.9, *слева*; считаем, что верхний узел содержит a, а нижний — b.

Адрес	i	l	r	i	k	i	d
1	a	2	0	a	0	a	1
2	b	0	0	b	0	b	0

2. Бинарное дерево на рис. 9.9, *справа*; считаем, что верхний узел содержит a, а нижний — b.

Адрес	i	l	r	i	k	i	d
1	a	0	2	a	2	a	2
2	b	0	0	b	0	b	0

3. Бинарное дерево на рис. 9.7, *слева*.

Адрес	i	l	r	i	k	i	d
1	$+$	2	3	$+$	3	$+$	3
2	a	0	0	a	0	a	0
3	$*$	4	5	$*$	5	$*$	3
4	b	0	0	b	0	b	0
5	c	0	0	c	0	c	0

4. Бинарное дерево на рис. 9.11, *справа*.

Адрес	i	l	r	i	k	i	d
1	a	2	0	a	0	a	1
2	b	3	5	b	5	b	3
3	d	0	4	d	4	d	2
4	e	0	0	e	0	e	0
5	c	6	0	c	0	c	1
6	f	0	7	f	7	f	2
7	g	0	8	g	8	g	2
8	h	9	0	h	0	h	1
9	i	0	0	i	0	i	0

9.3.6. Обходы бинарных деревьев

Большинство алгоритмов работы с деревьями основаны на обходах. Возможны следующие основные обходы бинарных деревьев.

Прямой (*префиксный*, *левый*) обход: попасть в корень,
обойти левое поддерево,
обойти правое поддерево.

Внутренний (*инфиксный*, *симметричный*) обход: обойти левое поддерево,
попасть в корень,
обойти правое поддерево.

Концевой (*постфиксный*, *правый*) обход: обойти левое поддерево,
обойти правое поддерево,
попасть в корень.

Кроме трёх основных, возможны еще три соответствующих обхода, отличающихся порядком рассмотрения левых и правых поддеревьев. Этим исчерпываются обходы, если в представлении фиксированы только дуги, ведущие от отцов к сыновьям.

ЗАМЕЧАНИЕ

Если кроме связей «отец–сын» в представлении есть другие связи, то возможны и другие (более эффективные) обходы. Деревья, в которых пустые поля l и r в структуре N используются для хранения дополнительных связей, называются *прошитыми деревьями*.

Пример Концевой обход дерева выражения $a + b * c$ дает *обратную* польскую запись этого выражения: $abc * +$.

ОТСТУПЛЕНИЕ

Польская запись выражений (прямая или обратная) применяется в некоторых языках программирования непосредственно и используется в качестве внутреннего представления программ во многих трансляторах и интерпретаторах. Причина заключается в том, что такая форма записи допускает очень эффективную интерпретацию (вычисление значения)

выражений. Например, значение выражения в обратной польской записи может быть вычислено при однократном просмотре выражения слева направо с использованием одного стека. В таких языках, как Forth и PostScript, обратная польская запись используется как основная.

9.3.7. Алгоритм симметричного обхода бинарного дерева

Реализация обходов бинарного дерева с помощью рекурсивных процедур не вызывает затруднений. В некоторых случаях из соображений эффективности применение явной рекурсии оказывается нежелательным. Следующий очевидный алгоритм реализует наиболее популярный симметричный обход без явной рекурсии, но с использованием стека.

Алгоритм 9.6 Алгоритм симметричного обхода бинарного дерева

Вход: бинарное дерево, представленное списочной структурой, r — указатель на корень.

Выход: последовательность узлов бинарного дерева в порядке симметричного обхода.

```
T := ∅; p := r { вначале стек пуст и p указывает на корень дерева }
M : { анализируем узел, на который указывает p }
if p = nil then
  if T = ∅ then
    stop { обход закончен }
  end if
  p ← T { левое поддерево обойдено }
  yield p { очередной узел при симметричном обходе }
  p := p.r { начинаем обход правого поддерева }
else
  p → T { запоминаем текущий узел... }
  p := p.l { ... и начинаем обход левого поддерева }
end if
goto M
```

9.4. Деревья сортировки

В этом разделе обсуждается одно конкретное применение деревьев в программировании, а именно *деревья сортировки* (также называемые *деревьями упорядочивания*). При этом рассматриваются как теоретические вопросы, связанные, например, с оценкой высоты деревьев, так и практическая реализация алгоритмов, а также целый ряд прагматических аспектов применения деревьев сортировки и некоторые смежные вопросы.

9.4.1. Ассоциативная память

В практическом программировании для организации хранения данных и доступа к ним часто используется механизм, который обычно называют *ассоциативной памятью*. При использовании ассоциативной памяти данные делятся на порции (называемые *записями*), и с каждой записью ассоциируется *ключ*. Ключ — это значение из некоторого линейно упорядоченного множества, а записи могут иметь произвольную природу и различные размеры. Доступ к данным осуществляется по значению ключа, которое обычно выбирается простым, компактным и удобным для работы.

Примеры

Ассоциативная память используется во многих областях жизни:

1. Толковый словарь или энциклопедия: записью является словарная статья, а ключом — заголовок словарной статьи (обычно его выделяют жирным шрифтом).
2. Адресная книга: ключом является имя абонента, а записью — адресная информация (телефон(ы), почтовый адрес и т. д.).
3. Банковские счета: ключом является номер счета, а записью — финансовая информация (которая может быть очень сложной).

Таким образом, ассоциативная память должна поддерживать по меньшей мере три основные операции:

1) добавить (ключ, запись);
2) найти (ключ, запись);
3) удалить (ключ).

Эффективность каждой операции зависит от структуры данных, используемой для представления ассоциативной памяти. Эффективность ассоциативной памяти в целом зависит от соотношения частоты выполнения различных операций в данной конкретной программе.

ЗАМЕЧАНИЕ

Таким образом, невозможно указать способ организации ассоциативной памяти, который оказался бы наилучшим во всех возможных случаях.

9.4.2. Способы реализации ассоциативной памяти

Для представления ассоциативной памяти используются следующие основные структуры данных:

1) неупорядоченный массив;
2) упорядоченный массив;

3) *дерево сортировки* — бинарное дерево, каждый узел которого содержит ключ (и указатель на запись) и обладает следующим свойством: значения ключа во всех узлах левого поддерева меньше, а во всех узлах правого поддерева — больше, чем значение ключа в узле;
4) *таблица расстановки* (или *хэш-таблица*).

При использовании неупорядоченного массива алгоритмы реализации операций ассоциативной памяти очевидны:

1. Операция «добавить (ключ, запись)» реализуется добавлением записи в конец массива.
2. Операция «найти (ключ, запись)» реализуется проверкой в цикле всех записей в массиве.
3. Операция «удалить (ключ)» реализуется поиском удаляемой записи, а затем перемещением последней записи на место удаляемой.

Для упорядоченного массива имеется эффективный алгоритм поиска, описанный в следующем подразделе. Реализация остальных операций очевидна. Основное внимание в этом разделе уделено алгоритмам выполнения операций с деревом сортировки.

ОТСТУПЛЕНИЕ

Таблицы расстановки являются чрезвычайно важным практическим приёмом программирования, подробное описание которого не включено в учебник из экономии места. Вкратце основная идея заключается в следующем. Подбирается специальная функция, которая называется *хэш-функцией*, переводящая значение ключа в адрес хранения записи (адресом может быть индекс в массиве, номер кластера на диске и т. д.). Таким образом, по значению ключа с помощью хэш-функции сразу определяется место хранения записи и открывается доступ к ней. Хэш-функция подбирается таким образом, чтобы разным ключам соответствовали, по возможности, разные адреса из диапазона возможных адресов записей. Как правило, мощность множества ключей существенно больше размера пространства адресов, которое, в свою очередь, больше количества одновременно хранимых записей. Поэтому при использовании хэширования возможны *коллизии* — ситуации, когда хэш-функция сопоставляет один и тот же адрес двум актуальным записям с различными ключами. Различные методы хэширования отличаются друг от друга способами разрешения коллизий и приёмами вычисления хэш-функций.

9.4.3. Алгоритм бинарного (двоичного) поиска

При использовании упорядоченного массива для представления ассоциативной памяти операция поиска записи по ключу может быть выполнена за время $O(\log_2 n)$ (где n — количество записей) с помощью следующего алгоритма, известного как алгоритм *бинарного* (или *двоичного*) поиска.

Алгоритм 9.7 Бинарный поиск

Вход: упорядоченный массив A : **array** $[1..n]$ **of record** $k : key; i : info$ **end record**; ключ $a : key$.
Выход: индекс записи с искомым ключом a в массиве A или 0, если записи с таким ключом нет.

```
b := 1 { начальный индекс части массива для поиска }
e := n { конечный индекс части массива для поиска }
while b ⩽ e do
  c := (b + e)/2 { индекс проверяемого элемента (округленный до целого) }
  if A[c].k > a then
    e := c − 1 { продолжаем поиск в первой половине }
  else if A[c].k < a then
    b := c + 1 { продолжаем поиск во второй половине }
  else
    return c { нашли искомый ключ }
  end if
end while
return 0 { искомого ключа нет в массиве }
```

Обоснование Достаточно заметить, что на каждом шаге основного цикла искомый элемент массива (если он есть) находится между (включительно) элементами с индексами b и e. Поскольку диапазон поиска на каждом шаге уменьшается вдвое, общая трудоёмкость не превосходит $\log_2 n$. □

9.4.4. Алгоритм поиска в дереве сортировки

Следующий алгоритм находит в дереве сортировки узел с указанным ключом, если он там есть.

Алгоритм 9.8 Поиск узла в дереве сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ $a : key$.
Выход: указатель p на найденный узел или **nil**, если в дереве нет такого ключа.

```
p := T { указатель на проверяемый узел }
while p ≠ nil do
  if a < p.i then
    p := p.l { продолжаем поиск слева }
  else if a > p.i then
    p := p.r { продолжаем поиск справа }
  else
    return p { нашли узел }
  end if
end while
return nil { искомого ключа нет в дереве }
```

Обоснование Этот алгоритм работает в точном соответствии с определением дерева сортировки: если текущий узел не искомый, то в зависимости от того, меньше или больше искомый ключ по сравнению с текущим, нужно продолжать поиск слева или справа соответственно. □

9.4.5. Алгоритм вставки в дерево сортировки

Следующий алгоритм вставляет в дерево сортировки узел с указанным ключом. Если узел с указанным ключом уже есть в дереве, то ничего не делается. Вспомогательная функция NewNode описана в подразделе 9.4.7.

Алгоритм 9.9 Вставка узла в дерево сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ $a : key$.
Выход: модифицированное дерево сортировки T.

```
if T = nil then
  T := NewNode(a) { первый узел в дереве }
  return T
end if
p := T { указатель на текущий узел }
while true do
  if a = p.i then
    return T { в дереве уже есть такой ключ }
  end if
  if a < p.i then
    if p.l = nil then
      p.l := NewNode(a) { создаём новый узел }
      return T { и подцепляем его к p слева }
    else
      p := p.l { продолжаем поиск места для вставки слева }
    end if
  else
    if a > p.i then
      if p.r = nil then
        p.r := NewNode(a) { создаём новый узел }
        return T { и подцепляем его к p справа }
      else
        p := p.r { продолжаем поиск места для вставки справа }
      end if
    end if
  end if
end while
```

Обоснование Алгоритм вставки, в сущности, аналогичен алгоритму поиска: в дереве ищется такой узел, имеющий свободную связь для подцепления нового узла, чтобы не нарушалось условие дерева сортировки. А именно, если новый ключ меньше текущего, то либо его можно подцепить слева (если левая связь свободна), либо нужно найти слева подходящее место. Аналогично, если новый ключ больше текущего. □

ОТСТУПЛЕНИЕ

В последнее время в среде профессиональных программистов наблюдается противоестественное и достойное сожаления явление — *обфускация* программ, то есть искусственное составление текстов программ таким образом, чтобы сделать их как можно более непонятными для читателя. Обычно обфускация обосновывается необходимостью защиты интеллектуальной собственности, хотя на самом деле, по мнению автора, часто является проявлением комплекса неполноценности, развивающегося на фоне недостаточного уровня профессиональной подготовки. Для иллюстрации последнего тезиса приведём пример того, как *не надо* писать программы, проведя обфускацию алгоритма 9.9.

Вход: дерево сортировки, заданное указателем на корень T : $\uparrow N$, где N = **record** i : key; l : **array** [0..1] **of** $\uparrow N$ **end record**; ключ a : key.
Выход: модифицированное дерево сортировки T.

```
if T = nil then
  T := NewNode(a)
  return T
end if
p := T
while true do
  if a = p.i then
    return T
  end if
  b := a > p.i
  if p.l[b] = nil then
    p.l[b] := NewNode(a)
    return T
  else
    p := p.l[b]
  end if
end while
```

Эта программа примерно вдвое короче, чуть медленнее (доступ к элементу массива обычно медленнее доступа к полю структуры), использует неявное приведение **false** $\mapsto$ 0, **true** $\mapsto$ 1, и её трудно понять без дополнительных пояснений.

9.4.6. Алгоритм удаления из дерева сортировки

Следующий алгоритм удаляет из дерева сортировки узел с указанным ключом. Если узла с указанным ключом нет в дереве, то ничего не делается. Вспомогательные процедуры Find и Delete описаны в следующем подразделе.

Алгоритм 9.10 Удаление узла из дерева сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ a : key.
Выход: модифицированное дерево сортировки T.

```
Find(T, a, p, q, s) { поиск удаляемого узла }
if p = nil then
  return T { нет такого узла — ничего делать не нужно }
end if
if p.r = nil then
  Delete(p, q, p.l, s) { случай 1, рис. 9.12, слева }
else
  u := p.r
  if u.l = nil then
    u.l := p.l
    Delete(p, q, u, s)  { случай 2, рис. 9.12, в центре }
  else
    w := u; v := u.l
    while v.l ≠ nil do
      w := v; v := v.l
    end while
    p.i := v.i
    Delete(v, w, v.r, −1) { случай 3, рис. 9.12, справа }
  end if
end if
return T
```

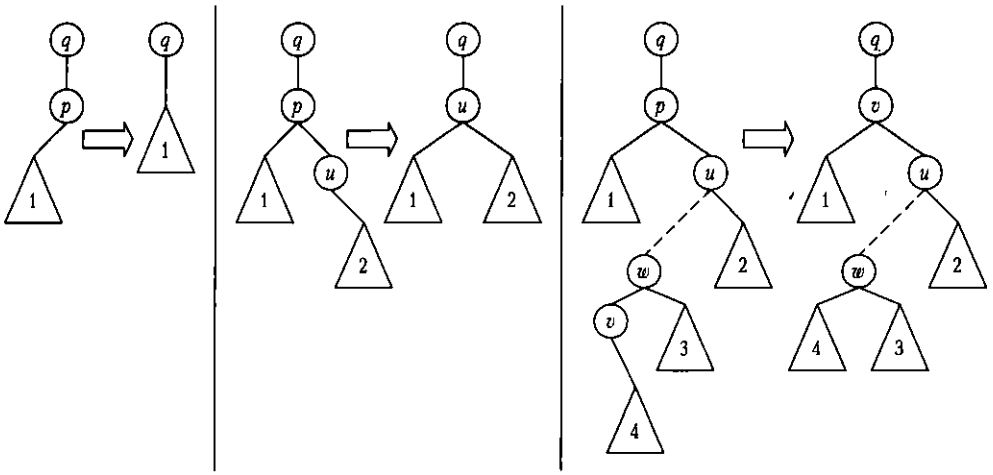

Рис. 9.12. Иллюстрация к алгоритму удаления узла из дерева сортировки

Обоснование Удаление узла производится перестройкой дерева сортировки. При этом возможны три случая (не считая тривиального случая, когда удаляемого узла нет в дереве и ничего делать не нужно).

[1] Правая связь удаляемого узла p пуста (см. рис. 9.12, *слева*). В этом случае левое поддерево 1 узла p подцепляется к родительскому узлу q с той же стороны, с которой был подцеплен узел p. Условие дерева сортировки, очевидно, выполняется.

[2] Правая связь удаляемого узла p не пуста и ведёт в узел u, левая связь которого пуста (см. рис. 9.12, *в центре*). В этом случае левое поддерево 1 узла p подцепляется к узлу u слева, а сам узел u подцепляется к родительскому узлу q с той же стороны, с которой был подцеплен узел p. Нетрудно проверить, что условие дерева сортировки выполняется и в этом случае.

[3] Правая связь удаляемого узла p не пуста и ведёт в узел u, левая связь которого не пуста. Поскольку дерево сортировки конечно, можно спуститься от узла u до узла v, левая связь которого пуста (см. рис. 9.12, *справа*). В этом случае выполняются два преобразования дерева. Сначала информация в узле p заменяется информацией узла v. Поскольку узел v находится в правом поддереве узла p и в левом поддереве узла u, имеем $p.i < v.i < u.i$. Таким образом, после этого преобразования условие дерева сортировки выполняется. Далее правое поддерево 4 узла v подцепляется слева к узлу w, а сам узел v удаляется. Поскольку поддерево 4 входило в левое поддерево узла w, условие дерева сортировки также сохраняется. □

ЗАМЕЧАНИЕ

В книге [14], из которой заимствован данный алгоритм, показано, что хотя алгоритм «выглядит несимметричным» (правые и левые связи обрабатываются по-разному), на самом деле в среднем характеристики дерева сортировки не искажаются.

9.4.7. Вспомогательные алгоритмы для дерева сортировки

Алгоритмы трех предыдущих разделов используют вспомогательные функции, описанные здесь.

1. Поиск узла — функция Find.

Вход: дерево сортировки T, заданное указателем на корень; ключ a : key.

Выход: p — указатель на найденный узел или **nil**, если в дереве нет такого ключа; q — указатель на отца узла p; s — способ подцепления узла q к узлу p ($s = -1$, если p слева от q; $s = +1$, если p справа от q; $s = 0$, если p — корень).

```
p := T; q := nil; s := 0 { инициализация }
while p ≠ nil do
  if p.i = a then
    return p, q, s
  end if
  q := p { сохранение значения p }
  if a < p.i then
    p := p.l; s := −1 { поиск слева }
  else
```

```
    p := p.r; s := +1 { поиск справа }
  end if
end while
return p, q, s
```

ОТСТУПЛЕНИЕ

В этой простой функции стóит обратить внимание на использование указателя q, который отслеживает значение указателя p «с запаздыванием», то есть указатель q «помнит» предыдущее значение указателя p. Такой приём полезен при обходе однонаправленных структур данных, в которых невозможно вернуться назад.

2. Удаление узла — процедура Delete.

Вход: $p1$ — указатель на удаляемый узел; $p2$ — указатель на подцепляющий узел; $p3$ — указатель на подцепляемый узел; s — способ подцепления.

Выход: преобразованное дерево.

```
if s = −1 then
  p2.l := p3 { подцепляем слева }
end if
if s = +1 then
  p2.r := p3 { подцепляем справа }
end if
dispose(p1) { удаляем узел }
```

3. Создание нового узла — конструктор NewNode.

Вход: ключ a.

Выход: указатель p на созданный узел.

```
new(p); p.i := a; p.l := nil; p.r := nil
return p
```

9.4.8. Сравнение представлений ассоциативной памяти

Пусть n — количество элементов в ассоциативной памяти. Тогда сложность операций для различных представлений ограничена сверху следующим образом.

	Неупорядоченный массив	Упорядоченный массив	Дерево сортировки
Добавить	$O(1)$	$O(n)$	$O(\log_2(n))..O(n)$
Найти	$O(n)$	$O(\log_2(n))$	$O(\log_2(n))..O(n)$
Удалить	$O(n)$	$O(n)$	$O(\log_2(n))..O(n)$

Эффективность операций с деревом сортировки ограничена сверху высотой дерева.

ЗАМЕЧАНИЕ

Дерево сортировки может расти неравномерно. Например, если при загрузке дерева исходные данные уже упорядочены, то полученное дерево будет право- или леволинейным и окажется даже менее эффективным, чем неупорядоченный массив.

В последующих подразделах обсуждаются методы уменьшения высоты дерева сортировки.

9.4.9. Выровненные и полные деревья

Бинарное дерево называется *выровненным*, если все листья находятся на одном (последнем) уровне. В выровненном дереве все ветви имеют одну длину, равную высоте дерева, однако выровненное дерево не всегда является эффективным деревом сортировки.

Пример Дерево, состоящее из корня и двух поддеревьев, леволинейного и праволинейного, ведущих к двум листьям, является выровненным, однако такое дерево имеет высоту $(p-1)/2$.

Бинарное дерево называется *заполненным*, если все узлы, степень которых меньше 2, располагаются на одном или двух последних уровнях. Другими словами, в заполненном дереве T все ярусы $D(r,i)$, кроме, может быть, последнего, заполнены:

$$\forall i \in 0..(h(T)-1)\ \left(|D(r,i)| = 2^i\right).$$

Пример На рис. 9.13 приведены диаграммы заполненного (*слева*) и незаполненного (*справа*) деревьев.

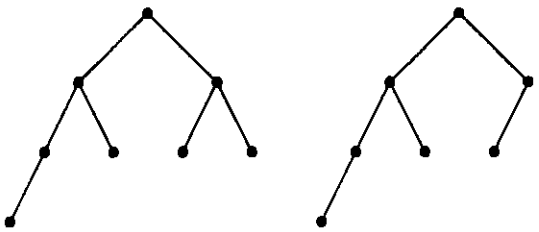

Рис. 9.13. Заполненное и незаполненное деревья

ЛЕММА $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$.

Доказательство По формуле для суммы геометрической прогрессии $\sum_{i=0}^{k} 2^i = 1 + 2 + \ldots + 2^k = \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1$. □

Заполненное дерево имеет наименьшую возможную для данного p высоту h.

ТЕОРЕМА *Для заполненного бинарного дерева* $\log_2(p+1) - 1 \leqslant h < \log_2(p+1)$.

Доказательство На i-м уровне может быть самое большее 2^i вершин, следовательно, $\sum_{i=0}^{h-1} 2^i < p \leqslant \sum_{i=0}^{h} 2^i$. По лемме $2^h < p+1 \leqslant 2^{h+1}$, и, логарифмируя, имеем: $h < \log_2(p+1)$ и $h+1 \geqslant \log_2(p+1)$. Следовательно, $\log_2(p+1) - 1 \leqslant h < \log_2(p+1)$. □

Выровненное заполненное бинарное дерево называется *полным*. В полном бинарном дереве все листья находятся на последнем уровне и все узлы, кроме листьев, имеют полустепень исхода 2. Другими словами, в полном дереве *все* ярусы заполнены.

СЛЕДСТВИЕ *Полное бинарное дерево высотой h имеет $2^{h+1} - 1$ узлов.*

Иногда полным называют бинарное дерево, в котором все нелистовые узлы имеют полустепень исхода 2, но листья могут встречаться на любом уровне. Высота дерева, полного в таком смысле, может изменяться в широких пределах.

ЗАМЕЧАНИЕ

Заполненные деревья дают наибольший возможный эффект при поиске. Однако известно, что вставка/удаление в заполненное дерево может потребовать полной перестройки всего дерева, и, таким образом, трудоёмкость операции в худшем случае составит $O(p)$.

Желательно выделить такой подкласс деревьев сортировки, в котором высота ограничена и в то же время операции по вставке и удалению элементов выполняются эффективно. Было найдено несколько таких подклассов, наиболее популярный из них описывается в следующем подразделе.

9.4.10. Сбалансированные деревья

(Бинарное) дерево называется *подровненным деревом*, или *АВЛ-деревом* (Адельсон-Вельский[1] и Ландис[2], 1962), или *сбалансированным по высоте деревом*, если для любого узла высоты левого и правого поддеревьев различаются не более чем на 1.

Пример На рис. 9.14 приведена диаграмма максимально несимметричного сбалансированного по высоте дерева, в котором для всех узлов высота левого поддерева ровно на 1 больше высоты правого поддерева.

ЗАМЕЧАНИЕ

Кроме сбалансированных по высоте, рассматривают и другие классы деревьев, например, *сбалансированные по весу* деревья, в которых для каждого узла количества узлов в левом и правом поддеревьях находятся в определённом соотношении. Здесь такие классы не рассматриваются, поэтому далее термин «сбалансированное дерево» означает сбалансированное по высоте бинарное дерево сортировки.

[1] Георгий Максимович Адельсон-Вельский (р. 1922).
[2] Евгений Михайлович Ландис (1921–1997).

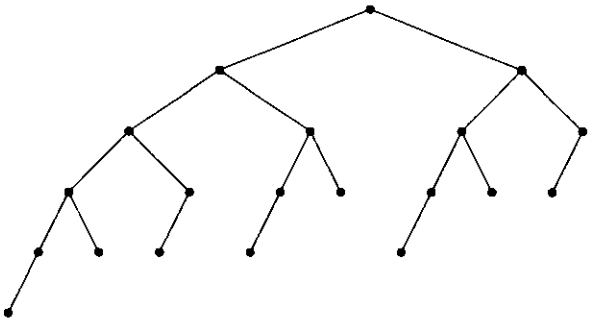

Рис. 9.14. Сбалансированное дерево

ТЕОРЕМА *Для сбалансированного по высоте бинарного дерева* $h < 2\log_2 p$.

Доказательство Рассмотрим наиболее несимметричные сбалансированные деревья, скажем, такие, у которых всякое левое поддерево на 1 выше правого. Такие сбалансированные деревья имеют максимальную возможную высоту среди всех сбалансированные деревьев с заданным числом вершин. Пусть P_h — число вершин в наиболее несимметричном сбалансированном дереве высоты h. По построению имеем: $p = P_h = P_{h-1} + P_{h-2} + 1$. Непосредственно проверяется, что $P_0 = 1$, $P_1 = 2$, $P_2 = 4$. Покажем по индукции, что $P_h \geqslant (\sqrt{2})^h$. База: $P_0 = 1$, $(\sqrt{2})^0 = 1$, $1 \geqslant 1$; $P_1 = 2$ $(\sqrt{2})^1 = \sqrt{2}$, $2 \geqslant \sqrt{2}$. Пусть $P_h \geqslant (\sqrt{2})^h$. Тогда

$$P_{h+1} = P_h + P_{h-1} + 1 \geqslant (\sqrt{2})^h + (\sqrt{2})^{h-1} + 1 =$$

$$= (\sqrt{2})^h \left(1 + \frac{1}{(\sqrt{2})} + \frac{1}{(\sqrt{2})^h}\right) > (\sqrt{2})^h \left(1 + \frac{1}{(\sqrt{2})}\right) > (\sqrt{2})^h \sqrt{2} = (\sqrt{2})^{h+1}.$$

Имеем: $(\sqrt{2})^h = 2^{h/2} \leqslant P_h = p$, следовательно, $\frac{h}{2} \leqslant \log_2 p$ и $h \leqslant 2\log_2 p$. □

Можно заметить, что рекуррентное соотношение для P_h похоже на определение чисел Фибоначчи $F(n)$ (см. 5.7.3), откуда легко выводимо

СЛЕДСТВИЕ *Если* $h > 0$, *то* $P_h = F(h) + P_{h-1}$.

Доказательство По индукции. База: $P_1 = 2$, $F(1) = 1$, $P_0 = 1$, $2 = 1 + 1$. Пусть $P_{h-1} = F(h-1) + P_{h-2}$. Рассмотрим P_h. Имеем $P_h = P_{h-1} + P_{h-2} + 1 = F(h-1) + P_{h-2} + F(h-2) + P_{h-3} + 1 = F(h-1) + F(h-2) + P_{h-2} + P_{h-3} + 1 = F(h) + P_{h-1}$. □

ЗАМЕЧАНИЕ

Известна более точная оценка высоты сбалансированного дерева: $h < \log_{(\sqrt{5}+1)/2} 2\log_2 p$.

Сбалансированные деревья уступают заполненным деревьям по скорости поиска (менее чем в два раза), однако их преимущество состоит в том, что известны алгоритмы вставки узлов в сбалансированное дерево и удаления их из него, которые сохраняют сбалансированность и в то же время при перестройке дерева

затрагивают только конечное число узлов (см. следующий подраздел). Поэтому во многих случаях сбалансированное дерево оказывается наилучшим вариантом представления дерева сортировки.

9.4.11. Балансировка деревьев

При добавлении (или при удалении) узла в сбалансированном дереве сортировки может возникнуть ситуация, в которой баланс нарушается. В этом случае необходимо перестроить дерево, чтобы восстановить баланс. Алгоритм балансировки дерева представлен на рис. 9.15 и 9.16, при этом использованы следующие обозначения: x — добавляемый узел, v — первый узел, в котором нарушен баланс на пути от корня r к новому узлу x. Тогда возможны два варианта. Путь от v к x состоит либо из дуг одной ориентации (скажем, только из левых дуг), либо из левых и правых дуг. Варианты, в которых путь состоит только из правых дуг либо из правых и левых дуг, зеркально симметричны. В первом варианте дерево перестраивается в соответствии с *правилом простого вращения*, представленным на рис. 9.15.

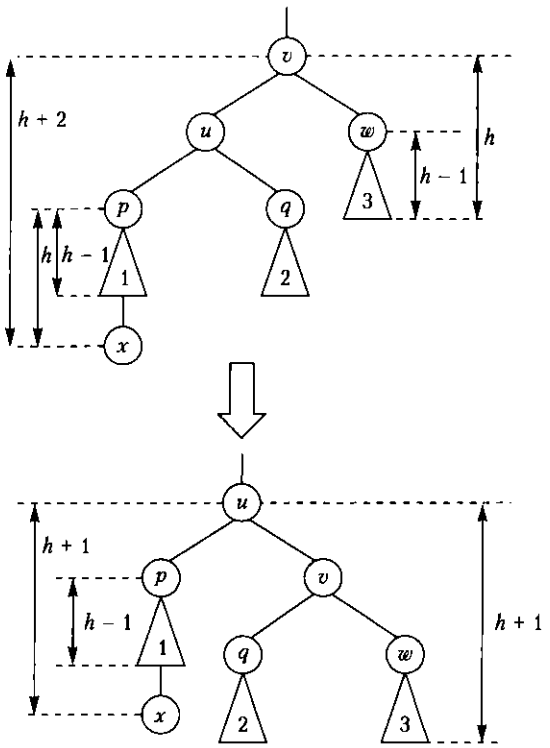

Рис. 9.15. Простое вращение

Простое вращение восстанавливает баланс и сохраняет условие дерева сортировки. Действительно, до преобразования имеем

$$u < v, p < v, q < v, p < u, q > u, w > v.$$

Следовательно,

$$p < u, u < v, u < q, u < w, q < v, w > v$$

и преобразование не нарушает условия дерева сортировки. Во втором варианте дерево перестраивается в соответствии с *правилом двойного вращения*, представленным на рис. 9.16. Двойное вращение восстанавливает баланс и сохраняет

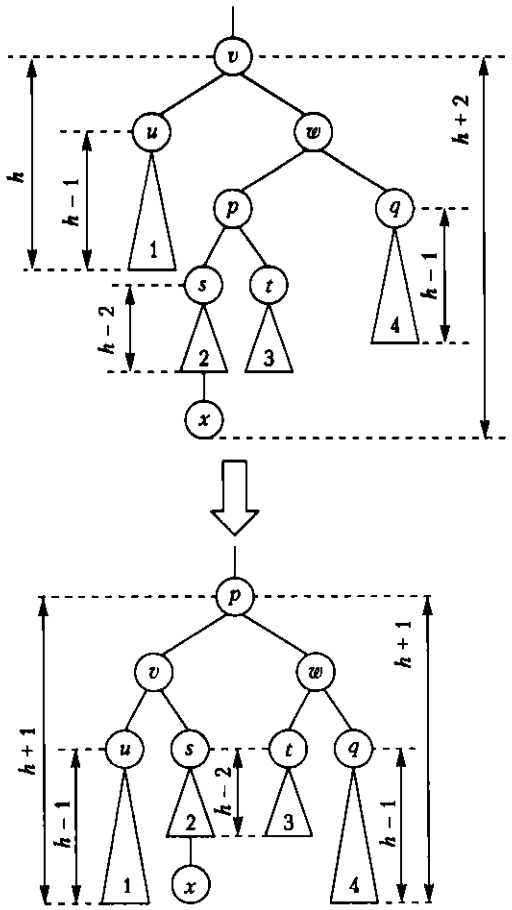

Рис. 9.16. Двойное вращение

условие дерева сортировки. Действительно, до преобразования имеем

$$u < v, w > v, p > v, q > v, p < w, q > w, s > v, t > v, s < w, t < w, s < p, t > p.$$

Следовательно,

$$v < p, u < p, s < p, u < v, s > v, w > p, t > p, q > p, t < w, q > w$$

и преобразование не нарушает условия дерева сортировки.

ЗАМЕЧАНИЕ

Детальное обсуждение алгоритмов работы с АВЛ-деревьями, включая оценки трудоёмкости в среднем и в худшем случае, можно найти в [26] и [16].

9.5. Кратчайший остов

Задача отыскания кратчайшего остова графа является классической задачей теории графов. Методы решения этой задачи послужили основой для многих других важных результатов. В частности, исследования алгоритма Краскала[1], описанного в подразделе 9.5.3, привели в конечном счёте к теории жадных алгоритмов, изложенной в подразделе 2.6.4.

9.5.1. Определения

Пусть $G(V, E)$ — граф. *Остовный подграф* графа $G(V, E)$ — это подграф, содержащий все вершины. Остовный подграф, являющийся деревом, называется *остовом* или *каркасом*.

ЗАМЕЧАНИЕ

Остов определяется множеством рёбер, поскольку вершины остова суть вершины графа.

Несвязный граф не имеет остова. Связный граф может иметь много остовов.

ОТСТУПЛЕНИЕ

Если задать длины рёбер, то можно поставить задачу нахождения *кратчайшего* остова. Эта задача имеет множество практических интерпретаций. Например, пусть задано множество аэродромов и нужно определить минимальный (по сумме расстояний) набор авиарейсов, который позволил бы перелететь с любого аэродрома на любой другой. Решением этой задачи будет кратчайший остов полного графа расстояний между аэродромами.

ЗАМЕЧАНИЕ

Существует множество различных способов найти какой-то остов графа. Например, алгоритм поиска в глубину строит остов (по рёбрам возврата). Множество кратчайших путей из заданной вершины ко всем остальным также образует остов. Однако этот остов может не быть кратчайшим.

Пример На рис. 9.17 показаны диаграммы (*слева направо*) графа, дерева кратчайших путей из вершины 1 с суммарным весом 5 и два кратчайших остова этого графа.

[1] Джозеф Бернард Краскал (р. 1928).

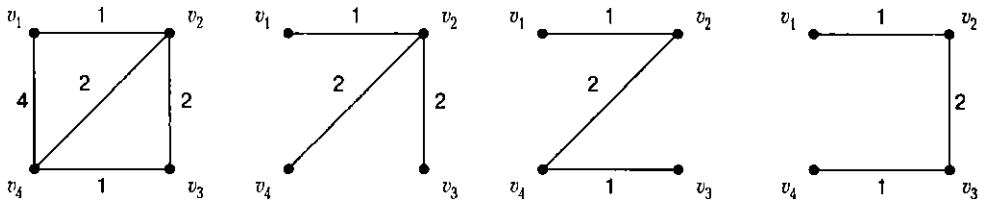

Рис. 9.17. Граф, дерево кратчайших путей и два кратчайших остова

9.5.2. Схема алгоритма построения кратчайшего остова

Рассмотрим следующую схему алгоритма построения кратчайшего остова. Пусть T — множество непересекающихся деревьев, являющихся подграфами графа G. Вначале T состоит из отдельных вершин графа G, в конце T содержит единственный элемент — кратчайший остов графа G.

Алгоритм 9.11 Построение кратчайшего остова

Вход: граф $G(V, E)$, заданный матрицей длин рёбер C.
Выход: кратчайший остов T.

```
T := V
while в T больше одного элемента do
  взять любое поддерево из T
  найти к нему ближайшее
  соединить эти деревья в T
end while
```

Обоснование Возьмём произвольное дерево T_i и выберем ближайшее к нему дерево T_j в T, $T_i \cap T_j = \varnothing$. Положим

$$\Delta_{i,j} := \min_{t_i \in T_i, t_j \in T_j} d(t_i, t_j).$$

Так как с самого начала все вершины G покрыты деревьями из T, а T_j выбирается ближайшим к T_i, то $\Delta_{i,j}$ всегда реализуется на некотором ребре с длиной $c_{i,j}$. Далее индукцией по шагам алгоритма 9.11 покажем, что все рёбра, включенные в деревья из T, принадлежат кратчайшему остову — SST[1]. Вначале выбранных рёбер нет, поэтому их множество включается в кратчайший остов. Пусть теперь все рёбра, добавленные в T, принадлежат SST. Рассмотрим очередной шаг алгоритма. Пусть на этом шаге добавлено ребро (i, j), соединяющее поддерево T_i с поддеревом T_j. Если $(i, j) \notin SST$, то, поскольку SST является деревом и, стало быть, связен, $\exists (i^*, j^*) \in SST$, соединяющее T_i с остальной частью SST. Тогда удалим из SST ребро (i^*, j^*) и добавим ребро (i, j): $SST' := SST - (i^*, j^*) + (i, j)$. Полученный подграф SST' является остовом, причём более коротким, чем SST, что противоречит выбору SST. □

[1] *SST* — *Shortest Spanning Tree* — стандартное обозначение для кратчайшего остова.

ЗАМЕЧАНИЕ

Различные способы выбора поддерева для наращивания на первом шаге тела цикла дают различные конкретные варианты алгоритма построения SST.

9.5.3. Алгоритм Краскала

Следующий алгоритм, известный как *алгоритм Краскала*, находит кратчайший остов в связном графе.

Алгоритм 9.12 Алгоритм Краскала

Вход: список E рёбер графа G с длинами, упорядоченный в порядке возрастания длин.
Выход: множество T рёбер кратчайшего остова.

```
T := ∅
k := 1 { номер рассматриваемого ребра }
for i from 1 to p − 1 do
  while z(T + E[k]) > 0 do
    k := k + 1 { пропустить это ребро }
  end while
  T := T + E[k] { добавить это ребро в SST }
  k := k + 1 { и исключить его из рассмотрения }
end for
```

Обоснование Для обоснования алгоритма Краскала достаточно показать, что выдерживается схема алгоритма предыдущего подраздела. Действительно, поскольку в множестве рёбер T нет циклов по построению, это множество суть совокупность рёбер некоторого множества деревьев. Если множество T содержит более одного дерева, то существует ребро, при добавлении которого не возникает цикла — оно соединяет два дерева в T. Добавляемое ребро — кратчайшее возможное, значит, на нём реализуется расстояние между некоторыми деревьями в T. По построению в конце работы алгоритма множество рёбер T содержит $p-1$ элемент, а значит, является искомым остовом. □

ЗАМЕЧАНИЕ

Исследование алгоритма Краскала дало толчок развитию теории жадных алгоритмов, см. 2.6.

ТЕОРЕМА *Семейство всех таких подмножеств множества рёбер графа, которые не содержат циклов, является матроидом.*

Доказательство Пустое множество рёбер не содержит циклов и аксиома M_1 (см. 2.6.1) выполнена. Далее, если множество рёбер не содержит циклов, то любое его подмножество также не содержит циклов, и аксиома M_2 выполнена. Пусть теперь $E' \subset E$ — произвольное множество рёбер, а G' — правильный подграф графа G, образованный этими рёбрами. Очевидно, что любое максимальное не содержащее циклов подмножество множества E' является объединением остовов компонент связности графа G' и по теореме 9.1.2 содержит $p(G') - k(G')$ элементов. Таким образом, все максимальные не содержащие циклов подмножества произвольного множества рёбер содержат одинаковое количество элементов, и по теореме 2.6.2 семейство ациклических подмножеств рёбер образует матроид. □

Таким образом, алгоритм Краскала как жадный алгоритм (см. 2.6.4), применённый к матроиду, находит ациклическое подмножество рёбер наименьшего веса. По построению алгоритма (основной цикл до $p - 1$) это подмножество рёбер древочисленно, а значит, является кратчайшим остовом.

9.5.4. Алгоритм Прима

В алгоритме Прима[1] кратчайший остов порождается в процессе разрастания одного дерева, к которому присоединяются одиночные вершины. При этом для каждой вершины v, кроме начальной, используются две пометки: $\alpha[v]$ — это ближайшая к v вершина, уже включённая в остов, а $\beta[v]$ — это длина ребра, соединяющего v с остовом. Если вершину v ещё нельзя соединить с остовом одним ребром, то $\alpha[v] := 0, \beta[v] := \infty$.

Алгоритм 9.13 Алгоритм Прима

Вход: граф $G(V, E)$, заданный матрицей длин рёбер C.
Выход: множество T рёбер кратчайшего остова.

```
  select u ∈ V  { выбираем произвольную вершину }
S := {u} { S — множество вершин, включённых в кратчайший остов }
T := ∅ { T — множество рёбер, включённых в кратчайший остов }
for v ∈ V − u do
  if v ∈ Γ(u) then
    α[v] := u { u — ближайшая вершина остова }
    β[v] := C[u, v] { C[u, v] — длина соответствующего ребра }
  else
    α[v] := 0 { ближайшая вершина остова неизвестна }
    β[v] := ∞ { и расстояние также неизвестно }
  end if
end for
for i from 1 to p − 1 do
```

[1] Роберт Клей Прим (р. 1921).

```
    x := ∞ { начальное значение для поиска ближайшей вершины }
    for v ∈ V \ S do
      if β[v] < x then
        w := v { нашли более близкую вершину }
        x := β[v] { и расстояние до неё }
      end if
    end for
    S := S + w { добавляем найденную вершину в остов }
    T := T + (α[w], w) { добавляем найденное ребро в остов }
    for v ∈ Γ(w) do
      if v ∉ S then
        if β[v] > C[v, w] then
          α[v] := w { изменяем ближайшую вершину остова }
          β[v] := C[v, w] { и длину ведущего к ней ребра }
        end if
      end if
    end for
  end for
```

Обоснование Алгоритм Прима буквально следует схеме алгоритма 9.11. В качестве первого из соединяемых деревьев используется одно и то же разрастающееся дерево T, а в качестве второго — ближайшая одиночная вершина. □

ОТСТУПЛЕНИЕ

Задача о нахождении кратчайшего остова принадлежит к числу немногих задач теории графов, которые можно считать полностью решёнными. Между тем, если изменить условия задачи, на первый взгляд даже незначительно, то она оказывается значительно более трудной. Рассмотрим следующую задачу. Пусть задано множество городов на плоскости и нужно определить минимальный (по сумме расстояний) набор железнодорожных линий, который позволил бы переехать из любого города в любой другой. Кратчайший остов полного графа расстояний между городами не будет являться решением этой (практически, очевидно, очень важной) задачи, известной как *задача Штейнера*. В задаче Штейнера допускается введение дополнительных вершин, называемых *точками Штейнера*. На рис. 9.18 приведены, соответственно, диаграммы кратчайшего остова, наивного «решения» задачи Штейнера и правильного решения для случая, когда города расположены в вершинах квадрата. Задача Штейнера на плоскости хорошо изучена, в частности, известно много важных свойств точного решения. Например, известно, что каждая точка Штейнера имеет степень 3 и отрезки в ней встречаются под углом 120°. Тем не менее до сих пор неизвестно достаточно эффективных алгоритмов решения данной задачи, и она остаётся предметом интенсивных исследований.

Комментарии

Материал этой главы затрагивает вопросы, которые очень часто возникают в практическом программировании. Поэтому различные сведения о деревьях можно найти не только в специальных учебниках по теории графов, но и в книгах по

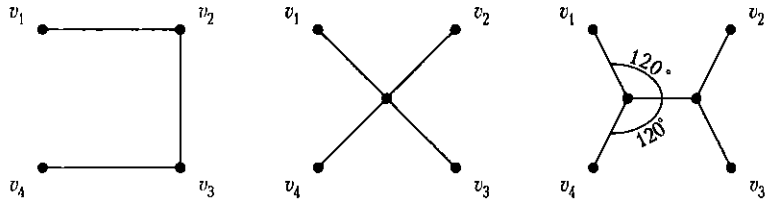

Рис. 9.18. Кратчайший остов, приближённое и точное решения задачи Штейнера

программированию и конструированию эффективных алгоритмов. В качестве рекомендуемых для программистов источников назовём [2], [14], [26]. Алгоритмы 9.1 и 9.2 заимствованы из редкой книги [6], в которой приведено большое число алгоритмов на графах, в том числе не очень широко известных. Алгоритмы 9.3 и 9.4 — программистский фольклор, переработанный автором. Алгоритмы 9.5–9.7 общеизвестны. Алгоритмы операций с деревом сортировки (алгоритмы 9.8–9.10) описаны в [14], откуда они заимствованы с некоторыми дополнительными уточнениями способов реализации. В книге [26] можно найти краткие и доступные описания алгоритмов работы с АВЛ-деревьями. Подробное изложение и анализ алгоритмов работы с различными классами деревьев имеется в прекрасном учебнике [27]. Алгоритмы поиска кратчайшего остова (алгоритмы 9.11–9.13) принадлежат к числу известных классических алгоритмов, их описание можно найти во многих источниках, например, в [21].

Упражнения

9.1. Нарисовать диаграммы всех деревьев с 7 вершинами.

9.2. Допустим, что в ордереве все узлы, кроме листьев, имеют одну и ту же полустепень исхода n. В этом случае говорят, что дерево имеет постоянную *ширину ветвления* n. Оценить высоту h ордерева, которое имеет p узлов и постоянную ширину ветвления n.

9.3. Составить алгоритм преобразования обратной польской записи арифметического выражения в прямую польскую запись.

9.4. Какой вид будет иметь дерево сортировки после того, как в него последовательно добавили следующие *текстовые* элементы: «1», «2», «3», «4», «5», «6», «7», «8», «9», «10», «11», «12», «13», «14», «15», «16», «17», «18», «19»?

9.5. Доказать, что полный граф K_p имеет $p^{(p-2)}$ остовов (это утверждение известно как *формула Кэли*[1].).

[1] Артур Кэли (1821–1895)

Глава 10 Циклы, независимость и раскраска

После рассмотрения ациклических связных графов, то есть деревьев, естественно перейти к рассмотрению графов с циклами. Некоторые задачи, связанные с циклами, в частности, с гамильтоновыми циклами, оказываются труднорешаемыми, так называемыми переборными задачами. В этой главе на примере задачи отыскания наибольшего независимого множества вершин рассматриваются подходы к программному решению таких задач. К числу переборных задач относится и задача раскрашивания графов, которая имеет много приложений в программировании. С раскраской связана задача об укладывании графа на заданной поверхности, имеющая приложения в электронике.

Таким образом, в этой заключительной главе рассматриваются некоторые известные задачи на графах и указываются связи между ними. В частности, приводятся решения тех исторических задач, с которых мы начали изложение теории графов в главе 7.

10.1. Фундаментальные циклы и разрезы

Первый раздел главы посвящён установлению структуры множеств циклов и разрезов в графе, с точки зрения векторных пространств.

10.1.1. Циклы и разрезы

Цикл может входить только в одну компоненту связности графа $G(V, E)$, а в несвязном графе понятие разреза является вырожденным, поэтому без ограничения общности в этом разделе граф $G(V, E)$ считается связным.

Цикл не может содержать одно ребро более одного раза, поэтому в этом разделе цикл рассматривается как множество рёбер. В этой связи можно дать эквивалентное определение простого цикла: *простым* называется цикл, никакое собственное подмножество которого циклом не является.

Напомним, что *разрезом* связного графа называется множество рёбер, удаление которых делает граф несвязным. Заметим, что любое разбиение множества вершин V на два непустых подмножества $V_1 \neq \varnothing$ и $V_2 \neq \varnothing$, $V_1 \cap V_2 = \varnothing$, $V_1 \cup V_2 = V$, определяет разрез $S := \{(v_1, v_2) \in E \mid v_1 \in V_1 \ \& \ v_2 \in V_2\}$, поскольку правильные подграфы G_1 и G_2, определяемые подмножествами V_1 и V_2 соответственно, являются, очевидно, компонентами связности графа $G - S$. Заметим далее, что множества V_1 и V_2 определяют друг друга: $V_1 = V \setminus V_2$, $V_2 = V \setminus V_1$, поэтому достаточно задать только одно из них. Естественно ввести обозначение

$$\forall U \subset V \ \left(\overline{U} \stackrel{\text{Def}}{=} V \setminus U\right).$$

Введём обозначение $E(V_1, V_2)$ для множества рёбер, соединяющих два дизъюнктных непустых подмножества вершин графа $G(V, E)$:

$$E(V_1, V_2) \stackrel{\text{Def}}{=} \{(v_1, v_2) \in E \mid v_1 \in V_1 \ \& \ v_2 \in V_2\},$$

где $V_1 \subset V$, $V_2 \subset V$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$, $V_1 \cap V_2 = \varnothing$. Заметим, что $E(V_1, V_2) = E(V_2, V_1)$.

Разрез связного графа $G(V, E)$, определяемый непустым подмножеством U множества вершин V, называется *правильным* разрезом и обозначается $S(U)$:

$$S(U) \stackrel{\text{Def}}{=} \left\{(v_1, v_2) \in E \mid v_1 \in U \ \& \ v_2 \in \overline{U}\right\} = E(U, \overline{U}).$$

Правильный разрез не содержит «лишних» рёбер, то есть таких рёбер, включение или исключение которых не меняет компонент связности, получаемых при удалении рёбер разреза. Ясно, что всякий разрез содержит некоторый правильный разрез.

ЛЕММА *Симметрическая разность двух различных правильных разрезов, определяемых множествами V_1 и V_2, является правильным разрезом, определяемым симметрической разностью множеств V_1 и V_2:*

$$V_1 \neq V_2 \Longrightarrow S(V_1) \triangle S(V_2) = S(V_1 \triangle V_2).$$

Доказательство «Пересечение» правильных разрезов $S(V_1)$ и $S(V_2)$ образует разбиение множества вершин V на четыре подмножества (рис. 10.1):

$$V_{11} := V_1 \cap V_2, \quad V_{10} := V_1 \cap \overline{V_2}, \quad V_{01} := \overline{V_1} \cap V_2, \quad V_{00} := \overline{V_1} \cap \overline{V_2}.$$

В этих обозначениях

$$S(V_1) = E(V_{11}, V_{01}) \cup E(V_{10}, V_{01}) \cup E(V_{11}, V_{00}) \cup E(V_{10}, V_{00}),$$
$$S(V_2) = E(V_{11}, V_{10}) \cup E(V_{01}, V_{10}) \cup E(V_{11}, V_{00}) \cup E(V_{01}, V_{00}),$$

откуда, учитывая, что $E(V_{10}, V_{01}) = E(V_{01}, V_{10})$, имеем:

$$S(V_1) \triangle S(V_2) = E(V_{11}, V_{01}) \cup E(V_{10}, V_{00}) \cup E(V_{11}, V_{10}) \cup E(V_{01}, V_{00}).$$

Заметим, что

$$V_1 \triangle V_2 = (V_1 \cap \overline{V_2}) \cup (\overline{V_1} \cap V_2) = V_{10} \cup V_{01},$$
$$\overline{V_1 \triangle V_2} = (V_1 \cap V_2) \cup (\overline{V_1} \cap \overline{V_2}) = V_{11} \cup V_{00}.$$

Поэтому

$$S(V_1 \vartriangle V_2) = E(V_{10}, V_{11}) \cup E(V_{10}, V_{00}) \cup E(V_{01}, V_{11}) \cup E(V_{01}, V_{00}),$$

и учитывая, что $E(V_{10}, V_{11}) = E(V_{11}, V_{10})$ и $E(V_{01}, V_{11}) = E(V_{11}, V_{01})$, окончательно имеем $S(V_1) \vartriangle S(V_2) = S(V_1 \vartriangle V_2)$. □

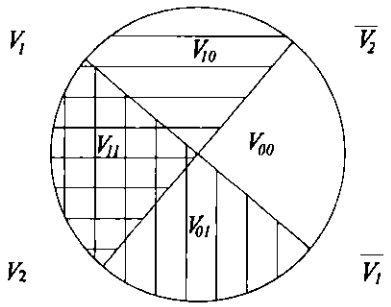

Рис. 10.1. К доказательству леммы 10.1.1

Простым разрезом называется минимальный разрез, то есть такой разрез, никакое собственное подмножество которого разрезом не является. Простой разрез является правильным.

ЗАМЕЧАНИЕ

Чем больше в графе циклов, тем труднее его разрёзать. В дереве, напротив, каждое ребро само по себе является разрезом.

10.1.2. Фундаментальная система циклов и циклический ранг

Пусть $T(V, E_T)$ — некоторый остов графа $G(V, E)$. *Кодеревом* $T^*(V, E_T^*)$ остова T называется остовный подграф, такой, что $E_T^* = E \setminus E_T$. (Кодерево не является деревом!) Рёбра кодерева называются *хордами* остова. По теореме 9.1.2 об основных свойствах деревьев каждая хорда $e \in T^*$ остова T порождает ровно один простой цикл, обозначим его Z_e. Таким образом, имеем систему простых циклов

$$\mathfrak{Z} \stackrel{\text{Def}}{=} \{Z_e\}_{e \in T^*},$$

определяемых выбранным остовом T, которая называется *фундаментальной системой циклов*. Циклы фундаментальной системы называются *фундаментальными*, а количество циклов в (данной) фундаментальной системе называется *циклическим рангом* (или *цикломатическим числом*) графа G и обозначается $m(G)$.

ТЕОРЕМА *Любой цикл в связном графе $G(V, E)$ можно представить как симметрическую разность нескольких фундаментальных циклов из системы $\mathfrak{Z}$, определяемой произвольным остовом T.*

Доказательство Если в графе G нет циклов, то это дерево, $G = T, T^* = \varnothing, \mathfrak{Z} = \varnothing$, и утверждение теоремы тривиально. Рассмотрим цикл Z в графе G. Этот цикл содержит хорды $e_1, \ldots, e_n \in T^*$. (Такие хорды в Z обязательно есть, в противном случае $Z \subset T$, что невозможно, поскольку T — дерево.) Докажем индукцией по n, что $Z = Z_{e_1} \triangle \ldots \triangle Z_{e_n}$. База: пусть $n = 1$, тогда $e_1 \notin T$, $Z - e_1 \subset T$ и $Z = Z_{e_1}$, так как если бы $Z \neq Z_{e_1}$, то концы e_1 были бы соединены в T двумя цепями, что невозможно по теореме 9.1.2. Пусть (индукционное предположение) $Z = Z_{e_1} \triangle \ldots \triangle Z_{e_m}$ для всех циклов Z с числом хорд $m < n$. Рассмотрим цикл Z с n хордами $e_1, \ldots, e_n \in T^*$ и цикл Z_{e_n} (рис. 10.2). Имеем $Z' := Z \triangle Z_{e_n} = (Z - e_n) \cup (Z_{e_n} - e_n)$ — тоже цикл (возможно, не простой). Но Z' содержит только $n-1$ хорд $e_1, \ldots, e_{n-1}$. По индукционному предположению $Z' = Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}}$. Имеем:

$$Z = Z' \triangle Z_{e_n} = \left(Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}}\right) \triangle Z_{e_n} = Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}} \triangle Z_{e_n}. \quad \square$$

СЛЕДСТВИЕ *Количество циклов в фундаментальной системе равно числу хорд остова:* $m(G) = q - p + 1$.

Доказательство $m(G) = q(T^*) = q(G) - q(T) = q - (p-1) = q - p + 1$. $\square$

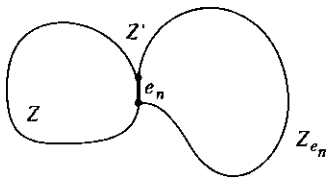

Рис. 10.2. К доказательству теоремы 10.1.2

Пример На рис. 10.3 представлена система фундаментальных циклов, определяемых некоторым остовом. Рёбра остова выделены жирными линиями, а соответствующие фундаментальные циклы, их четыре — $Z_1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$, $Z_2 = \{(v_2, v_5), (v_5, v_3), (v_3, v_2)\}$, $Z_3 = \{(v_2, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_2)\}$, $Z_4 = \{(v_2, v_4), (v_4, v_5), (v_5, v_2)\}$, — пунктирными контурами.

ЗАМЕЧАНИЕ

Совокупность всевозможных симметрических разностей фундаментальных циклов содержит множеств больше, чем нужно: в неё входят не только все циклы графа G, но и некоторые объединения таких циклов. В частности, симметрическая разность двух (фундаментальных) циклов, которые не имеют общих вершин, снова даёт два этих же цикла. Иногда множество объектов, определяемое всевозможными циклическими разностями фундаментальных циклов, называют множеством *циклических векторов*.

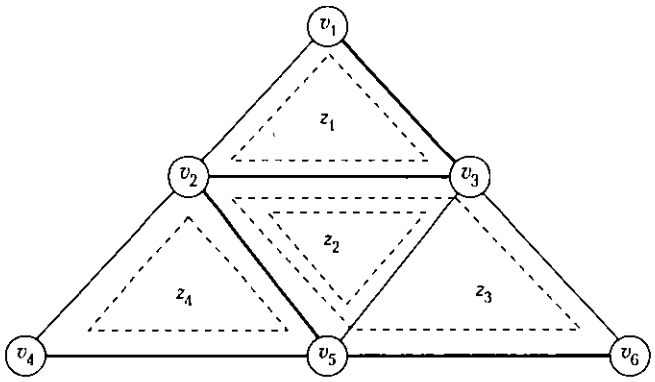

Рис. 10.3. Фундаментальные циклы

Пример На рис. 10.3 нет фундаментальных циклов, не имеющих общих вершин. Симметрическая разность трёх фундаментальных циклов $Z_1 \triangle Z_3 \triangle Z_4$ даёт цикл $(v_1, v_2), (v_2, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_1)$ — «внешнюю границу» графа. Симметрическая разность всех фундаментальных циклов $Z_1 \triangle Z_2 \triangle Z_3 \triangle Z_4$ даёт эйлеров цикл данного графа (см. следующий раздел).

10.1.3. Фундаментальная система разрезов и коциклический ранг

Пусть опять $T(V, E_T)$ — некоторый остов графа $G(V, E)$. Рассмотрим ребро остова $e \in E_T$ и определим разрез S_e следующим образом. Ребро e — мост в дереве T. Следовательно, удаление ребра e разбивает множество вершин V на два непустых подмножества V_1 и V_2, так что $V_1 \subset V$, $V_1 \neq \varnothing$, $V_2 \subset V$, $V_2 \neq \varnothing$, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$. Включим в разрез S_e ребро e и все хорды остова T, соединяющие вершины множества V_1 с вершинами множества V_2:

$$S_e \stackrel{\text{Def}}{=} e + \{(v_1, v_2) \in T^* \mid v_1 \in V_1 \ \& \ v_2 \in V_2\} = E(V_1, V_2).$$

Тогда S_e — это простой разрез. Система разрезов

$$\mathcal{S} \stackrel{\text{Def}}{=} \{S_e\}_{e \in T}$$

называется *фундаментальной системой разрезов*. Разрезы фундаментальной системы называются *фундаментальными*, а количество разрезов в (данной) фундаментальной системе называется *коциклическим рангом* (или *коцикломатическим числом*) графа G и обозначается $m^*(G)$.

ЗАМЕЧАНИЕ

Между циклами и разрезами существует определённая двойственность, поэтому разрезы иногда называют *коциклами*. Отсюда название «фундаментальная система коциклов» и «коциклический ранг». Детальное рассмотрение этой двойственности выходит за рамки данной книги.

ТЕОРЕМА *Любой правильный разрез в связном графе $G(V, E)$ можно представить как симметрическую разность некоторых фундаментальных разрезов из системы $\mathcal{S}$, определяемой произвольным остовом T.*

Доказательство Действительно, любой разрез S содержит хотя бы одно ребро из остова T, так как T — связный остовный подграф. Пусть S — правильный разрез, который содержит рёбра $e_1, \ldots, e_n \in T$. Докажем индукцией по n, что $S = S_{e_1} \triangle \ldots \triangle S_{e_n}$. База: пусть $n = 1$, тогда $T - e_1 = T_1 \cup T_2$, где T_1 и T_2 — компоненты, получаемые из остова T удалением ребра e_1. Имеем $S_{e_1} \subset S$, иначе S не был бы разрезом, и $S \subset S_{e_1}$, иначе S не был бы правильным разрезом. Таким образом, $S = S_{e_1}$. Пусть теперь $S = S_{e_1} \triangle \ldots \triangle S_{e_m}$ для всех правильных разрезов S с числом рёбер остова $m < n$. Рассмотрим правильный разрез S с n рёбрами $e_1, \ldots, e_n \in T$. Положим $S' := S \triangle S_{e_n}$, то есть исключим из разреза S все рёбра фундаментального разреза S_{e_n}. Разрез S' — правильный по лемме 10.1.1 и содержит рёбра $e_1, \ldots, e_{n-1}$, а значит, по индукционному предположению $S' = S_{e_1} \triangle \ldots \triangle S_{e_{n-1}}$. Имеем $S = S' \triangle S_{e_n}$, и окончательно $S = S_{e_1} \triangle \ldots \triangle S_{n-1} \triangle S_{e_n}$. □

СЛЕДСТВИЕ *Количество разрезов в фундаментальной системе равно числу рёбер остова:*

$$m^*(G) = p - 1.$$

Доказательство По определению. □

Пример На рис. 10.4 представлены все пять фундаментальных разрезов, соответствующих графу и его остову, представленным на рис. 10.3

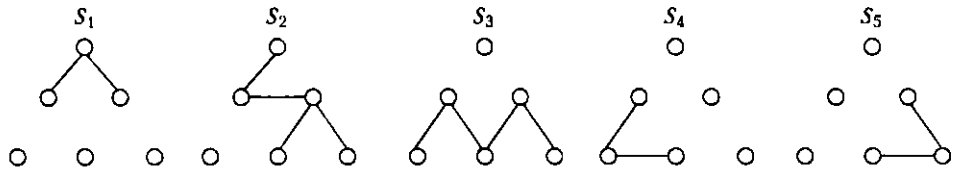

Рис. 10.4. Фундаментальные разрезы

10.1.4. Подпространства циклов и коциклов

Рассмотрим векторное пространство (см. 2.4.1) над двоичной арифметикой (см. пример 3 в 2.3.3), натянутое на множество рёбер E графа $G(V, E)$. Элемент этого векторного пространства (линейную комбинацию, см. 2.4.2) можно отождествлять с подмножеством рёбер: если коэффициент в линейной комбинации равен 1, то ребро входит в подмножество, если коэффициент равен 0, то не входит. При такой интерпретации сложению векторов соответствует симметрическая разность множеств рёбер.

Множество циклических векторов и множество правильных разрезов замкнуты относительно симметрической разности, а потому являются подпространствами общего пространства подмножеств рёбер графа.

ЗАМЕЧАНИЕ

Строго говоря, множество циклов, равно как и множество разрезов, не образует подпространства, поскольку не замкнуто относительно симметрической разности. Множество циклических векторов шире множества циклов, а множество правильных разрезов у́же множества разрезов. Однако интуитивно ясно, что «по потребительским свойствам» циклические векторы и циклы, равно как разрезы и правильные разрезы, весьма близки, поэтому далее в этом подразделе рассматриваются только циклические векторы и правильные разрезы, которые для краткости называются циклами и разрезами (коциклами).

Множество циклов $\{Z_i\}_{i=1}^n$ называется *независимым*, если ни один из циклов Z_i не является линейной комбинацией остальных.

Множество разрезов $\{S_i\}_{i=1}^n$ называется *независимым*, если ни один из разрезов S_i не является линейной комбинацией остальных.

Максимальное независимое множество циклов (или минимальное множество циклов, от которых зависят все остальные, или независимое порождающее множество циклов) является *базисом* пространства циклов. Максимальное независимое множество разрезов (или минимальное множество разрезов, от которых зависят все остальные, или независимое порождающее множество разрезов) является *базисом* пространства разрезов.

Введенная в подразделе 10.1.2 фундаментальная система циклов является базисом подпространства циклов. Действительно, циклы любой фундаментальной системы $\mathfrak{Z} = \{Z_e\}_{e \in T^*}$ независимы, поскольку каждый из них содержит индивидуальную хорду $e \in T^*$, и по теореме 10.1.2 фундаментальная система порождает множество циклов. Таким образом, циклический ранг — это размерность пространства циклов.

ЗАМЕЧАНИЕ

Фундаментальные системы циклов, порождаемые остовами, — это не единственные базисы пространства циклов. Например, четыре «естественных» треугольника на рис. 10.3 — $\{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$, $\{(v_2, v_4), (v_4, v_5), (v_5, v_2)\}$, $\{(v_2, v_5), (v_5, v_3), (v_3, v_2)\}$, $\{(v_3, v_5), (v_5, v_6), (v_6, v_3)\}$ — являются базисом подпространства циклов, но не являются фундаментальной системой ни для какого остова.

Введенная в подразделе 10.1.3 фундаментальная система разрезов является базисом подпространства разрезов. Действительно, разрезы любой фундаментальной системы $\mathfrak{S} = \{S_e\}_{e \in T}$ независимы, поскольку каждый из них содержит индивидуальное ребро $e \in T$, и по теореме 10.1.3 фундаментальная система порождает множество разрезов. Таким образом, коциклический ранг — это размерность пространства разрезов.

ЗАМЕЧАНИЕ

Все утверждения этого и следующего разделов распространяются на случай мультиграфов.

10.2. Эйлеровы циклы

Здесь приведено исчерпывающее решение задачи о Кёнигсбергских мостах (см. подраздел 7.1.1), приведшей к исторически первой успешной попытке развития теории графов как самостоятельного предмета.

10.2.1. Эйлеровы графы

Если граф имеет цикл (не обязательно простой), содержащий все рёбра графа, то такой цикл называется *эйлеровым* циклом, а граф называется *эйлеровым* графом. Если граф имеет цепь (не обязательно простую), содержащую все рёбра, то такая цепь называется *эйлеровой* цепью, а граф называется *полуэйлеровым* графом.

Эйлеров цикл содержит не только все рёбра (по одному разу), но и все вершины графа (возможно, по несколько раз). Ясно, что эйлеровым может быть только связный граф.

ТЕОРЕМА *Если граф G связен и нетривиален, то следующие утверждения эквивалентны:*

1. *G — эйлеров граф.*
2. *Каждая вершина G имеет чётную степень.*
3. *Множество рёбер G можно разбить на простые циклы.*

Доказательство

[1$\Longrightarrow$ 2] Пусть Z — эйлеров цикл в G. Двигаясь по Z, будем подсчитывать степени вершин, полагая их до начала прохождения нулевыми. Прохождение каждой вершины вносит 2 в степень этой вершины. Поскольку Z содержит все рёбра, то, когда обход Z будет закончен, будут учтены все рёбра, а степени всех вершин — чётные.

[2$\Longrightarrow$ 3] G — связный и нетривиальный граф, следовательно, $\forall v_i \ \ (d(v_i) > 0)$. Степени вершин чётные, следовательно, $\forall v_i \ \ (d(v_i) \geqslant 2)$. Имеем:

$$2q = \sum_{i=1}^{p} d(v_i) \geqslant 2p \Longrightarrow q \geqslant p \Longrightarrow q > p - 1.$$

Следовательно, граф G — не дерево, а значит, граф G содержит (хотя бы один) простой цикл Z_1. (Z_1 — множество рёбер.) Тогда $G - Z_1$ — остовный подграф, в котором опять все степени вершин чётные. Исключим из рассмотрения изолированные вершины. Таким образом, $G - Z_1$ тоже удовлетворяет условию 2, следовательно, существует простой цикл $Z_2 \subset (G - Z_1)$. Далее выделяем циклы Z_i, пока граф не будет пуст. Имеем: $E = \bigcup Z_i$ и $\bigcap Z_i = \varnothing$.

[3$\Longrightarrow$1] Возьмем какой-либо цикл Z_1 из данного разбиения. Если $Z_1 = E$, то теорема доказана. Если нет, то существует цикл Z_2, не имеющий общих рёбер с Z_1 (см. рис. 10.5), такой, что $\exists v_1 \ \ ((v_1 \in Z_1 \ \& \ v_1 \in Z_2))$, так как G связен. Маршрут $Z_1 \cup Z_2$ является циклом и содержит все свои рёбра по одному разу.

Если $Z_1 \cup Z_2 = E$, то теорема доказана. Если нет, то существует цикл Z_3, такой, что $\exists\, v_2\ (v_2 \in Z_1 \cup Z_2\ \&\ v_2 \in Z_3)$. Далее будем наращивать эйлеров цикл, пока он не исчерпает разбиения. □

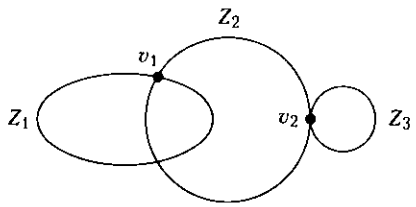

Рис. 10.5. К доказательству теоремы 10.2.1

10.2.2. Алгоритм построения эйлерова цикла в эйлеровом графе

В предыдущем разделе был установлен эффективный способ проверки наличия эйлерова цикла в графе. А именно, для этого достаточно убедиться, что степени всех вершин чётные, что нетрудно сделать при любом представлении графа. Следующий алгоритм находит эйлеров цикл в графе, если известно, что граф эйлеров.

Алгоритм 10.1 Построение эйлерова цикла

Вход: эйлеров граф $G(V, E)$, заданный списками смежности ($\Gamma[v]$ — список вершин, смежных с вершиной v).
Выход: последовательность вершин эйлерова цикла.

```
S := ∅ { стек для хранения вершин }
select v ∈ V { произвольная вершина }
v → S { положить v в стек S }
while S ≠ ∅ do
  v := top S { v — верхний элемент стека }
  if Γ[v] = ∅ then
    v ← S; yield v { очередная вершина эйлерова цикла }
  else
    select u ∈ Γ[v] { взять первую вершину из списка смежности }
    u → S { положить u в стек }
    Γ[v] := Γ[v] − u; Γ[u] := Γ[u] − v { удалить ребро (v, u) }
  end if
end while
```

Обоснование Принцип действия этого алгоритма заключается в следующем. Начав с произвольной вершины v, строим путь, удаляя рёбра и запоминая вершины в стеке, до тех пор пока множество смежности очередной вершины не окажется пустым, что означает, что путь удлинить нельзя. Заметим, что при

этом мы с необходимостью придём в ту вершину, с которой начали. В противном случае вершина v имеет нечётную степень, что невозможно по условию. Таким образом, из графа были удалены рёбра цикла, а вершины цикла были сохранены в стеке S. Заметим, что при этом степени всех вершин остались чётными. Далее вершина v выводится в качестве первой вершины эйлерова цикла, а процесс продолжается с вершины, стоящей на вершине стека. □

10.2.3. Оценка числа эйлеровых графов

Пусть $\mathcal{G}(p)$ — множество всех графов с p вершинами, а $\mathcal{E}(p)$ — множество эйлеровых графов с p вершинами.

ЗАМЕЧАНИЕ

В этом подразделе речь идёт о числе *нумерованных* графов, то есть о числе графов, в которых вершины перенумерованы. Считается, что если перенумеровать вершины в другом порядке, то это будет другой граф. Очевидно, что нумерованных графов (любого типа) больше, чем графов, определяемых как классы эквивалентности по отношению изоморфизма, поэтому приводимые здесь оценки являются достаточно грубыми.

ТЕОРЕМА *Эйлеровых графов почти нет, то есть*

$$\lim_{p\to\infty} \frac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} = 0.$$

Доказательство Пусть $\mathcal{E}'(p)$ — множество графов с p вершинами и чётными степенями. Тогда по предыдущей теореме $\mathcal{E}(p) \subset \mathcal{E}'(p)$ и $|\mathcal{E}(p)| \leqslant |\mathcal{E}'(p)|$. В любом графе число вершин нечётной степени чётно, следовательно, любой граф из $\mathcal{E}'(p)$ можно получить из некоторого графа $\mathcal{G}(p-1)$, если добавить новую вершину и соединить её со всеми старыми вершинами нечётной степени. Следовательно, $|\mathcal{E}'(p)| \leqslant |\mathcal{G}(p-1)|$. Но $|\mathcal{G}(p)| = 2^{C(p,2)}$, поскольку (пумерованный) граф с p вершинами определяется подмножеством включённых в него рёбер, выбранных из множества всех возможных рёбер, а их $C(p,2)$.

Заметим, что

$$C(k,2) - C(k-1,2) = \frac{k!}{2!(k-2)!} - \frac{(k-1)!}{2!(k-3)!} = \frac{k(k-1)}{2} - \frac{(k-1)(k-2)}{2} = k-1.$$

Далее имеем:

$$|\mathcal{E}(p)| \leqslant |\mathcal{E}'(p)| \leqslant |\mathcal{G}(p-1)| = 2^{C(p-1,2)} = 2^{C(p,2)-(p-1)} = |\mathcal{G}(p)|2^{-(p-1)}$$

и $\dfrac{\mathcal{E}(p)}{\mathcal{G}(p)} \leqslant \dfrac{1}{2^{p-1}}$, откуда $\lim\limits_{p\to\infty} \dfrac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} = 0.$ □

10.3. Гамильтоновы циклы

Название «гамильтонов цикл» произошло от задачи «Кругосветное путешествие», придуманной Гамильтоном[1] в XIX веке: нужно обойти все вершины графа, диаграмма которого показана на рис. 10.6 (в исходной формулировке вершины были помечены названиями столиц различных стран), по одному разу и вернуться в исходную точку. Этот граф представляет собой укладку додекаэдра.

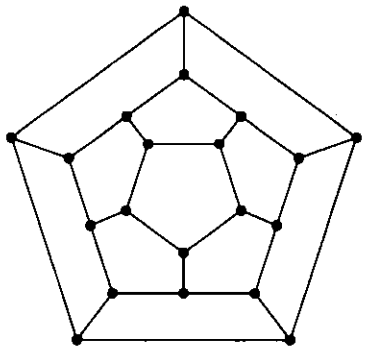

Рис. 10.6. Задача «Кругосветное путешествие»

10.3.1. Гамильтоновы графы

Если граф имеет простой цикл, содержащий все вершины графа (по одному разу), то такой цикл называется *гамильтоновым* циклом, а граф называется *гамильтоновым* графом.

Гамильтонов цикл не обязательно содержит все рёбра графа. Ясно, что гамильтоновым может быть только связный граф.

ЗАМЕЧАНИЕ

Любой граф G можно превратить в гамильтонов, добавив достаточное количество новых вершин и инцидентных им рёбер и не добавляя рёбер, инцидентных только старым вершинам. Для этого, например, достаточно к вершинам $v_1, \ldots, v_p$ графа G добавить вершины $u_1, \ldots, u_p$ и множество рёбер $\{(v_i, u_i)\} \cup \{(u_i, v_{i+1})\}$.

Простые необходимые и достаточные условия гамильтоновости графа неизвестны. Известны только некоторые достаточные условия, одно из которых приведено в следующей теореме.

[1] Уильям Гамильтон (1805–1856).

ТЕОРЕМА *Если $\delta(G) \geqslant p/2$, то граф G является гамильтоновым.*

Доказательство От противного. Пусть G — не гамильтонов. Добавим к G *минимальное* количество новых вершин $u_1, \ldots, u_n$, соединяя их со *всеми* вершинами G так, чтобы граф $G' := G + u_1 + \cdots + u_n$ был гамильтоновым.

Пусть $v, u_1, w, \ldots, v$ — гамильтонов цикл в графе G', причём $v \in G$, $u_1 \in G'$, $u_1 \notin G$. Такая пара вершин v и u_1 в гамильтоновом цикле обязательно найдется, иначе граф G был бы гамильтоновым. Тогда $w \in G$, $w \notin \{u_1, \ldots, u_n\}$, иначе вершина u_1 была бы не нужна. Более того, вершина v не смежна с вершиной w, иначе вершина u_1 была бы не нужна. Далее, если в цикле $v, u_1, w, \ldots, v', w', \ldots, v$ есть вершина w', смежная с вершиной w, то вершина v' не смежна с вершиной v, так как иначе можно было бы построить гамильтонов цикл $v, v', \ldots, w, w' \ldots v$ без вершины u_1, взяв последовательность вершин $w, \ldots, v'$ в обратном порядке. Отсюда следует, что число вершин графа G', не смежных с v, не менее числа вершин, смежных с w. Но для любой вершины w графа G $d(w) \geqslant p/2 + n$ по построению, в том числе $d(v) \geqslant p/2 + n$. Общее число вершин (смежных и не смежных с v, за исключением самой вершины v) составляет $n + p - 1$. Таким образом, имеем:

$$n + p - 1 = \overline{d}(v) + d(v) \geqslant d(w) + d(v) \geqslant \frac{p}{2} + n + \frac{p}{2} + n = 2n + p.$$

Следовательно, $0 \geqslant n + 1$, что противоречит тому, что $n > 0$. □

10.3.2. Задача коммивояжёра

Рассмотрим следующую задачу, известную как *задача коммивояжёра*. Имеется p городов, расстояния между которыми известны. Коммивояжёр должен посетить все p городов по одному разу, вернувшись в тот, с которого начал. Требуется найти такой маршрут движения, при котором суммарное пройденное расстояние будет минимальным.

Очевидно, что задача коммивояжёра — это задача отыскания кратчайшего гамильтонова цикла в нагруженном полном графе.

Можно предложить следующую простую схему решения задачи коммивояжёра: сгенерировать все $p!$ возможных перестановок вершин полного графа, подсчитать для каждой перестановки длину маршрута и выбрать из них кратчайший. Очевидно, такое вычисление потребует не менее $O(p!)$ шагов.

Как известно, $p!$ — быстро растущая функция. Таким образом, решение задачи коммивояжёра описанным методом *полного перебора* оказывается практически неосуществимым даже для сравнительно небольших p. Более того, известно, что задача коммивояжёра принадлежит к числу так называемых *NP-полных* задач, подробное обсуждение которых выходит за рамки этого учебника.

Вкратце суть проблемы *NP*-полноты сводится к следующему. В различных областях дискретной математики, комбинаторики, логики и т. п. известно множество задач, принадлежащих к числу наиболее фундаментальных, для которых, несмотря на все усилия, не удалось найти алгоритмов решения, имеющих полиномиальную сложность. Более того, если бы удалось отыскать эффектив-

ный алгоритм решения хотя бы одной из этих задач, то из этого немедленно следовало бы существование эффективных алгоритмов для всех остальных задач данного класса. На этом основано общепринятое мнение, что таких алгоритмов не существует.

ОТСТУПЛЕНИЕ

Полезно сопоставить задачи отыскания эйлеровых и гамильтоновых циклов, рассмотренные в этом и предыдущем разделах. Внешне формулировки этих задач очень похожи, однако они оказываются принципиально различными с точки зрения практического применения. Уже давно Эйлером получено просто проверяемое необходимое и достаточное условие существования в графе эйлерова цикла. Что касается гамильтоновых графов, то для них неизвестны простые необходимые и достаточные условия. На основе необходимого и достаточного условия существования эйлерова цикла можно построить эффективные алгоритмы отыскания такого цикла. В то же время задача проверки существования гамильтонова цикла оказывается *NP*-полной (так же как и задача коммивояжёра). Далее, известно, что почти нет эйлеровых графов, и эффективный алгоритм отыскания эйлеровых циклов редко оказывается применимым на практике. С другой стороны, можно показать, что почти все графы — гамильтоновы, то есть

$$\lim_{p\to\infty}\frac{|\mathcal{H}(p)|}{|\mathcal{G}(p)|}=1,$$

где $\mathcal{H}(p)$ — множество гамильтоновых графов с p вершинами, а $\mathcal{G}(p)$ — множество всех графов с p вершинами. Таким образом, задача отыскания гамильтонова цикла или задача коммивояжёра являются практически востребованными, но эффективный алгоритм решения для них неизвестен (и, скорее всего, не существует).

10.4. Независимые и покрывающие множества

Прежде всего введём определения и рассмотрим основные свойства независимых и покрывающих множеств вершин и рёбер. Эти определения и свойства используются в последующих разделах.

10.4.1. Покрывающие множества вершин и рёбер

Говорят, что вершина *покрывает* инцидентные ей рёбра, а ребро *покрывает* инцидентные ему вершины. Множество вершин, которые в совокупности покрывают все рёбра, называется *вершинным покрытием*. Наименьшее число вершин во всех вершинных покрытиях называется *числом вершинного покрытия* и обозначается α_0.

Примеры

1. $\alpha_0(K_p)=p-1$, если K_p — полный граф.
2. $\alpha_0(K_{m,n})=\min(m,n)$, если $K_{m,n}$ — полный двудольный граф.
3. $\alpha_0(\overline{K}_p)=0$, если $\overline{K}_p$ — вполне несвязный граф.
4. $\alpha_0(G_1\cup G_2)=\alpha_0(G_1)+\alpha_0(G_2)$, если G_1 и G_2 — компоненты несвязного графа.

Множество рёбер, которые в совокупности покрывают все вершины, называется *рёберным покрытием*. Наименьшее число рёбер во всех рёберных покрытиях называется *числом рёберного покрытия* и обозначается α_1.

ЗАМЕЧАНИЕ

Для графа с изолированными вершинами α_1 не определено.

Примеры

1. $\alpha_1(C_{2n}) = n$, если C_{2n} — чётный цикл.
2. $\alpha_1(C_{2n+1}) = n + 1$, если C_{2n+1} — нечётный цикл.
3. $\alpha_1(K_{2n}) = n$, если K_{2n} — полный граф с чётным числом вершин.
4. $\alpha_1(K_{2n+1}) = n + 1$, если K_{2n+1} — полный граф с нечётным числом вершин.
5. $\alpha_1(K_{m,n}) = \max(m, n)$, если $K_{m,n}$ — полный двудольный граф.

10.4.2. Независимые множества вершин и рёбер

Множество вершин называется *независимым* (или *внутренне устойчивым*), если никакие две из них не смежны. Наибольшее число вершин в независимом множестве вершин называется *вершинным числом независимости* и обозначается β_0.

Примеры

1. $\beta_0(K_p) = 1$, если K_p — полный граф .
2. $\beta_0(K_{m,n}) = \max(m, n)$, если $K_{m,n}$ — полный двудольный граф.
3. $\beta_0(\overline{K}_p) = p$, если $\overline{K}_p$ — вполне несвязный граф.
4. $\beta_0(G_1 \cup G_2) = \beta_0(G_1) + \beta_0(G_2)$, если G_1 и G_2 — компоненты несвязного графа.

Ясно, что множество вершин S является независимым тогда и только тогда, когда $\forall v \in S \ (\Gamma(v) \cap S = \varnothing)$.

Множество рёбер называется *независимым* (или *паросочетанием*), если никакие два из них не смежны. Наибольшее число рёбер в независимом множестве рёбер называется *рёберным числом независимости* и обозначается β_1.

Примеры

1. $\beta_1(C_{2n}) = n$, если C_{2n} — чётный цикл.
2. $\beta_1(C_{2n+1}) = n - 1$, если C_{2n+1} — нечётный цикл.
3. $\beta_1(K_{2n}) = n$, если K_{2n} — полный граф с чётным числом вершин.
4. $\beta_1(K_{2n+1}) = n$, если K_{2n+1} — полный граф с нечётным числом вершин.
5. $\beta_1(K_{m,n}) = \min(m, n)$, если $K_{m,n}$ — полный двудольный граф.

ЗАМЕЧАНИЕ

Индексы 0 и 1 в обозначениях α_0, α_1, β_0, β_1 выбраны из следующих мнемонических соображений. Индекс 0 соответствует вершинам, так как вершина нульмерна, а 1 — рёбрам, так как ребро одномерно.

10.4.3. Связь чисел независимости и покрытий

Приведённые примеры наводят на мысль, что числа независимости и покрытия связаны друг с другом и с количеством вершин p.

ТЕОРЕМА *Для любого нетривиального связного графа*

$$\alpha_0 + \beta_0 = p = \alpha_1 + \beta_1.$$

Доказательство Докажем четыре неравенства, из которых следуют два требуемых равенства.

[$\alpha_0 + \beta_0 \geqslant p$] Пусть M_0 — наименьшее вершинное покрытие, то есть $|M_0| = \alpha_0$. Рассмотрим $V \setminus M_0$. Тогда $V \setminus M_0$ — независимое множество, так как если бы в множестве $V \setminus M_0$ были смежные вершины, то M_0 не было бы покрытием. Имеем $|V \setminus M_0| \leqslant \beta_0$, следовательно, $p = |M_0| + |V \setminus M_0| \leqslant \alpha_0 + \beta_0$.

[$\alpha_0 + \beta_0 \leqslant p$] Пусть N_0 — наибольшее независимое множество вершин, то есть $|N_0| = \beta_0$. Рассмотрим $V \setminus N_0$. Тогда $V \setminus N_0$ — вершинное покрытие, так как нет рёбер, инцидентных только вершинам из N_0, стало быть, любое ребро инцидентно вершине (или вершинам) из $V \setminus N_0$. Имеем $|V \setminus N_0| \geqslant \alpha_0$, следовательно, $p = |N_0| + |V \setminus N_0| \geqslant \beta_0 + \alpha_0$.

[$\alpha_1 + \beta_1 \geqslant p$] Пусть M_1 — наименьшее рёберное покрытие, то есть $|M_1| = \alpha_1$. Множество M_1 не содержит цепей длиной больше 2. Действительно, если бы в M_1 была цепь длиной 3, то среднее ребро этой цепи можно было бы удалить из M_1 и это множество все равно осталось бы покрытием. Следовательно, M_1 состоит из звёзд. (*Звездой* называется граф, диаметр которого не превосходит двух, $D(G) \leqslant 2$, и имеется одна центральная вершина, смежная с остальными, а остальные не смежны между собой. Другими словами, звезда — это полный двудольный граф $K_{1,n}$.) Пусть этих звёзд m. Имеем $|M_1| = \sum_{i=1}^{m} n_i$, где n_i — число рёбер в i-й звезде. Заметим, что звезда из n_i рёбер покрывает n_i+1 вершину. Имеем: $p = \sum_{i=1}^{m}(n_i + 1) = m + \sum_{i=1}^{m} n_i = m + |M_1|$. Возьмём по одному ребру из каждой звезды и составим из них множество X. Тогда $|X| = m$, множество X — независимое, то есть $|X| \leqslant \beta_1$. Следовательно, $p = |M_1| + m = |M_1|+|X| \leqslant \alpha_1+\beta_1$.

[$\alpha_1 + \beta_1 \leqslant p$] Пусть N_1 — наибольшее независимое множество рёбер, то есть $|N_1| = \beta_1$. Построим рёберное покрытие Y следующим образом. Множество N_1 покрывает $2|N_1|$ вершин. Добавим по одному ребру, инцидентному непокрытым $p - 2|N_1|$ вершинам, таких рёбер $p - 2|N_1|$. Тогда множество Y — рёберное покрытие, то есть $|Y| \geqslant \alpha_1$ и $|Y| = |N_1| + p - 2|N_1| = p - |N_1|$. Имеем: $p = p - |N_1| + |N_1| = |Y| + |N_1| \geqslant \alpha_1 + \beta_1$. □

ЗАМЕЧАНИЕ

Условия связности и нетривиальности гарантируют, что все четыре инварианта определены. Хотя это условие является достаточным, оно не является необходимым. Например, для графа $K_n \cup K_n$ заключение теоремы остается справедливым, хотя условие не выполнено.

ОТСТУПЛЕНИЕ

Задача отыскания экстремальных независимых и покрывающих множеств возникает во многих практических случаях. Например, пусть дано множество процессов, использующих неразделяемые ресурсы. Соединим рёбрами вершины, соответствующие процессам, которым требуется один и тот же ресурс. Тогда β_0 будет определять количество возможных параллельных процессов.

10.5. Построение независимых множеств вершин

Этот раздел, фактически, является вводным обзором методов решения переборных задач. Методы рассматриваются на примере задачи отыскания максимального независимого множества вершин графа. Наш обзор не претендует на полноту, но описание основополагающих идей и терминов в нём присутствует.

10.5.1. Постановка задачи отыскания наибольшего независимого множества вершин

Задача отыскания наибольшего независимого множества вершин (и тем самым определения вершинного числа независимости β_0) принадлежит к числу трудоёмких.

Эту задачу можно поставить следующим образом. Пусть задан граф $G(V, E)$. Найти такое множество вершин X, $X \subset V$, что

$$w(X) = \max_{Y \in \mathcal{E}} w(Y),$$

где $w(Y) \stackrel{\text{Def}}{=} |Y|$, а $\mathcal{E} \stackrel{\text{Def}}{=} \{Y \subset V \mid \forall u, v \in Y \ ((u, v) \notin E)\}$ (см. 2.6.4).

Если бы семейство $\mathcal{E}$ независимых множеств оказалось матроидом, то поставленную задачу можно было бы решить жадным алгоритмом (алгоритм 2.2). Однако семейство $\mathcal{E}$ матроидом не является. Действительно, хотя аксиомы M_1 и M_2 (см. 2.6.1) выполнены, так как пустое множество вершин независимо и всякое подмножество независимого множества независимо, но аксиома M_3 не выполнена, как видно из следующего примера.

Пример Рассмотрим полный двудольный граф $K_{n,n+1}$. Доли этого графа образуют (максимальные) независимые множества, и их мощности различаются на 1. Однако никакая вершина из большей доли не может быть добавлена к вершинам меньшей доли с сохранением независимости.

10.5.2. Поиск с возвратами

Даже если для решения задач, подобных поставленной в предыдущем подразделе, не удаётся найти эффективного алгоритма, остаётся возможность попробовать найти решение «полным перебором» всех возможных вариантов, просто в силу конечности числа возможностей. Например, наибольшее независимое множество можно найти по следующей схеме.

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0 { наилучшее известное значение β0 }
for Y ∈ 2^V do
  if Y ∈ 𝓔 & |Y| > m then
    m := |Y|; X := Y { наилучшее известное значение X }
  end if
end for
```

ЗАМЕЧАНИЕ

Для выполнения этого алгоритма потребуется $O(2^p)$ шагов.

ОТСТУПЛЕНИЕ

Алгоритм, трудоёмкость которого (число шагов) ограничена полиномом от характерного размера задачи, принято называть *эффективным*, в противоположность *неэффективным* алгоритмам, трудоёмкость которых ограничена функцией, растущей быстрее, например, экспонентой. Таким образом, жадный алгоритм эффективен, а полный перебор — нет.

При решении переборных задач большое значение имеет способ организации перебора (в нашем случае — способ построения и последовательность перечисления множеств Y). Наиболее популярным является следующий способ организации перебора, основанный на идее поиска в глубину и называемый *поиском с возвратами*.

ЗАМЕЧАНИЕ

Иногда употребляется термин *бэктрекинг* (транслитерация английского названия этого метода — *backtracking*).

Идея поиска с возвратами состоит в следующем. Находясь в некоторой ситуации, пробуем изменить её допустимым образом в надежде найти решение. Если изменение не привело к успеху, то возвращаемся в исходную ситуацию (отсюда название «поиск с возвратами») и пробуем изменить её другим образом, и так до тех пор, пока не будут перебраны все возможности.

Для рассматриваемой задачи отыскания наибольшего независимого множества вершин метод поиска с возвратами может быть реализован с помощью следующего рекурсивного алгоритма.

Алгоритм 10.2 Поиск с возвратами

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0 { наилучшее известное значение β0 }
X := ∅ { наибольшее известное независимое множество X }
BT(∅, V) { вызов рекурсивной процедуры BT }
```

Основная работа выполняется рекурсивной процедурой BT.

Вход: S — текущее независимое множество вершин, T — оставшиеся вершины графа.
Выход: изменение глобальной переменной X, если текущее множество не может быть расширено (является максимальным).

```
for v ∈ T do
  if S + v ∈ E then
    if |S| > m then
      X := S; m := |S| { наибольшее известное независимое множество }
    end if
    BT(S + v, T \ Γ*(v)) { пробуем добавить v }
  end if
end for
```

Обоснование По построению вершина v добавляется в множество S только при сохранении независимости расширенного множества. В алгоритме это обстоятельство указано в форме условия $S + v \in \mathcal{E}$. Проверить сохранение условия независимости нетрудно, например, с помощью следующей функции.

Вход: независимое множество S и проверяемая вершина v.
Выход: **true**, если множество $S + v$ независимое, **false** — в противном случае.

```
for u ∈ S do
  if (u, v) ∈ E then
    return false { множество S + v зависимое }
  end if
end for
return true { множество S + v независимое }
```

Этот цикл не включен в явном виде в рекурсивную процедуру BT, чтобы не загромождать основной текст и не затуманивать идею поиска с возвратами. Таким образом, множество S, а следовательно, и множество X — независимые. В тот момент, когда множество S нельзя расширить, оно максимально по определению. Переменная m глобальна, поэтому среди всех максимальных независимых множеств в конце работы алгоритма построенное множество X является наибольшим независимым множеством вершин. □

10.5.3. Улучшенный перебор

Применение метода поиска с возвратами не гарантирует эффективности — трудоёмкость поиска с возвратами имеет тот же порядок, что и другие способы перебора (в худшем случае).

Используя конкретную информацию о задаче, в некоторых случаях можно существенно сократить трудоёмкость выполнения каждого шага перебора или уменьшить количество перебираемых возможностей в среднем (при сохранении оценки количества шагов в худшем случае). Такие приёмы называются методами улучшения перебора. Например, может оказаться, что некоторые варианты заведомо не могут привести к решению, а потому их можно не рассматривать.

ЗАМЕЧАНИЕ

Рекурсивная форма метода поиска с возвратами удобна для понимания, но не является самой эффективной. По сути, рекурсия здесь используется для сохранения *контекста* – то есть информации, характеризующей текущий рассматриваемый вариант. Если использовать другие методы сохранения контекста, то поиск с возвратами может быть реализован без явной рекурсии, а значит, более эффективно.

Рассмотрим методы улучшения перебора на примере задачи отыскания всех максимальных независимых множеств вершин.

Идея: начинаем с пустого множества и пополняем его вершинами с сохранением независимости (пока возможно).

Пусть S_k — уже полученное множество из k вершин, Q_k — множество вершин, которое можно добавить к S_k, то есть $S_k \cap \Gamma(Q_k) = \varnothing$. Среди вершин Q_k будем различать те, которые уже использовались для расширения S_k (обозначим их множество Q_k^-), и те, которые еще не использовались (Q_k^+). Тогда общая схема нерекурсивной реализации поиска с возвратами будет состоять из следующих шагов.

Шаг вперед от k к $k+1$ состоит в выборе вершины $x \in Q_k^+$:

$$S_{k+1} := S_k + x,$$
$$Q_{k+1}^- := Q_k^- \setminus \Gamma^+(x),$$
$$Q_{k+1}^+ := Q_k^+ \setminus \Gamma^*(x).$$

Шаг назад от $k+1$ к k:

$$S_k := S_{k+1} - x,$$
$$Q_k^+ := Q_k^+ - x,$$
$$Q_k^- := Q_k^- + x.$$

Если S_k — максимальное, то $Q_k^+ = \varnothing$. Если $Q_k^- \neq \varnothing$, то S_k было расширено раньше и не является максимальным. Таким образом, проверка максимальности задается следующим условием: $Q_k^+ = Q_k^- = \varnothing$.

Перебор можно улучшить, если заметить следующее.

Пусть $x \in Q_k^-$ и $\Gamma(x) \cap Q_k^+ = \varnothing$. Эту вершину x никогда не удалить из Q_k^-, так как из Q_k^- удаляются только вершины, смежные с Q_k^+. Таким образом, существование x, такого, что $x \in Q_k^-$ и $\Gamma(x) \cap Q_k^+ = \varnothing$, является достаточным условием для возвращения. Кроме того, $k \leqslant p - 1$.

10.5.4. Алгоритм построения максимальных независимых множеств вершин

Приведённый ниже алгоритм, обоснование которого дано в предыдущем подразделе, строит все максимальные независимые множества вершин заданного графа.

Алгоритм 10.3 Построение максимальных независимых множеств

Вход: граф $G(V, E)$, заданный списками смежности $\Gamma[v]$.
Выход: последовательность максимальных независимых множеств.

$k := 0$ { количество элементов в текущем независимом множестве }
$S[k] := \varnothing$ { $S[k]$ — независимое множество из k вершин }
$Q^-[k] := \varnothing$ { $Q^-[k]$ — множество вершин, уже использованных для расширения $S[k]$ }
$Q^+[k] := V$
{ $Q^+[k]$ — множество вершин, которые можно использовать для расширения $S[k]$ }
$M1$: { шаг вперед }
select $v \in Q^+[k]$ { расширяющая вершина }
$S[k+1] := S[k] \cup \{v\}$ { расширенное множество }
$Q^-[k+1] := Q^-[k] \setminus \Gamma[v]$ { вершина v использована для расширения }
$Q^+[k+1] := Q^+[k] \setminus (\Gamma[v] \cup \{v\})$
{ все вершины, смежные с v, не могут быть использованы для расширения }
$k := k + 1$
$M2$: { проверка }
for $u \in Q^-[k]$ **do**
  if $\Gamma[u] \cap Q^+[k] = \varnothing$ **then**
    goto $M3$ { можно возвращаться }
  end if
end for
if $Q^+[k] = \varnothing$ **then**
  if $Q^-[k] = \varnothing$ **then**
    yield $S[k]$ { множество $S[k]$ максимально }
  end if
  goto $M3$ { можно возвращаться }
else
  goto $M1$ { можно идти вперед }
end if
$M3$: { шаг назад }
$v := \mathrm{last}(S[k])$ { последний добавленный элемент }
$k := k - 1$
$S[k] := S[k+1] - \{v\}$
$Q^-[k] := Q^-[k] \cup \{v\}$ { вершина v уже добавлялась }
$Q^+[k] := Q^+[k] \setminus \{v\}$
if $k = 0$ & $Q^+[k] = \varnothing$ **then**
  stop { перебор завершён }
else
  goto $M2$ { переход на проверку }
end if

Пример Известная *задача о восьми ферзях* (расставить на шахматной доске 8 ферзей так, чтобы они не били друг друга) является задачей об отыскании максимальных независимых множеств. Действительно, достаточно представить доску в виде графа с 64 вершинами (соответствующими клеткам доски), которые смежны, если клетки находятся на одной вертикали, горизонтали или диагонали.

10.6. Доминирующие множества

Задача о наименьшем покрытии (сокращённо ЗНП) является примером общей экстремальной задачи, к которой прямо или косвенно сводятся многие практические задачи. Эта задача является классической, хорошо изучена и часто используется в качестве теста для сравнения и оценки различных общих методов решения трудоёмких задач.

В этом разделе на основе рассмотрения понятия доминирующего множества ЗНП формулируются и приводятся сведения о связи ЗНП с другими задачами.

10.6.1. Минимальное и наименьшее доминирующее множество

Множество вершин $S \subset V$ графа $G(V, E)$ называется *доминирующим множеством* (или *внешне устойчивым*), если $S \cup \Gamma(S) = V$, то есть

$$\forall v \in V \ \ (v \in S \vee \exists s \in S \ \ ((s, v) \in E)).$$

Очевидно, что множество S доминирует тогда и только тогда, когда

$$\forall v \notin S \ \ (\Gamma(v) \cap S \neq \varnothing),$$

что равносильно утверждению $\forall v \in V \ \ (\exists s \in S \ \ (d(v, s) \leqslant 1))$.

Доминирующее множество называется *минимальным*, если никакое его подмножество не является доминирующим. Доминирующее множество называется *наименьшим*, если число элементов в нём наименьшее возможное.

Пример Известная *задача о пяти ферзях* (расставить на шахматной доске 5 ферзей так, чтобы они били всю доску) является задачей об отыскании наименьших доминирующих множеств.

10.6.2. Доминирование и независимость

Доминирование тесно связано с вершинной независимостью.

ТЕОРЕМА *Независимое множество вершин является максимальным тогда и только тогда, когда оно является доминирующим.*

Доказательство

[$\Longrightarrow$] Пусть множество вершин $S \subset V$ — максимальное независимое. Допустим (от противного), что оно не доминирующее. Тогда существует вершина v, находящаяся на расстоянии больше 1 от всех вершин множества S. Эту вершину можно добавить к S с сохранением независимости, что противоречит максимальности.

[$\Leftarrow$] Пусть S — независимое доминирующее множество. Допустим (от противного), что оно не максимальное. Тогда существует вершина v, не смежная ни с одной из вершин множества S, то есть находящаяся на расстоянии больше 1 от всех вершин множества S. Это противоречит тому, что множество S — доминирующее. □

Независимое доминирующее множество вершин называется *ядром* графа.

Пример В полном графе K_p каждая из p вершин является ядром и других ядер нет.

СЛЕДСТВИЕ *Любой граф имеет ядро.*

Доказательство Следующий простой алгоритм, основанный на доказательстве предыдущей теоремы, строит некоторое ядро S.

$S := \varnothing$ { ядро }
while $V \neq \varnothing$ **do**
  select $v \in V$ { любая нерассмотренная вершина }
  $S := S + v$ { расширяем множество S }
  $V := V \setminus \Gamma^*(v)$ { удаляем вершины, которые не могут быть использованы для расширения }
end while □

Понятия независимости, доминирования и ядра применимы как к графам, так и к орграфам. Множество узлов S орграфа называется *независимым*, если $\forall v \in S\ (\Gamma(v) \cap S = \varnothing)$, и называется *доминирующим*, если $\forall v \notin S\ (\Gamma(v) \cap S \neq \varnothing)$. Независимое доминирующее множество узлов в орграфе называется *ядром*.

ЗАМЕЧАНИЕ

Существуют орграфы, не имеющее ядра. Например, контур C_3 не имеет ядра. Более того, задача выделения ядра в произвольном орграфе оказывается *NP*-полной.

10.6.3. Задача о наименьшем покрытии

Рассмотрим следующую задачу. Пусть каждой вершине сопоставлена некоторая цена. Требуется выбрать доминирующее множество с наименьшей суммарной ценой. Эта задача называется *задачей о наименьшем покрытии* (сокращённо ЗНП).

ЗНП является весьма общей задачей, к которой сводятся многие другие задачи.

Примеры

1. **Задача о выборе переводчиков.** Организации нужно нанять переводчиков для перевода с определённого множества языков. Каждый из имеющихся переводчиков владеет некоторыми иностранными языками и требует определённую зарплату. Требуется определить, каких переводчиков следует нанять, чтобы сумма расходов на зарплату была минимальной. Задача о выборе переводчиков сводится к ЗНП следующим образом. Рассмотрим полный двудольный граф с долями V_1 и V_2, где доля V_1 соответствует множеству переводчиков, а доля V_2 — множеству языков. Вершины $v_1 \in V_1$ и $v_2 \in V_2$ смежны, если переводчик v_1 владеет языком v_2. Требуется найти наименьшее покрывающее множество $S \subset V_1$.
2. **Задача о развозке.** Поставщику нужно доставить товары своим потребителям. Имеется множество возможных маршрутов, каждый из которых позволяет обслужить определённое подмножество потребителей и требует определённых расходов. Требуется определить, какие маршруты следует использовать, чтобы все потребители были обслужены, а сумма транспортных расходов была минимальной. Задача о развозке сводится к ЗНП аналогичным образом. Здесь V_1 — множество маршрутов, V_2 — множество потребителей и вершины $v_1 \in V_1$ и $v_2 \in V_2$ смежны, если маршрут v_1 обслуживает потребителя v_2.

10.6.4. Связь задачи о наименьшем покрытии с другими задачами

ЗНП может быть сформулирована многими разными способами.

Пример Пусть имеется конечное множество $V = \{v_1, \ldots, v_p\}$ и семейство подмножеств этого множества $E = \{E_1, \ldots, E_p\}$. Каждому подмножеству E_i приписан вес. Найти покрытие E' ($E' \subset E$) наименьшего веса. На языке графов v_i — это вершина, а E_i — множество смежности вершины, то есть $E_i := \Gamma^*(v_i)$.

ЗАМЕЧАНИЕ

Из этой формулировки происходит название «задача о наименьшем покрытии».

Известно, что ЗНП относится к числу трудоёмких задач, и для её решения применяются переборные алгоритмы с теми или иными улучшениями.

На рис. 10.7 приведена схема (заимствованная из [21]), показывающая связь ЗНП и некоторых других задач. На этой схеме стрелка от задачи A к задаче B означает, что решение задачи A влечет за собой решение задачи B.

10.7. Раскраска графов

Задача раскрашивания графов, которая на первый взгляд кажется просто праздной головоломкой, имеет неожиданно широкое применение в программировании, особенно при решении фундаментальных теоретических проблем (см., например, [23]).

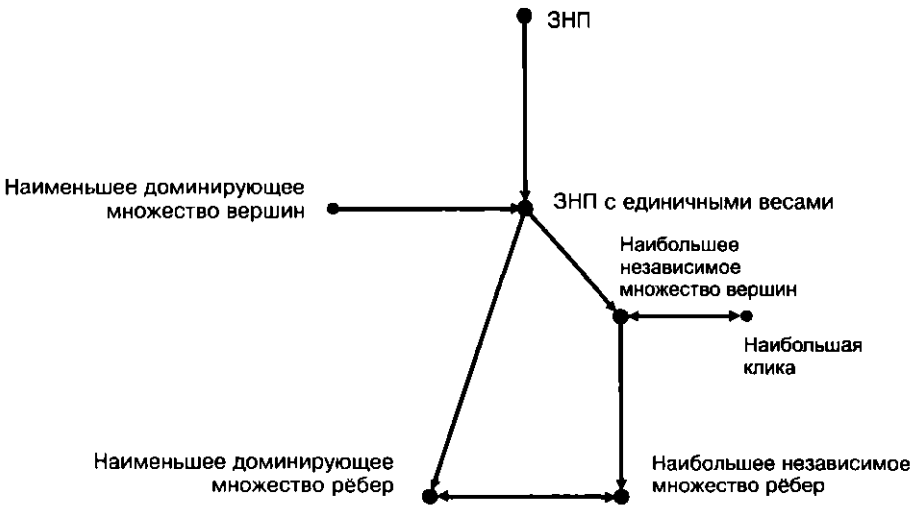

Рис. 10.7. Связь различных задач

10.7.1. Оценки хроматического числа

Раскраской графа G называется такое приписывание цветов (натуральных чисел) его вершинам, что никакие две смежные вершины не получают одинаковый цвет. Если задана допустимая раскраска графа, использующая m цветов, то говорят, что граф m-раскрашиваемый. Наименьшее возможное количество цветов в раскраске называется *хроматическим числом* и обозначается $\chi(G)$.

Примеры

$\chi(\overline{K}_p) = 1$, $\chi(K_p) = p$, $\chi(K_{m,n}) = 2$, $\chi(C_{2n}) = 2$, $\chi(C_{2n+1}) = 3$, $\chi(T) = 2$, где T — свободное дерево.

Очевидно, что существует m-раскраска графа G для любого m в диапазоне $\chi(G) \leqslant m \leqslant p$. Множество вершин, покрашенных в один цвет, называется *одноцветным классом*. Одноцветные классы образуют независимые множества вершин, то есть никакие две вершины в одноцветном классе не смежны.

Способ явного выражения хроматического числа через другие стандартные инварианты графа неизвестен. Известны только некоторые оценки, часть из которых приведена ниже.

ТЕОРЕМА 1 $\chi(G) \leqslant 1 + \Delta(G)$.

Доказательство Индукция по p. База: $p = 1 \Longrightarrow \chi(G) = 1$ & $\Delta(G) = 0$. Пусть

$$\forall G \ (p(G) < p \Longrightarrow \chi(G) \leqslant \Delta(G) + 1).$$

Рассмотрим граф G, такой, что $p(G) = p$. Тогда

$$\forall v \in V \ (\chi(G - v) \leqslant \Delta(G - v) + 1 \leqslant \Delta(G) + 1).$$

Но $d(v) \leqslant \Delta(G)$, значит, хотя бы один цвет в $(\Delta(G)+1)$-раскраске графа $G-v$ свободен для v. Покрасив вершину v в этот цвет, получаем $(\Delta(G)+1)$-раскраску графа G. □

ТЕОРЕМА 2 $p/\beta_0(G) \leqslant \chi(G) \leqslant p - \beta_0(G) + 1$.

Доказательство

[$p/\beta_0(G) \leqslant \chi(G)$] Пусть $\chi(G) = n$ и $V = V_1 \cup \cdots \cup V_n$, где V_i — одноцветные классы. V_i — независимое множество вершин, следовательно, $|V_i| \leqslant \beta_0(G)$. Имеем:

$$p = \sum_{i=1}^{n} |V_i| \leqslant n\beta_0(G) \Longrightarrow p/\beta_0 \leqslant \chi.$$

[$\chi(G) \leqslant p - \beta_0(G) + 1$] Пусть $S \subset V$ — наибольшее независимое множество, $|S| = \beta_0(G)$. Тогда $\chi(G-S) \leqslant |V-S| = p - \beta_0(G)$. Из n-раскраски графа $G-S$ можно получить $(n+1)$-раскраску графа G, так как все вершины из S можно покрасить в один новый цвет. Следовательно, $\chi(G) \leqslant \chi(G-S) + 1 \leqslant p - \beta_0 + 1$. □

10.7.2. Хроматические числа графа и его дополнения

Хроматическое число графа G и его дополнения $\overline{G}$ связаны: если в G сравнительно мало рёбер, то и $\chi(G)$ будет невелико, но тогда в $\overline{G}$ — много рёбер и $\chi(\overline{G})$ будет близко к p.

ТЕОРЕМА *Пусть* $\chi := \chi(G)$, $\overline{\chi} := \chi(\overline{G})$. *Тогда*

$$2\sqrt{p} \leqslant \chi + \overline{\chi} \leqslant p+1, \quad p \leqslant \chi\overline{\chi} \leqslant \left(\frac{p+1}{2}\right)^2.$$

Доказательство

[$p \leqslant \chi\overline{\chi}$] Пусть $\chi(G) = n$, $V_1, \ldots, V_n$ — одноцветные классы, $p_i := |V_i|$. Тогда $\sum\limits_{i=1}^{n} p_i = p$, следовательно, $\max\limits_{i=1}^{n} p_i \geqslant p/n$. Но V_i — независимые множества в G, следовательно, V_i — клики в $\overline{G}$. Значит, $\overline{\chi} \geqslant \max\limits_{i=1}^{n} p_i \geqslant p/n$. Имеем $\chi\overline{\chi} \geqslant n \cdot p/n = p$.

[$2\sqrt{p} \leqslant \chi + \overline{\chi}$] Известно, что среднее геометрическое двух чисел не превосходит среднего арифметического: $(a+b)/2 \geqslant \sqrt{ab}$. Следовательно, $\chi + \overline{\chi} \geqslant 2\sqrt{\chi\overline{\chi}} \geqslant 2\sqrt{p}$.

[$\chi + \overline{\chi} \leqslant p+1$] Индукция по p. База: $p = 1 \Longrightarrow \chi = 1$ & $\overline{\chi} = 1$. Пусть $\chi + \overline{\chi} \leqslant p$ для всех графов с $p-1$ вершинами. Рассмотрим граф G с p вершинами и вершину $v \in V$. Тогда, очевидно, $\chi(G) \leqslant \chi(G-v)+1$ и $\chi(\overline{G}) \leqslant \chi(\overline{G}-v)+1$. Если $\chi(G) < \chi(G-v)+1$ или $\chi(\overline{G}) < \chi(\overline{G}-v)+1$, то

$$\chi + \overline{\chi} = \chi(G) + \chi(\overline{G}) < \chi(G-v) + 1 + \chi(\overline{G}-v) + 1 \leqslant p+2.$$

Следовательно, $\chi + \overline{\chi} \leqslant p+1$. Пусть теперь $\chi(G) = \chi(G-v)+1$ и $\chi(\overline{G}) = \chi(\overline{G}-v)+1$. Положим $d := d(v)$ в графе G, тогда $\overline{d} = p - d - 1$ — степень вершины v в графе $\overline{G}$. Имеем $d \geqslant \chi(G-v)$. Действительно, $\chi(G) = \chi(G-v)+1$, и если бы $d < \chi(G-v)$,

то вершину v можно было бы покрасить в любой из свободных $\chi(G-v)-d$ цветов и получить $\chi(G-v)$-раскраску графа G. Аналогично, $\overline{d}=p-d-1\geqslant\chi(\overline{G}-v)$. Таким образом,

$$\chi+\overline{\chi}=\chi(G)+\chi(\overline{G})=\chi(G-v)+1+\chi(\overline{G}-v)+1\leqslant d+1+p-d-1+1=p+1.$$

[$\chi\overline{\chi}\leqslant\left(\frac{p+1}{2}\right)^2$] Имеем $2\sqrt{\chi\overline{\chi}}\leqslant\chi+\overline{\chi}\leqslant p+1$. Следовательно, $\left(\frac{p+1}{2}\right)^2\geqslant\chi\overline{\chi}$. □

10.7.3. Точный алгоритм раскрашивания

Поскольку формула для хроматического числа неизвестна, задача нахождения наилучшей раскраски графа оказывается, как и следовало ожидать, труднорешаемой.

Рассмотрим следующую схему рекурсивной процедуры P:

1. Выбрать в графе G некоторое максимальное независимое множество вершин S.
2. Покрасить вершины множества S в очередной цвет.
3. Применить процедуру P к графу $G-S$.

На псевдокоде процедура P может быть записана следующим образом.

Вход: граф $G(V,E)$, номер свободного цвета i.
Выход: раскраска, заданная массивом $C[V]$, — номера цветов, приписанные вершинам.

```
if V = ∅ then
  return { раскраска закончена }
end if
S := Selectmax(G) {S — максимальное независимое множество }
C[S] := i { раскрашиваем вершины множества S в цвет i }
P(G − S, i + 1) { рекурсивный вызов }
```

ЗАМЕЧАНИЕ

Функция Selectmax может быть реализована, например, алгоритмом 10.3.

ТЕОРЕМА *Если граф G — k-раскрашиваемый, то существует такая последовательность выборов множества S на шаге 1 процедуры P, что применение процедуры P к графу G построит не более чем k-раскраску графа G.*

Доказательство Пусть имеется некоторая k-раскраска графа $G(V,E)$. Перестроим её в такую не более чем k-раскраску, которая может быть получена процедурой P. Пусть $V_1\subset V$ — множество вершин в данной k-раскраске, покрашенных в цвет 1. Множество V_1 — независимое, но, может быть, не максимальное. Рассмотрим множество V_1', такое, что $V_1\cup V_1'$ — максимальное независимое множество (может оказаться, что $V_1'=\varnothing$). Вершины из V_1' не смежны с V_1, значит, вершины из V_1' можно перекрасить в цвет 1. Пусть далее $V_2\subset V\backslash(V_1\cup V_1')$ — множество вершин, покрашенных в цвет 2. Аналогично рассмотрим множество V_2', такое, что $V_2\cup V_2'$ — максимальное независимое в $G\backslash(V_1\cup V_1')$, покрасим

вершины $V_2 \cup V_2'$ в цвет 2 и т. д. Всего в исходной раскраске было k независимых множеств. При перекраске их число не возрастёт (но может уменьшиться, если $\chi(G) < k$). На каждом шаге алгоритма рассматривается одно из множеств, следовательно, процесс закончится. □

10.7.4. Приближённый алгоритм последовательного раскрашивания

В этом подразделе на примере алгоритмов раскрашивания графов вводится понятие приближённого алгоритма в более широком смысле по сравнению с тем, который обычно подразумевается в вычислительной математике и при проведении численных расчётов на компьютере.

В предыдущем подразделе некоторый алгоритм точного раскрашивания был построен на основе алгоритма выделения максимальных независимых множеств вершин, который имеет переборный характер. Таким образом, предложенный алгоритм точного раскрашивания также имеет переборный характер. Можно показать, что это не случайно и задача построения точной раскраски является NP-полной. При практическом решении NP-полных задач целесообразно рассматривать *приближённые* алгоритмы, которые не всегда находят точное решение задачи (иногда они находят только приближение к нему, и мы не можем знать этого заранее), но зато достаточно эффективны. Рассмотрим следующий *алгоритм последовательного раскрашивания*.

Алгоритм 10.4 Алгоритм последовательного раскрашивания

Вход: граф G.
Выход: раскраска графа — массив C : **array** $[1..p]$ **of** $1..p$.

```
for v ∈ V do
  C[v] := 0 { все вершины не раскрашены }
end for
for v ∈ V do
  A := {1, ..., p} { все цвета }
  for u ∈ Γ+(v) do
    A := A \ {C[u]} { занятые для вершины v цвета }
  end for
  C[v] := min A { минимальный свободный цвет }
end for
```

ЗАМЕЧАНИЕ

Таким образом, красить вершины необходимо последовательно, выбирая среди допустимых цветов минимальный.

Обоснование В основном цикле рассматриваются все вершины, и каждая из них получает допустимую раскраску. Таким образом, процедура строит допустимую раскраску. □

10.7.5. Улучшенный алгоритм последовательного раскрашивания

Следующий алгоритм также строит допустимую раскраску, применяя такую эвристику: начинать раскрашивать следует с вершин наибольшей степени, поскольку если их раскрашивать в конце процесса, то более вероятно, что для них не найдётся свободного цвета и придётся использовать еще один цвет.

Алгоритм 10.5 Улучшенный алгоритм последовательного раскрашивания

Вход: граф G.
Выход: раскраска графа — массив C : **array** $[1..p]$ **of** $1..p$.

```
Sort(V) { упорядочить вершины по невозрастанию степени }
c := 1 { первый цвет }
for v ∈ V do
  C[v] := 0 { все не раскрашены }
end for
while V ≠ ∅ do
  for v ∈ V do
    for u ∈ Γ+(v) do
      if C[u] = c then
        next for v { вершину v нельзя покрасить в цвет c }
      end if
    end for
    C[v] := c { красим вершину v в цвет c }
    V := V \ {v} { и удаляем её из рассмотрения }
  end for
  c := c + 1 { следующий цвет }
end while
```

Обоснование Заметим, что данный алгоритм отличается от предыдущего тем, что основной цикл идет не по вершинам, а по цветам: сначала всё, что можно, красим в цвет 1, затем в оставшемся красим всё, что можно, в цвет 2 и т. д. В остальном алгоритмы аналогичны, и данный алгоритм заканчивает свою работу построением допустимой раскраски по тем же причинам, что и предыдущий. □

ОТСТУПЛЕНИЕ

Алгоритм 10.5 несколько сложнее алгоритма 10.4 и основан на остроумной эвристике. Можно было бы ожидать, что он даст существенно лучшие результаты. Однако прямые вычислительные эксперименты показывают, что алгоритм 10.4 работает почти так же хорошо, как алгоритм 10.5. Таким образом, программистские «хитрости» далеко не всегда дают практически значимые результаты.

10.8. Планарность

Обсуждение планарности в этом разделе позволяет решить вторую историческую задачу из перечисленных в подразделе 7.1.1, а также подготавливает результаты, необходимые для доказательства теоремы о пяти красках.

10.8.1. Укладка графов

Граф *укладывается* на некоторой поверхности, если его можно нарисовать на этой поверхности так, чтобы рёбра графа при этом не пересекались. Граф называется *планарным*, если его можно уложить на плоскости. *Плоский* граф — это граф, уже уложенный на плоскости.

Область, ограниченная ребрами в плоском графе, называется *гранью*. Грань не содержит других граней. Число граней плоского графа G обозначается $f(G)$.

ЗАМЕЧАНИЕ

Внешняя часть плоскости также образует грань.

Пример На рис. 10.8 показаны диаграммы плапарного графа K_4 и его укладка на плоскости. Этот граф имеет 4 грани.

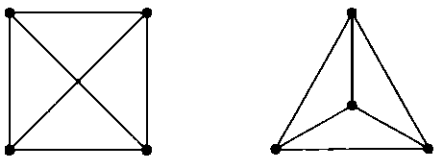

Рис. 10.8. Планарный граф и его укладка

ОТСТУПЛЕНИЕ

Точное определение некоторых понятий, используемых в этом разделе, в частности, таких понятий, как «поверхность», «область», «граница», выходит далеко за рамки этого учебника и стандартного курса дискретной математики. Мы полагаемся на геометрическую интуицию читателя и не даём никаких определений. Для понимания простейших рассуждений этого раздела достаточно неформальных интуитивных представлений. Однако читателю следует иметь в виду, что при решении сложных практических задач, например, из вычислительной геометрии, интуитивных представлений может оказаться недостаточно.

10.8.2. Эйлерова характеристика

Для графов, уложенных на некоторой поверхности, справедливо определённое соотношение между числом вершин, рёбер и граней графов, которые укладываются на этой поверхности.

ТЕОРЕМА (формула Эйлера) *Для связного планарного графа справедливо следующее соотношение:*

$$p - q + f = 2.$$

ЗАМЕЧАНИЕ

Число в правой части этого соотношения называется *эйлеровой характеристикой* поверхности.

Доказательство Индукция по q. База: $q = 0 \Longrightarrow p = 1$ & $f = 1$. Пусть теорема верна для всех графов с q рёбрами: $p - q + f = 2$. Добавим еще одно ребро. Если добавляемое ребро соединяет существующие вершины, то легко видеть, что $q' = q+1$, $p' = p$, $f' = f+1$ и $p' - q' + f' = p - q - 1 + f + 1 = p - q + f = 2$. Если добавляемое ребро соединяет существующую вершину с новой, то $p' = p + 1$, $q' = q + 1$, $f' = f$ и $p' - q' + f' = p + 1 - q - 1 + r = p - q + f = 2$. □

СЛЕДСТВИЕ 1 *Если G — связный планарный граф ($p > 3$), то $q \leqslant 3p - 6$.*

Доказательство Каждая грань ограничена по крайней мере тремя ребрами, каждое ребро ограничивает не более двух граней, отсюда $3f \leqslant 2q$. Имеем

$$2 = p - q + f \leqslant p - q + \frac{2q}{3} \Longrightarrow 3p - 3q + 2q \geqslant 6 \Longrightarrow q \leqslant 3p - 6. \qquad \square$$

СЛЕДСТВИЕ 2 *Графы K_5 и $K_{3,3}$ не планарны.*

Доказательство

[K_5 не планарен] Имеем $p = 5$, $q = 10$. Если K_5 планарен, то по предыдущему следствию $q = p(p-1)/2 = 10 \leqslant 3p - 6 = 3 \cdot 5 - 6 = 9$ — противоречие.

[$K_{3,3}$ не планарен] Имеем $p = 6$, $q = 9$. В этом графе нет треугольников, значит, если этот граф планарен, то в его плоской укладке каждая грань ограничена не менее чем четырьмя ребрами и, следовательно, $4f \leqslant 2q$ или $2f \leqslant q$. По формуле Эйлера $6-9+f=2$, откуда $f=5$. Имеем $2f=2\cdot 5=10\leqslant q=9$ — противоречие. □

ЗАМЕЧАНИЕ

Операция подразбиения ребра $x = (u, v)$ состоит в замене его двумя рёбрами (u, w) и (w, v), где w — новая вершина. Два графа называются *гомеоморфными*, если они могут быть получены из одного графа подразбиением рёбер. Граф планарен тогда и только тогда, когда он не содержит подграфов, гомеоморфных K_5 или $K_{3,3}$. Доказательство достаточности этого утверждения (*теоремы Куратовского*) выходит за рамки данного курса.

СЛЕДСТВИЕ 3 *В любом планарном графе существует вершина, степень которой не больше 5.*

Доказательство От противного. Пусть $\forall v \in V \ (d(v) \geqslant 6)$. Тогда

$$6p \leqslant \sum_{v \in V} d(v) = 2q \Longrightarrow 3p \leqslant q,$$

но $q \leqslant 3p - 6$. Имеем: $3p \leqslant 3p - 6$ — противоречие. □

ОТСТУПЛЕНИЕ

Для случая многогранников формулу Эйлера вывел Декарт. Действительно, каркас многогранника — это плоский граф, и обратно, связному плоскому графу соответствует многогранник.

10.8.3. Теорема о пяти красках

ТЕОРЕМА *Всякий планарный граф можно раскрасить пятью красками.*

Доказательство Достаточно рассматривать связные графы, потому что

$$\chi\left(\bigcup_{i=1}^{n} G_i\right) = \max_{i=1}^{n} \chi(G_i).$$

Индукция по p. База: если $p \leqslant 5$, то $\chi \leqslant p \leqslant 5$. Пусть теорема верна для всех связных планарных графов с p вершинами. Рассмотрим граф G с $p+1$ вершинами. По третьему следствию к формуле Эйлера $\exists v \in V \ (d(v) \leqslant 5)$. По индукционному предположению $\chi(G - v) \leqslant 5$. Нужно раскрасить вершину v. Если $d(v) < 5$, то в 5-раскраске графа $G - v$ существует цвет, свободный для вершины v. Если $d(v) = 5$ и для $\Gamma^+(v)$ использованы не все пять цветов, то в 5-раскраске графа $G - v$ существует цвет, свободный для вершины v. Остался случай, когда $d(v) = 5$ и все пять цветов использованы (вершина v_i покрашена в цвет i, рис. 10.9). Пусть G_{13} — правильный подграф графа $G - v$, порожденный всеми вершинами, покрашенными в цвета 1 или 3 в 5-раскраске графа $G - v$. Если v_1 и v_3 принадлежат разным компонентам связности графа G_{13}, то в той компоненте, в которой находится вершина v_1, произведем перекраску $1 \leftrightarrow 3$. При этом получится 5-раскраска графа $G - v$, но цвет 1 будет свободен для вершины v. В противном случае существует простая цепь, соединяющая v_1 и v_3 и состоящая из вершин, покрашенных в цвета 1 и 3. Тогда вершины v_2 и v_4 принадлежат разным компонентам связности подграфа G_{24} (так как граф G — планарный). Перекрасим вершины $2 \leftrightarrow 4$ в той компоненте связности графа G_{24}, которой принадлежит v_2, и получим 5-раскраску графа $G - v$, в которой цвет 2 свободен для вершины v. □

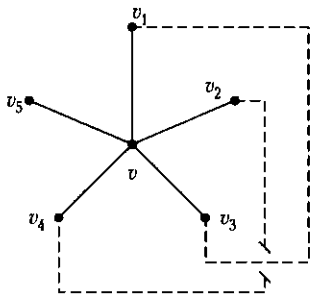

Рис. 10.9. К доказательству теоремы о пяти красках

Комментарии

Из вопросов, рассмотренных в этой главе, в литературе наибольшее внимание уделено задаче коммивояжёра. Анализ различных подходов к её решению см., например, в [21]. Алгоритм 10.1 описан в [8], в других источниках (например, в [5]) можно найти другие варианты решения этой задачи.

Обсуждение переборных задач в этой главе носит в основном ознакомительный характер. Для получения более точной и детальной информации следует обратиться к специальной литературе, прежде всего к фундаментальной книге [19]. Алгоритм построения максимальных независимых множеств заимствован из [21]. Здесь этот алгоритм использован в качестве примера для иллюстрации способов и особенностей решения переборных задач.

Центральный результат этой главы — теорема о пяти красках — изложен по книге [28]. Алгоритмы раскрашивания изложены по книге [21], в которой можно найти их более детальное и подробное обсуждение. Различные применения задачи о раскраске графов в программировании и смежные вопросы освещены в [20].

Упражнения

10.1. Доказать, что кодерево связного графа является максимальным подграфом, не содержащим коциклов.

10.2. Доказать, что эйлеров граф не имеет мостов.

10.3. Доказать, что если в связном графе $G(V, E)$ $\forall u \neq v \in V$ $(d(u) + d(v) \geqslant p)$, то граф G гамильтонов.

10.4. Доказать, что $\alpha_0 \geqslant \beta_1, \alpha_1 \geqslant \beta_0$.

10.5. Написать алгоритм построения всех клик графа.

10.6. Доказать или опровергнуть следующее утверждение: наименьшее доминирующее множество является ядром.

10.7. Доказать, что $\chi(G(V, E)) \leqslant 1 + \max\limits_{V' \subset V} \delta(G'(V', E'))$.

10.8. Доказать, что если в планарном графе каждая грань есть C_n, то

$$q = \frac{n(p-2)}{n-2}.$$

Указатель основных обозначений

Метаобозначения

Обозначение	Смысл	Пример
$\stackrel{\text{Def}}{=}$	По определению есть	$\lvert\varnothing\rvert \stackrel{\text{Def}}{=} 0$
$:=$	Положим равным	$c:=(a+b)/2$
$=$	Равно	$1=1, 2\times 2=4$

Числовые множества

Обозначение	Название	Примеры
$\mathbb{N}$	Натуральные числа	$1, 2, 3$
$\mathbb{Z}$	Целые числа	$0, 1, -1, 2, -2$
$\mathbb{Q}$	Рациональные числа	$\frac{1}{2}, \frac{1}{3}, \frac{553}{113}, 2,718281828$
$\mathbb{R}$	Вещественные числа	$1, 1.5, \sqrt{2}, \pi, e$

Совокупности

Обозначение	Применение	Примеры
$\{a_1, \ldots, a_n\}$	Неупорядоченное множество различных элементов	$\{1, 2, 3\}$
$(a_1, \ldots, a_n)$	Упорядоченная последовательность однородных элементов	$(0, 1, 0, 1)$
$\langle A; B, C\rangle$	Упорядоченная последовательность разнородных элементов	$\langle\mathbb{R}; +, *\rangle$
$\langle a_1, \ldots, a_n\rangle$	Упорядоченная последовательность составных элементов	$\langle a \to 01, b \to 10\rangle$

Операции с множествами

Обозначение	Прочтение	Примеры
$a \in A$	Элемент a принадлежит множеству A	$1 \in \{1,2,3\}$
$a \notin A$	Элемент a не принадлежит множеству A	$4 \notin \{1,2,3\}, \sqrt{2} \notin \mathbb{Q}$
$A \subset B$	Множество A является подмножеством множества B	$\{2,3\} \subset \{1,2,3\}$
$A \cup B$	Объединение множеств A и B	$\{1,2\} \cup \{2,3\} = \{1,2,3\}$
$A \cap B$	Пересечение множеств A и B	$\{1,2\} \cap \{2,3\} = \{2\}$
$A \setminus B$	Разность множеств A и B	$\{1,2\} \setminus \{2,3\} = \{1\}$
$A \triangle B$	Симметрическая разность множеств A и B	$\{1,2\} \triangle \{2,3\} = \{1,3\}$
$A \times B$	Прямое произведение множеств A и B	$\{1,2\} \times \{2,3\} = = \{(1,2),(1,3),(2,2),(2,3)\}$
$\varnothing$	Пустое множество	$\{\ \}$
$\lvert A \rvert$	Мощность множества A	$\lvert\{1,2\}\rvert = 2$

Логические обозначения

Обозначение	Название	Прочтение
$\neg P$	Отрицание	Не P
$P \vee Q$	Дизъюнкция	P или Q
$P \,\&\, Q$	Конъюнкция	P и Q
$P \Longrightarrow Q$	Импликация	Если P, то Q
$\forall x\ (P(x))$	Квантор всеобщности	Для всех x выполнено $P(x)$
$\exists x\ (P(x))$	Квантор существования	Существует x, такой, что выполнено $P(x)$

Отношения

Обозначение	Название
$R \circ S$	Композиция отношений
R^n	Степень отношения
$\boxed{R}$	Матрица отношения
$\equiv$	Отношение эквивалентности
$\prec$	Отношение порядка
$<$	Отношение строгого линейного порядка
$\leqslant$	Отношение нестрогого линейного порядка

Функции

Обозначение	Прочтение	Примечание
$f\colon A \to B$	Функция f из A в B	$f \subset A \times B$
$b = f(a)$	b является значением функции f для аргумента a	$a \mapsto b$
$f \circ g$	Суперпозиция функций f и g	$(f \circ g)(x) = f(g(x))$
f^{-1}	Обратная функция к f	$f^{-1}\colon B \to A$
$\mathrm{Dom}\, f$	Множество определения функции $f\colon A \to B$	$\forall a \in \mathrm{Dom}\, f\ (\exists b \in B\ (b = f(a)))$
$\mathrm{Im}\, f$	Множество значений функции $f\colon A \to B$	$\forall b \in \mathrm{Im}\, f\ (\exists a \in A\ (b = f(a)))$

Групповые операции

Обозначение	Прочтение	Примечание
$\sum\limits_{i=1}^{n} a_i$	Сумма элементов $a_1, \ldots, a_n$	$\sum\limits_{i=1}^{n} a_i = a_1 + \ldots + a_n$
$\prod\limits_{i=1}^{n} a_i$	Произведение элементов $a_1, \ldots, a_n$	$\prod\limits_{i=1}^{n} a_i = a_1 \times \cdots \times a_n$
$\min(a_1, \ldots, a_n)$	Минимальный из элементов $a_1, \ldots, a_n$	$\min(a, b) \leqslant a$ & $\min(a, b) \leqslant b$
$\max(a_1, \ldots, a_n)$	Максимальный из элементов $a_1, \ldots, a_n$	$\max(a, b) \geqslant a$ & $\max(a, b) \geqslant b$

Список литературы

1. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. Мир, 1978.
2. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. Мир, 1979.
3. *Владимиров Д. А.* Булевы алгебры. Наука, 1969.
4. *Евстигнеев В. А.* Применение теории графов в программировании. Наука, 1985.
5. *Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И.* Лекции по теории графов. Наука, 1990.
6. *Зыков А. А.* Основы теории графов. Наука, 1987.
7. *Кук В., Бейз Г.* Компьютерная математика. Наука, 1990.
8. *Липский В.* Комбинаторика для программистов. Мир, 1988.
9. *Романовский И. В.* Дискретный анализ. Невский диалект, 1999.
10. *Яблонский С. В., Лупанов О. Б.* Дискретная математика и математические вопросы кибернетики. Наука, 1974.
11. *Яблонский С. В.* Введение в дискретную математику. Наука, 1986.
12. *Карпов Ю. Г.* Теория автоматов. Питер, 2002.
13. *Андерсон Д.* Дискретная математика и комбинаторика. Вильямс, 2003.
14. *Кнут Д.* Искусство программирования для ЭВМ. т. 1. основные алгоритмы. Мир, 1977.
15. *Кнут Д.* Искусство программирования для ЭВМ. т. 2. получисленные алгоритмы. Мир, 1977.
16. *Кнут Д.* Искусство программирования для ЭВМ. т. 3. сортировка и поиск. Мир, 1977.
17. *Нечаев В. И.* Элементы криптографии. основы теории защиты информации. Высшая школа, 1999.
18. *Берж К.* Теория графов и её применения. Изд. иностр. лит., 1962.
19. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. Мир, 1982.

20. *Касьянов В. Н., Евстигнеев В. А.* Графы в программировании: обработка, визуализация и применение. БХВ-Петербург, 2003.

21. *Кристофидес Н.* Теория графов. алгоритмический подход. Мир, 1978.

22. *Сачков В. Н.* Введение в комбинаторные методы дискретной математики. Наука, 1982.

23. *Ершов А. П.* Введение в теоретическое программирование. Наука, 1977.

24. *Кон П.* Универсальная алгебра. Мир, 1968.

25. *Уилсон Р.* Введение в теорию графов. Мир, 1977.

26. *Лавров С. С. Гончарова Л. И.* Автоматическая обработка данных, хранение информации в памяти ЭВМ. Наука, 1971.

27. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. МЦНМО, 1999.

28. *Харари Ф.* Теория графов. Мир, 1973.

29. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. Наука, 1983.

30. *Мендельсон Э.* Введение в математическую логику. Наука, 1984.

31. *Фрид Э.* Элементарное введение в абстрактную алгебру. Мир, 1979.

Предметный указатель

Б

В

E

З

И

К

Л

М

Н

О

П

P

C

T

Новиков Федор Александрович

Дискретная математика для программистов

Учебник для вузов

3-е издание

Заведующий редакцией	*А. Сандрыкин*
Научный редактор	*проф. Н. Ю. Нецветаев*
Руководитель проекта	*П. Маннинен*
Ведущий редактор	*Д. Петров*
Художник	*Л. Адуевская*
Корректоры	*Н. Рощина, Н. Солнцева*
Верстка	*А. Дмитриев*

Подписано в печать 03.07.08. Формат 70x100/16. Усл. п. л. 31,0. Доп. тираж 4000. Заказ 10253

ООО «Питер Пресс», 198206, Санкт-Петербург, Петергофское шоссе, д. 73, лит. А29.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2;
95 3005 — литература учебная.

Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., д. 15.