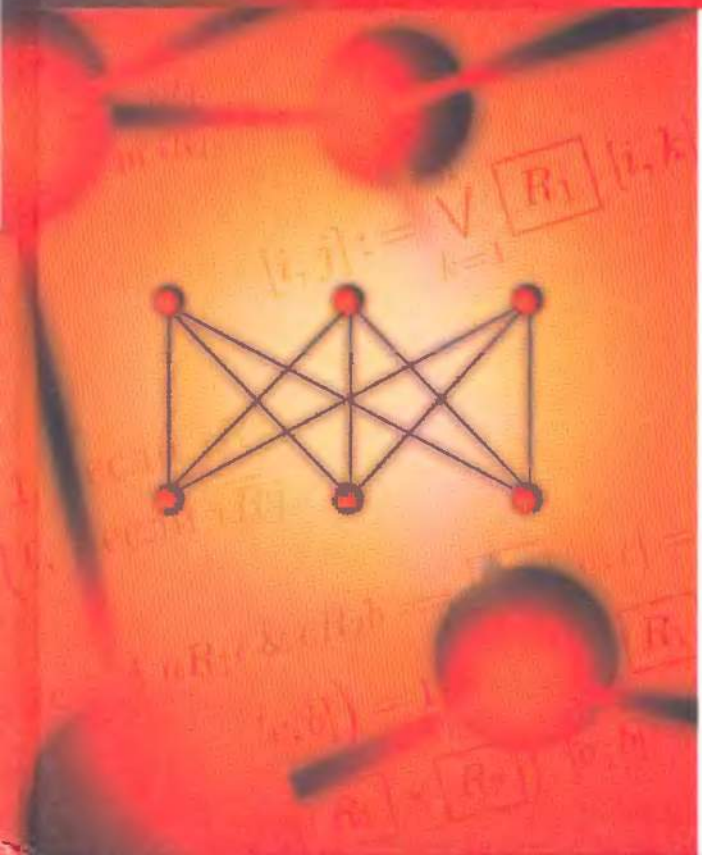
Ф. А. Новиков

ПИТЕР

ДИСКРЕТНАЯ МАТЕМАТИКА

ДЛЯ ПРОГРАММИСТОВ

УЧЕБНИК



- систематическое изложение основных разделов дискретной математики
- описание важнейших алгоритмов над объектами дискретной математики
- основные способы представления объектов дискретной математики с помощью стандартных структур данных

Рецензенты:

Кафедра «Прикладная математика»
Санкт-Петербургского государственного технического университета

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

Н73 Дискретная математика для программистов / Ф. А. Новиков — СПб: Питер, 2000. — 304 с.: ил.

ISBN 5-272-00183-4

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.

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

ББК 32.973-018я7+22.174
УДК 681.3.06:51(075)

ISBN 5-272-00183-4

Краткое содержание

Содержание

Вступительное слово

Автор этой книги Ф. А. Новиков имеет большой опыт практического программирования, чтения лекций по дискретной математике и написания книг, посвященных различным вопросам вычислительной техники и ее 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, причем затронута даже нетрадиционная проблема автоматического доказательства теорем, комбинаторику и кодирование. Особое внимание уделено общей теории графов — одному из важнейших инструментов программиста, и главным ее 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 моем 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 ее 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

Назначение и особенности книги

Данная книга представляет собой конспективный 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 нетривиальных 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 кончая исчерпывающими справочниками для специалистов. С другой стороны, большое число монографий, многие из которых стали классическими, посвящены вопросам теории и технологии программирования. 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 редкого исключения можно назвать книгу В. Липского «Комбинаторика для программистов» [14], перевод которой выдержал уже два издания в России. 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 уже в течение четырнадцати лет читает студентам кафедры «Прикладная математика» Санкт-Петербургского государственного технического университета, что наложило определенный отпечаток на состав материала.

Учебник охватывает почти все основные разделы дискретной математики: наивную теорию множеств, математическую логику, общую алгебру, комбинаторику и теорию графов. Из основных разделов отсутствуют теория алгоритмов и (машинная) арифметика, равно как и некоторые необходимые для программиста, но более специальные разделы, такие как теория формальных грамматик, вычислительная геометрия и теория конечных автоматов. Это объясняется тем, что данные разделы студенты изучают в специальных курсах. В то же время в учебник

включены такие специальные разделы, как теория булевых функций и теория кодирования, поскольку по этим темам специальных курсов не предусмотрено.

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

Фактически весь материал делится на две части, соответствующие двум семестрам курса. Первая часть, несколько большая по объему (главы 1–6), содержит самые общие сведения из различных разделов дискретной математики. Доля алгоритмов в первой части несколько меньше, а доля определений несколько больше, чем во второй части. Вторая часть (главы 7–12) целиком посвящена теории графов, поскольку связанные с ней вопросы (в частности, алгоритмы на графах) являются наиболее широко применяющимися в практическом программировании разделами дискретной математики. Тексты алгоритмов на графах, некоторые из которых весьма нетривиальны, составляют почти половину объема материала второй части.

Главы делятся на разделы, которые, в свою очередь, делятся на подразделы. Каждый раздел посвящен одному конкретному вопросу темы главы и по объему 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.

Главы книги более или менее независимы и могут изучаться в любом порядке, за исключением первых глав каждой части, которые содержат набор базисных определений для дальнейшего изложения.

В конце каждой главы имеются два специальных раздела: «Комментарии» и «Упражнения». В разделах первого типа 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. Упражнения немногочисленны — ровно по одному на каждый раздел — и очень просты. Как правило, в упражнения выносится дополнительный материал, непосредственно связанный с темой раздлаа (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, похожий по синтаксису на Паскаль (но, конечно, Паскалем не являющийся).

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

```
for x ∈ 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 два «естественных» оператора, аналоги которых в явном виде редко встречаются в настоящих языках программирования. Оператор

```
select m ∈ 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.

Поскольку данная 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, отметим широкое использование символа присваивания $:=$, который в математическом контексте можно читать как «по определению» или «положим». Если в левой части такого присваивания стоит простая переменная, то это имеет очевидный смысл введения обозначения. Но в левой части может стоять и *выражение*. В этом случае левую часть следует понимать как заголовок процедуры вычисления значения выражения, а правую часть — как тело этой процедуры. Например, формула

$$A = B := A \subset B \,\&\, B \subset A$$

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

Наряду с обычным для математики «статическим» использованием знака $:=$, когда выражению в левой части придается постоянное значение (определение константы), знак присваивания используется и в «динамическом», программистском смысле. Например, пусть в графе G есть вершина v. Тогда формулу $G := G \setminus \{v\}$ следует понимать так: «удалим в графе G вершину v».

Кроме того, в книге используются и другие приемы, хорошо известные программистам. Например, описание структуры формул задается с помощью формальной грамматики в общепринятых обозначениях.

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

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

может быть написано

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

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 объем текста, но требует внимания при чтении.

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 объема 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 алфавитный указатель, помещенный 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).

Формулировки теорем, лемм и следствий, в соответствии с общепринятой в математической литературе практикой, выделены курсивом. При этом формулировке предшествует соответствующее ключевое слово: «теорема», «лемма», «следствие». Как правило, утверждения не нумеруются, за исключением случаев вхождения нескольких однородных утверждений в один подраздел. Для ссылок на утверждения используются номера подразделов, в которых утверждения сформулированы.

Доказательства относятся к предшествующим утверждениям, а *обоснования* — к предшествующим алгоритмам. В очевидных случаях они опускаются или выносятся в упражнения. Доказательства и обоснования начинаются с соответствующего ключевого слова («доказательство» или «обоснование») и заканчиваются специальным значком □.

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 неспецифицированном языке программирования, синтаксис которого считается очевидным. Как правило, перед текстом алгоритма на естественном языке указывается 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.

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 понятия, которые должны быть известны читателю из курса математики средней школы, и понятия, уже рассмотренные в тексте книги.

Благодарности

Автор выражает благодарность своей жене Елене Владимировне Новиковой за постоянную поддержку в работе, коллеге Вадиму Эдуардовичу Халепскому за неоценимую помощь в оформлении рукописи и многочисленным студентам, прослушавшим этот курс, за выявление ошибок в тексте.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter-press.ru (издательство «Питер», компьютерная редакция). Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на Web-сайте издательства http://www.piter-press.ru.

ГЛАВА 1 Множества и отношения

В этой книге рассматриваются некоторые элементарные понятия «дискретной» или «конечной» математики, то есть математики, не связанной с понятиями бесконечности, предела и непрерывности. 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.

Понятия «множества», «отношения», «функции» и близкие к ним составляют основной словарь дискретной (равно как и классической «непрерывной») математики. Именно эти базовые понятия рассматриваются в первой главе, закладывая необходимую основу для дальнейших построений. Отличие состоит в том, что здесь рассматриваются почти исключительно *конечные* множества, а тонкие и сложные вопросы, связанные с рассмотрением бесконечных множеств, сознательно опущены. С другой стороны, значительное внимание уделяется «представлению» множеств в программах. Эти вопросы не имеют никакого отношения к собственно теории множеств в ее классическом виде, но очень важны для практикующего программиста.

1.1. Множества

Человеческое мышление устроено так, что мир представляется состоящим из отдельных «объектов». Философам давно ясно, что мир — единое неразрывное целое, и выделение в нем объектов — это не более чем произвольный акт нашего мышления, позволяющий сформировать доступную для рационального анализа картину мира. Но как бы там 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 неопределяемых понятий математики. Можно сказать, что *множество* — это любая определенная совокупность объектов. Объекты, из которых составлено множество, называются его *элементами*. Элементы множества различны и отличимы друг от друга.

Пример

Множество S страниц в данной книге. Множество $\mathbb{N}$ *натуральных* чисел 1, 2, 3, ... Множество $\mathbb{P}$ *простых* чисел 2, 3, 5, 7, 11, ... Множество $\mathbb{Z}$ целых чисел: ..., -2, -1, 0, 1, 2, ... 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.

ОТСТУПЛЕНИЕ

Понятия множества, элемента и принадлежности, которые на первый взгляд представляются интуитивно ясными, при ближайшем рассмотрении такую ясность утрачивают. Во-первых, проблематична отличимость элементов. Например, символы a и α, которые встречаются на этой странице, — это один элемент множества A или два разных элемента? 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, обычно называется *классом* или *семейством*.

ЗАМЕЧАНИЕ

Семейства множеств обычно обозначают прописными «рукописными» буквами латинского алфавита, чтобы отличить их от множеств, не содержащих множеств в качестве элементов.

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

Обычно в конкретных рассуждениях элементы всех множеств 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 можно сделать различными способами:

перечислением элементов: $M := \{a_1, a_2, \ldots, a_k\}$

характеристическим предикатом: $M := \{x \mid P(x)\}$;

порождающей процедурой: $M := \{x \mid x :=_f\}$.

ЗАМЕЧАНИЕ

При задании множеств перечислением обозначения элементов обычно заключают в фигурные скобки и разделяют запятыми. Характеристический предикат — это некоторое условие, выраженное в форме логического утверждения или процедуры, возвращающей логическое значение. Если для данного элемента условие выполнено, то он принадлежит определяемому множеству, в противном случае — не принадлежит. Порождающая процедура — это процедура, которая, будучи запущенной, порождает некоторые объекты, являющиеся элементами определяемого множества.

Пример

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

ЗАМЕЧАНИЕ

Множество целых чисел в диапазоне от m до n обозначают так: $m..n$. То есть

$$m..n := \{k \in \mathbb{Z} \mid 0 \leqslant k \,\&\, k \leqslant n\}.$$

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

Пример

$\mathbb{N} := \{n \mid n \quad 0; \textbf{while true } n \quad n + 1 \textbf{ do yield } n \textbf{ endwhile}\}$

1.1.3. Парадокс Рассела

Задание множеств характеристическим предикатом может приводить к противоречиям. Например, все рассмотренные в примерах множества не содержат себя в качестве элемента. Рассмотрим множество всех множеств, не содержащих себя в качестве элемента:

$$Y = \{X \mid X \notin X\}$$

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*. Вот три способа избежать этого 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 имеют тип 1, множества множеств — тип 2 и т. д. Y не имеет типа и множеством не является.
3. Характеристический предикат $P(x)$ задан в виде вычислимой функции (алгоритма). Способ вычисления значения предиката $X \in X$ не задан, а потому Y множеством не является.

ОТСТУПЛЕНИЕ

Последний из перечисленных способов лежит в основе 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 некоторые понятия и методы классической математики, чреватые возможными парадоксами.

1.2. Операции над множествами

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

1.2.1. Сравнение множеств

Множество A *содержится* в множестве B (множество B *включает* множество A), если каждый элемент A есть элемент B:
$$A \subset B := x \in A \Longrightarrow x \in B.$$
В этом случае A называется *подмножеством* B, B — *надмножеством* A. Если $A \subset B$ и $A \neq B$, то A называется *собственным* подмножеством B. Заметим, что $\forall M \; M \subset M$. По определению $\forall M \; \varnothing \subset M$.

ЗАМЕЧАНИЕ

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

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

$$A = B := A \subset B \& B \subset A.$$

Мощность множества M обозначается как $|M|$. Для конечных множеств мощность — это число элементов. Например, $|\varnothing| = 0$, но $|\{\varnothing\}| = 1$. Если $|A| = |B|$, то множества A и B называются *равномощными*.

1.2.2. Операции над множествами

Обычно рассматриваются следующие операции над множествами:

объединение:
$$A \cup B := \{x \mid x \in A \vee x \in B\};$$

пересечение:
$$A \cap B := \{x \mid x \in A \& x \in B\};$$

разность:
$$A \setminus B := \{x \mid x \in A \& x \notin B\};$$

симметрическая разность:
$$A \triangle B := (A \cup B) \setminus (A \cap B) = \{x \mid (x \in A \& x \notin B) \vee (x \notin A \& x \in B)\};$$

дополнение:
$$\overline{A} := \{x \mid x \notin A\}.$$

Операция дополнения подразумевает некоторый универсум U: $\overline{A} = U \setminus A$.

Пример

Пусть $A := \{1, 2, 3\}$, $B := \{3, 4, 5\}$. Тогда

$$A \cup B = \{1, 2, 3, 4, 5\}, \quad A \cap B = \{3\}, \quad A \setminus B = \{1, 2\}, \quad A \triangle B = \{1, 2, 4, 5\}$$

На рис. 1.1 приведены *диаграммы Эйлера*, иллюстрирующие операции над множествами. Сами исходные множества изображаются фигурами (в данном случае овалами), а результат графически выделяется (в данном случае для выделения использована штриховка).

Операции пересечения и объединения допускают следующее обобщение. Пусть I — некоторое множество, элементы которого используются как индексы, и пусть для любого $i \in I$ множество A_i известно. Тогда

$$\bigcup_{i \in I} A_i := \{x \mid \exists i \in I \; x \in A_i\}, \qquad \bigcap_{i \in I} A_i := \{x \mid \forall i \in I \; x \in A_i\}.$$

1.2.3. Разбиения и покрытия

Пусть $\mathcal{E} = \{E_i\}_{i \in I}$ — некоторое семейство подмножеств множества M, $E_i \subset M$

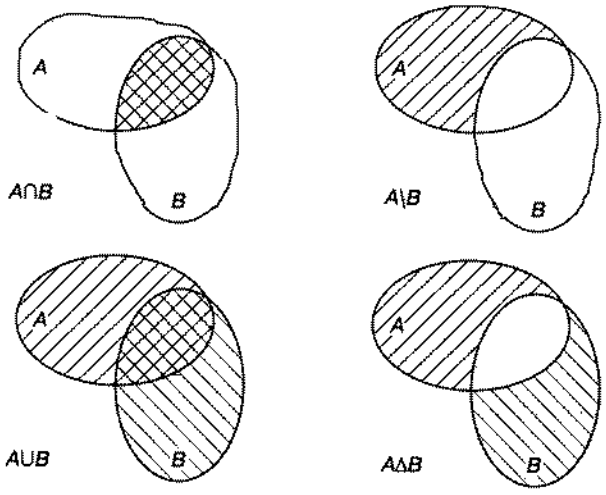


Рис. 1.1. Операции над множествами

Семейство $\mathcal{E}$ называется *покрытием* множества M, если каждый элемент M принадлежит хотя бы одному из E_i:

$$M \subset \bigcup_{i \in I} E_i \Longleftrightarrow \forall x \in M \; \exists i \in I \; x \in E_i.$$

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 $\mathcal{E}$ называется *разбиением* множества M.

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.

1.3. Алгебра подмножеств

Операции над множествами обладают целым рядом важных свойств, которые рассматриваются в этом разделе.

1.3.1. Булеан

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

$$2^M := \{A \mid A \subset M\}$$

ТЕОРЕМА *Для конечного множества* M $|2^M| = 2^{|M|}$

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

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

Имеем $2^M = \mathcal{M}_1 \cup \mathcal{M}_2$ и $\mathcal{M}_1 \cap \mathcal{M}_2 = \varnothing$. По индукционному предположению $|\mathcal{M}_1| = 2^{k-1}$, $|\mathcal{M}_2| = 2^{k-1}$. Следовательно, $|2^M| = |\mathcal{M}_1| + |\mathcal{M}_2| = 2^{k-1} + 2^{k-1} = 2 \cdot 2^{k-1} = 2^k = 2^{|M|}$. □

Пересечение, объединение и разность подмножеств множества U (универсума) являются подмножествами множества U. Множество всех подмножеств множества U с операциями пересечения, объединения, разности и дополнения образует *алгебру подмножеств* множества U.

1.3.2. Свойства операций над множествами

Пусть задан универсум U. Тогда $\forall A, B, C \subset U$ выполняются следующие свойства.

1. *идемпотентность*:
   $A \cup A = A$, $A \cap A = A$;
2. *коммутативность*:
   $A \cup B = B \cup A$, $A \cap B = B \cap A$;
3. *ассоциативность*:
   $A \cup (B \cup C) = (A \cup B) \cup C$, $A \cap (B \cap C) = (A \cap B) \cap C$;
4. *дистрибутивность*:
   $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
5. *поглощение*:
   $(A \cap B) \cup A = A$, $(A \cup B) \cap A = A$;
6. свойства нуля:
   $A \cup \varnothing = A$, $A \cap \varnothing = \varnothing$;
7. свойства единицы:
   $A \cup U = U$, $A \cap U = A$;
8. *инволютивность*:
   $\overline{\overline{A}} = A$;

9. *законы де Моргана*:
$\overline{A \cap B} = \overline{A} \cup \overline{B}$, $\overline{A \cup B} = \overline{A} \cap \overline{B}$.
10. свойства дополнения:
$A \cup \overline{A} = U$, $A \cap \overline{A} = \varnothing$;
11. выражение для разности:
$A \setminus B = A \cap \overline{B}$.

В справедливости перечисленных свойств можно убедиться различными способами. Например, нарисовать диаграммы Эйлера 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, обнаружили, 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, $A \cup A = A$. Аналогичные рассуждения нетрудно провести и для остальных равенств.

1.4. Представление множеств в ЭВМ

В этом параграфе рассматривается представление множеств в программах. Термин «представление» (еще употребляют термин «реализация») применительно к программированию означает следующее. Задать представление какого-либо объекта (в данном случае множества) — значит описать в терминах используемой системы программирования структуру данных, используемую для хранения информации о представляемом объекте, и алгоритмы над выбранными структурами данных, которые реализуют присущие данному объекту операции. В данной книге предполагается, что в используемой системе программирования доступны такие общеупотребительные структуры данных, как массивы, структуры (или записи) и указатели. 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 введенных операций.

Следует подчеркнуть, что, как правило, один и тот же объект может быть представлен многими разными способами, причем 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 отиосительной частоты использования операций в конкретной задаче и т. д. Умение выбрать наиболее подходящее 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.

1.4.1. Реализация операций над подмножествами заданного универсума *U*

Пусть универсум U — конечный, и число элементов в нем не превосходит разрядности ЭВМ: $|U| < n$. Элементы универсума нумеруются: $U = \{u_1, \ldots, u_n\}$. Подмножество A универсума U представляется кодом (машинным словом или битовой шкалой) C, в котором:

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

где $C[i]$ — это i-й разряд кода C.

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 ЭВМ для этих операций есть соответствующие машинные команды. Таким образом, операции над небольшими множествами выполняются весьма эффективно.

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.

1.4.2. Генерация всех подмножеств универсума

Во многих переборных алгоритмах требуется последовательно рассмотреть все подмножества заданного множества. В большинстве компьютеров целые числа представляются кодами в двоичной системе счисления, причем число $2^k - 1$ представляется кодом, содержащим k единиц. Таким образом, число 0 является представлением пустого множества $\varnothing$, число 1 является представлением подмножества, состоящего из первого элемента, и т. д. Следующий тривиальный алгоритм перечисляет все подмножества n-элементного множества.

Алгоритм 1.1. Алгоритм генерации 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
end for
```

Обоснование

Алгоритм выдает 2^n различных целых чисел, следовательно, 2^n различных кодов. С увеличением числа увеличивается количество двоичных разрядов, необходимых для его представления. Самое большое (из генерируемых) число $2^n - 1$ требует для своего представления ровно n разрядов. Таким образом, все подмножества генерируются, причем 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 будет перечислено подмножество с кодом 1000.

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 трудоемкой и зависеть от состава элементов очередного рассматриваемого подмножества. В частности, если очередное рассматриваемое подмножество незначительно отличается по набору элементов от предыдущего, то иногда можно воспользоваться результатами оценки элементов, которые рассматривались на предыдущем шаге перебора. В таком случае, если перебирать множества в определенном порядке, можно значительно ускорить работу переборного алгоритма.

1.4.3. Алгоритм построения бинарного кода Грея

Данный алгоритм генерирует последовательность всех подмножеств n-элементного множества, причем каждое следующее подмножество получается из предыдущего удалением или добавлением одного элемента.

Алгоритм 1.2. Алгоритм построения 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 элемента, подлежащего добавлению или удалению }
  B[p] := 1 − B[p] { добавление или удаление элемента }
  yield B { очередное подмножество }
end for
proc Q(i) { количество 2 в разложении i на множители + 1 }
q := 1; j := i
while j четно do
  j := j/2; q := q + 1
end while
return q
end proc
```

Обоснование

Для $n = 1$ искомая последовательность кодов суть 0, 1. Пусть есть 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 выдает 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 свое 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 $0 \leqslant m \leqslant 2^k - 1$. □

Пример

Протокол выполнения алгоритма 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

1.4.4. 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 представляется указателем на первый элемент.

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

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

Если элементы в списках упорядочить, например, по возрастанию значения поля i, то трудоемкость всех операций составит $O(n)$. Эффективная реализация операций над множествами, представленными в виде упорядоченных списков, основана на весьма общем алгоритме, известном как алгоритм типа *слияния*.

Алгоритм типа слияния параллельно просматривает два множества, представленных упорядоченными списками, причем на каждом шаге продвижение происходит в том множестве, в котором текущий элемент меньше.

1.4.5. Проверка включения слиянием

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

Алгоритм 1.3. Проверка включения слиянием

Вход: проверяемые множества A и B, которые заданы указателями a и b.
Выход: 1, если $A \subset B$, в противном случае 0.

```
pa := a; pb := b
while pa ≠ nil & pb ≠ nil do
  if pa.i < pb.i then
    return 0 { 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
```

Обоснование

На каждом шаге основного цикла возможна одна из трех 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 случая: либо $pa = \mathbf{nil}$, либо $pa \neq \mathbf{nil}$. Первый случай означает, что для всех элементов множества A удалось найти совпадающие элементы в множестве B. Второй случай означает, что множество B закончилось раньше, то есть не для всех элементов множества A удалось найти совпадающие элементы в множестве B. □

1.4.6. Вычисление объединения слиянием

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

Алгоритм 1.4. 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
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
  end if
  Append(c, e, d){ добавление элемента d в конец списка c }
end while
p := nil
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)
  p := p.n
end while
```

Обоснование

На каждом шаге основного цикла возможна одна из трех 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, причем 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 1.5. 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
else
  e.n := q
end if
e := q
```

Обоснование

До первого вызова функции Append имеем: $c = \mathbf{nil}$ и $e = \mathbf{nil}$. 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, поскольку из текста основной программы видно, что, кроме инициализации, все остальные манипуляции с этими указателями выполняются только с помощью функции Append. □

1.4.7. Вычисление пересечения слиянием

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

Алгоритм 1.6. 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
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
  end if
end while
```

Обоснование

На каждом шаге основного цикла возможна одна из трех 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, причем ровно один раз. □

1.5. Отношения

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

1.5.1. Упорядоченные пары

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

$$(a,b) = (c,d) :\Leftrightarrow a = c \,\&\, b = d.$$

Вообще говоря, $(a,b) \neq (b,a)$.

ЗАМЕЧАНИЕ

Упорядоченные пары можно рассматривать как множества, если определить их так:

$$(a,b) :\Leftrightarrow \{a,\{a,b\}\}.$$

Таким образом, понятие упорядоченной пары не выводит рассмотрение за пределы теории множеств, но независимое определение технически удобнее.

1.5.2. Прямое произведение множеств

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

$$A \times B :\Leftrightarrow \{(a,b) \mid a \in A \,\&\, b \in B\}.$$

ЗАМЕЧАНИЕ

Точка на плоскости может быть задана упорядоченной парой координат, то есть двумя точками на координатных осях. Таким образом, $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$. Метод координат ввел в употребление Рене Декарт (1596–1650), отсюда и название «декартово произведение».

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

$$A^n :\Leftrightarrow \underbrace{A \times \cdots \times A}_{n \text{ раз}}.$$

Соответственно, $A^1 :\Leftrightarrow A$, $A^2 :\Leftrightarrow A \times A$ и вообще $A^n :\Leftrightarrow A \times A^{n-1}$.

ТЕОРЕМА $|A \times B| = |A|\,|B|$.

Доказательство

Первый компонент упорядоченной пары можно выбрать $|A|$ способами, второй — $|B|$ способами. Таким образом, всего имеется $|A||B|$ различных упорядоченных пар. □

СЛЕДСТВИЕ $|A^n| = |A|^n$.

1.5.3. Отношения

Пусть A и B — два множества. *(Бинарным) отношением* R из множества A в множество B называется подмножество прямого произведения A и B:

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

Для бинарных отношений обычно используется *инфиксная* форма записи:

$$aRb := (a,b) \in R \subset A \times B.$$

Если $A = B$, то говорят, что R есть отношение *на множестве* A.

Пример

Пусть задан универсум U. Тогда $\in$ (принадлежность) — отношение из множества U в множество 2^U, а $\subset$ (включение) и $=$ (равенство) — отношения на 2^U. Хорошо известны отношения $=, <, \leqslant, >, \geqslant, \neq$, определенные на множестве чисел.

Пусть R есть отношение на A: $R \subset A \times A$, $a, b \in A$. Введем следующие понятия:

Обратное отношение: $R^{-1} := \{(a,b) \mid (b,a) \in R\}$.

Дополнение отношения: $\overline{R} := \{(a,b) \mid (a,b) \notin R\}$.

Тождественное отношение: $I := \{(a,a) \mid a \in A\}$.

Универсальное отношение: $U := \{(a,b) \mid a \in A \,\&\, b \in A\}$.

Введем обобщенное понятие отношения: *n-местное* (*n-арное*) отношение R — это множество упорядоченных наборов (*кортежей*):

$$R \subset A_1 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_1 \in A_1 \,\&\, \ldots \,\&\, a_n \in A_n\}.$$

Множества A_i не обязательно различны.

ОТСТУПЛЕНИЕ

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

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

1.5.4. Композиция отношений

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

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

Композиция отношений на множестве A является отношением на множестве A

1.5.5. Степень отношения

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

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

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

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

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

Доказательство

Существование требуемой степени k отношения R обеспечивается следующим построением.

while $k \geqslant n$ **do**
$c_0 := a$; $c_k := b$
$(a, b) \in R^k \Longrightarrow \quad \exists c_1, \ldots, c_{k-1} \in A \; c_0 R c_1 R c_2 R \ldots R c_{k-1} R c_k$
$|A| = n \Longrightarrow \exists i, j \; c_i = c_j \Longrightarrow c_0 R c_1 R \ldots R c_i R c_{j+1} R \ldots R c_{k-1} R c_k \Longrightarrow$
$\Longrightarrow (a, b) \in R^{k-(j-i)}$
$k := k - (j - i)$
end while □

СЛЕДСТВИЕ

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

1.5.6. Ядро отношения

Если $R \subset A \times B$ — отношение из A в B, то $R \circ R^{-1}$ называется *ядром* отношения R. Ядро отношения R из A в B является отношением на A.

Пример

Пусть задано множество M, $|M| = n$. Рассмотрим отношение P из булеана 2^M в множество целых чисел

$$0..n = \{0, 1, \ldots, n\}, \quad P \subset 2^M \times 0..n,$$

где $P := \{(X, k) \mid X \subset M \,\&\, k \in 0..n \,\&\, |X| = k\}$. Тогда ядром отношения P является отношение равномощности.

1.5.7. 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 $\forall a, b, c \in A\ aRb \,\&\, bRc \Longrightarrow aRc$;
полным, или линейным,	если $\forall a, b \in A\ a \neq b \Longrightarrow aRb \vee bRa$.

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

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

Доказательство

Используя определения свойств отношений, имеем:

1. $\Longrightarrow$: $\forall a \in A\ aRa \Longrightarrow \forall a \in A\ (a, a) \in R \Longrightarrow I \subset R$.
1. $\Longleftarrow$: $I \subset R \Longrightarrow \forall a\ (a, a) \in R \Longrightarrow \forall a\ aRa$.
2. $\Longrightarrow$: $(\forall a, b \in A\ aRb \Longrightarrow bRa) \Longrightarrow \big(\forall a, b \in A\ (a, b) \in R \Longrightarrow (b, a) \in R\big)$, $\big((a, b) \in R = (b, a) \in R^{-1}\big) \Longrightarrow (R \subset R^{-1}\ \&\ R^{-1} \subset R) \Longrightarrow R = R^{-1}$.
2. $\Longleftarrow$: $R = R^{-1} \Longrightarrow \forall a, b \in A\ \big((a, b) \in R \Longrightarrow (a, b) \in R^{-1}\ \&(a, b) \in R^{-1} \Longrightarrow$ $\Longrightarrow (a, b) \in R\big)$, $\big((a, b) \in R = (b, a) \in R^{-1}\big) \Longrightarrow$ $\Longrightarrow \big((a, b) \in R \Longrightarrow (b, a) \in R\big) \Longrightarrow (\forall a, b \in A\ aRb \Longrightarrow bRa)$.
3. $\Longrightarrow$: $\forall a, b, c\ aRb \,\&\, bRc \Longrightarrow aRc$; $aR \circ Rb = \exists c \in A\ aRc \,\&\, cRb \Longrightarrow$ $\Longrightarrow aRb \Longrightarrow R \circ R \subset R$.
3. $\Longleftarrow$: $R \circ R \subset R \Longrightarrow \big(\forall a, b \in A\ (a, b) \in R \circ R \Longrightarrow (a, b) \in R\big) \Longrightarrow$ $\Longrightarrow (aRc \,\&\, cRb \Longrightarrow aRb)$.
4. $\Longrightarrow$: От противного. $R \cap R^{-1} \not\subset I \Longrightarrow \exists a \neq b\ aRb \,\&\, aR^{-1}b \Longrightarrow$ $\Longrightarrow \exists a \neq b\ aRb \,\&\, bRa$.
4. $\Longleftarrow$: $R \cap R^{-1} \subset I \Longrightarrow (aRb \,\&\, aR^{-1}b \Longrightarrow a = b) \Longrightarrow (aRb \,\&\, bRa \Longrightarrow a = b)$.
5. $\Longrightarrow$: От противного. $R \cap I \neq \varnothing \Longrightarrow (\exists a \in A\ aRa \,\&\, aIa) = (\exists a \in A\ aRa) =$ $= \neg\forall a \in A\ \neg aRa$.
5. $\Longleftarrow$: $R \cap I = \varnothing \Longrightarrow \neg\exists a \in A\ aRa \Longrightarrow \forall a \in A\ \neg aRa$.

6. $\Longrightarrow$: $(\forall a,b\in A\ a\neq b\Longrightarrow aRb\vee bRa)\Longrightarrow \big(a\neq b\Longrightarrow (a,b)\in R\cup R^{-1}\big),\ \big(a=b\Longrightarrow$
$\Longrightarrow (a,b)\in I\big)\Longrightarrow \big(\forall a,b\in A\ (a,b)\in R\cup I\cup R^{-1}\big)\Longrightarrow U\subset R\cup I\cup R^{-1}\Longrightarrow$
$\Longrightarrow U=R\cup I\cup R^{-1}$.

6. $\Longleftarrow$: $R\cup I\cup R^{-1}=U\Longrightarrow \big(a=b\,\&\,(a,b)\in I\vee a\neq b\,\&\,(a,b)\in R\cup R^{-1}\big)\Longrightarrow$
$\Longrightarrow \big(a=b\,\&\,(a,b)\in I\vee (a\neq b\Longrightarrow (a,b)\in R\vee (a,b)\in R^{-1})\big)\Longrightarrow$
$\Longrightarrow (a\neq b\Longrightarrow aRb\vee bRa)$. □

1.5.8. Представление отношений в ЭВМ

Пусть $R\subset A^2$ и $|A|=n$. Перенумеруем элементы множества A. Тогда отношение R можно представить матрицей R : **array** $[1..n,1..n]$ **of** $0..1$, где

$$R[i,j]=\begin{cases}1, & \text{если } iRj,\\ 0, & \text{если } i\overline{R}j.\end{cases}$$

Матрица отношения R обозначается $\boxed{R}$.

ТЕОРЕМА

$\boxed{R_1\circ R_2}=\boxed{R_1}\times\boxed{R_2}$, *где* $\left(\boxed{R_1}\times\boxed{R_2}\right)[i,j]:=\bigvee\limits_{k=1}^{n}\boxed{R_1}[i,k]\,\&\,\boxed{R_2}[k,j]$

Доказательство

Пусть $(a,b)\in R_1\circ R_2$.

Тогда $\exists c\in A\ aR_1c\,\&\,cR_2b\Longrightarrow \boxed{R_1}[a,c]=1\,\&\,\boxed{R_2}[c,b]=1\Longrightarrow$

$\Longrightarrow \left(\boxed{R_1}[a,c]\,\&\,\boxed{R_2}[c,b]\right)=1\Longrightarrow \left(\bigvee\limits_{k=1}^{n}\boxed{R_1}[a,k]\,\&\,\boxed{R_2}[k,b]\right)=1$

и, следовательно, $\left(\boxed{R_1}\times\boxed{R_2}\right)[a,b]=1$.

Пусть теперь $(a,b)\notin R_1\circ R_2$.

Тогда $\neg\exists c\in A\ aR_1c\,\&\,cR_2b\Longrightarrow \forall c\in A\ \neg(aR_1c\,\&\,cR_2b)\Longrightarrow$

$\Longrightarrow \forall c\in A\ \neg aR_1c\vee\neg cR_2b\Longrightarrow \forall c\in A\ \boxed{R_1}[a,c]=0\vee\boxed{R_2}[c,b]=0\Longrightarrow$

$\Longrightarrow \forall c\in A\ \left(\boxed{R_1}[a,c]\,\&\,\boxed{R_2}[c,b]\right)=0\Longrightarrow \left(\bigvee\limits_{k=1}^{n}\boxed{R_1}[a,k]\,\&\,\boxed{R_2}[k,b]\right)=0$

и, следовательно, $\left(\boxed{R_1}\times\boxed{R_2}\right)[a,b]=0$. □

СЛЕДСТВИЕ $\boxed{R^k}=\boxed{R}^k$

ТЕОРЕМА

$\boxed{R_1\cup R_2}=\boxed{R_1}\vee\boxed{R_2}$, *где* $\left(\boxed{R_1}\vee\boxed{R_2}\right)[i,j]:=\boxed{R_1}[i,j]\vee\boxed{R_2}[i,j]$.

Доказательство

Пусть $(a,b) \in R_1 \cup R_2$.

Тогда $aR_1b \vee aR_2b \Longrightarrow \boxed{R_1}\,[a,b]=1 \vee \boxed{R_2}\,[a,b]=1 \Longrightarrow \boxed{R_1}\,[a,b] \vee \boxed{R_2}\,[a,b] \Longrightarrow$
$\Longrightarrow \left(\boxed{R_1} \vee \boxed{R_2}\right)\,[a,b]=1.$

Пусть теперь $(a,b) \notin R_1 \cup R_2$.

Тогда $\neg aR_1b \,\&\, \neg aR_2b \Longrightarrow \boxed{R_1}\,[a,b]=0 \,\&\, \boxed{R_2}\,[a,b]=0 \Longrightarrow$
$\Longrightarrow \left(\boxed{R_1} \vee \boxed{R_2}\right)\,[a,b]=0.$ □

1.6. Функции

Понятие «функции» является одним из основополагающих в математике. В данном случае подразумеваются прежде всего функции, отображающие одно конечное множество объектов в другое конечное множество. Мы избегаем использования термина «отображение» и предпочитаем слово «функция» в расчете на постоянное сопоставление читателем математического понятия функции с понятием функции в языках программирования.

1.6.1. Определения

Пусть f — отношение из A в 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 и обозначается следующим образом:

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

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

$$b = f(a) :=(a,b) \in f.$$

Если $b = f(a)$, то a называют *аргументом*, а b — *значением* функции.

ЗАМЕЧАНИЕ

Вообще, всякому отношению R из A в B ($R \subset A \times B$) можно сопоставить (тотальную) функцию $R\colon A \times B \to 0..1$ (эта функция называется *характеристической* функцией отношения), полагая

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

Пусть $f\colon A \to B$, тогда

область определения функции: $f_A := \{a \in A \mid \exists b \in B\ b = f(a)\}$;

область значений функции: $f_B := \{b \in B \mid \exists a \in A\ b = f(a)\}$.

Если $f_A = A$, то функция называется *тотальной*, а если $f_A \neq A$ — *частичной*. *Сужением* функции $f\colon A \to B$ на множество $M \subset A$ называется функция $f|_M$, определяемая следующим образом:

$$f|_M := \{(a,b) \mid (a,b) \in f \,\&\, a \in M\}.$$

Для тотальной функции $f = f|_{f_A}$.

Функция $f\colon A_1 \times \cdots \times A_n \to B$ называется функцией n *аргументов*, или n-*местной* функцией.

1.6.2. Инъекция, сюръекция и биекция

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

инъективной, если $b = f(a_1) \,\&\, b = f(a_2) \Longrightarrow a_1 = a_2$;

сюръективной, если $\forall b \in B\ \exists a \in A\ b = f(a)$;

биективной, если она инъективная и сюръективная.

ЗАМЕЧАНИЕ

Биективную функцию также называют *взаимнооднозначной*.

Рис. 1.2 иллюстрирует понятия отношения, функции, инъекции, сюръекции и биекции.

ТЕОРЕМА *Если $f\colon A \to B$ — тотальная биекция ($f_A = A$), 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.

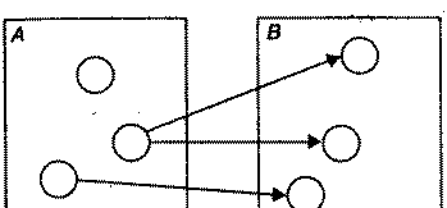
$$f^{-1} := \{(b,a) \mid a \in A \,\&\, b \in B \,\&\, b = f(a)\}.$$

Пусть $a_1 = f^{-1}(b) \,\&\, a_2 = f^{-1}(b)$. Тогда $b = f(a_1) \,\&\, b = f(a_2) \Longrightarrow a_1 = a_2$.

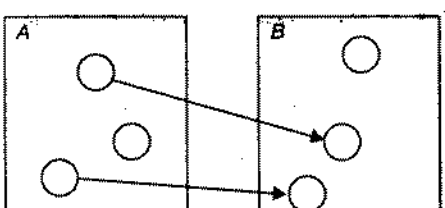
Покажем, что f^{-1} — инъекция. Пусть $a = f^{-1}(b_1) \,\&\, a = f^{-1}(b_2)$. Тогда $b_1 = f(a) \,\&\, b_2 = f(a) \Longrightarrow b_1 = b_2$.

Покажем от противного, что f^{-1} — сюръекция.

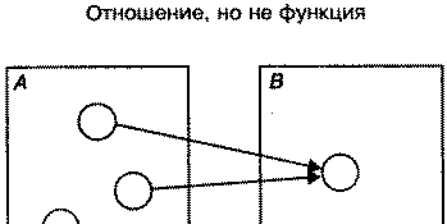
Пусть $\exists a \in A\ \neg\exists b \in B\ a = f^{-1}(b)$. Тогда $\exists a \in A\ \forall b \in B\ a \neq f^{-1}(b)$. Обозначим этот элемент a_0. Имеем $\forall b\ a_0 \neq f^{-1}(b) \Longrightarrow \forall b\ b \neq f(a_0) \Longrightarrow a_0 \notin f_A \subset A \Longrightarrow$ $\Longrightarrow a_0 \notin A$. □



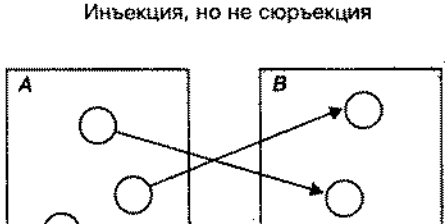
Отношение, но не функция



Инъекция, но не сюръекция



Сюръекция, но не инъекция



Биекция

Рис. 1.2. Различные виды функций

1.6.3. Индуцированная функция

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

$$F(A_1) := \{b \in B \mid \exists a \in A_1 \; b \quad f(a)\}$$

называется *образом* множества A_1, а множество

$$F^{-1}(B_1) := \{a \in A \mid \exists b \in B_1 \; b = f(a)\}$$

прообразом множества B_1. Заметим, что F является отношением из множества 2^{f_A} в множество 2^{f_B}:

$$F := \{(A_1, B_1) \mid A_1 \subset A \,\&\, B_1 \subset B \,\&\, B_1 = F(A_1)\}$$

ТЕОРЕМА *Если* $f\colon A \to B$ *функция, то* $F\colon 2^{f_A} \to 2^{f_B}$ *и* $F^{-1} \quad 2^{f_B} \to 2^{f_A}$ — *тоже функции.*

ЗАМЕЧАНИЕ

F называется *индуцированной* функцией, а F^{-1} — *переходом к прообразам*

1.6.4. Представление функций в ЭВМ

Пусть $f: A \to B$, 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 получение значения 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 некоторых других языках вызов функции и обращение к массиву синтаксически неотличимы, 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 1.5.7, 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 определенным набором свойств. Такое «абстрактное» изучение классов отношений обладает тем преимуществом, что один раз установив некоторые следствия из наличия у отношения определенного 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. Определения

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

Пример

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

1.7.2. Классы эквивалентности

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

$$[x]_{\equiv} := \{y \mid y \in M \,\&\, y \equiv x\}$$

Если отношение подразумевается, то индекс $\equiv$ 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 $x \in [a] \Longrightarrow x \equiv a \,\&\, a \equiv b \Longrightarrow x \equiv b \Longrightarrow x \in [b]$;
$x \in [b] \Longrightarrow x \equiv b \Longrightarrow x \equiv b \,\&\, a \equiv b \Longrightarrow x \equiv b \,\&\, b \equiv a \Longrightarrow x \equiv a \Longrightarrow x \in [a]$. □

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

Доказательство

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

ТЕОРЕМА *Всякое отношение эквивалентности на множестве M определяет разбиение множества M, причем среди элементов разбиения нет пустых; и обратно, всякое разбиение множества M, не содержащее пустых элементов, определяет отношение эквивалентности на множестве M:*

$$\equiv \subset M^2 \Longleftrightarrow \exists \mathcal{B} = \{B_i\}\ B_i \subset M \,\&\, B_i \neq \varnothing \,\&\, M = \bigcup B_i \,\&\, \forall i, j\ i \neq j \Longrightarrow$$
$$\Longrightarrow B_i \cap B_j = \varnothing.$$

Доказательство

Необходимость. Построение требуемого разбиения обеспечивается следующим алгоритмом.

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

```
U := M; B := ∅
while U ≠ ∅ do
  select a ∈ U { возьмем любой элемент из M }
  A := Eq(a, M, ≡) { построим его класс эквивалентности }
  U := U \ A { удалим класс из множества }
  B := B ∪ {A} { и добавим в разбиение }
end while
```

Фуикция Eq обеспечивает построение класса эквивалентности:

Вход: элемент a из множества M, отношение эквивалентности $\equiv$ на M.
Выход: последовательность элементов, образующих класс эквивалентности A.

```
for b ∈ M do
  if b ≡ a then
    yield b
  end if
end for
return A
```

Достаточность. Положим $a \equiv b := \exists i\ a \in B_i \,\&\, b \in B_i$. Тогда

1. рефлексивность: $M = \cup B_i \Longrightarrow \forall a \in M\ \exists i\ a \in B_i$;
   $a \in B_i \Longrightarrow a \in B_i \,\&\, a \in B_i \Longrightarrow a \equiv a$;
2. симметричность: $a \equiv b \Longrightarrow \exists i\ a \in B_i \,\&\, b \in B_i \Longrightarrow \exists i\ b \in B_i \,\&\, a \in B_i \Longrightarrow b \equiv a$;
3. транзитивность: $a \equiv b \,\&\, b \equiv c \Longrightarrow [a] = [b] \,\&\, [b] = [c] \Longrightarrow [a] = [c] \Longrightarrow a \equiv c$. □

1.7.3. Фактормножества

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

$$M/R := \{[x]_R\}_{x \in M}$$

Фактормножество является подмножеством булеана: $M/R \subset 2^M$.

Функция $\mathrm{nat}\, R\colon M \to M/R$ называется *отождествлением* и определяется следующим образом:

$$\mathrm{nat}\, R(x) := [x]_R.$$

1.7.4. Ядро функции

Всякая функция, будучи отношением, имеет ядро. *Ядро* функции f обозначается $\ker f$:

$$\ker f := f \circ f^{-1}$$

ТЕОРЕМА *Ядро функции является отношением эквивалентности на области определения функции.*

Доказательство

Пусть $f\colon A \to B$. Не ограничивая общности, можно считать, что $f_A = A$ и $f_B = B$.
Тогда $\forall b \in B\ \exists a \in A\ b = f(a) \,\&\, \forall a \in A\ \exists b \in B\ b = f(a)$,
$\ker f = \{(a_1, a_2) \mid a_1, a_2 \in A \,\&\, \exists b\ b = f(a_1) \,\&\, a_2 \in f^{-1}(b)\}$.

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

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

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

Пример

Мощность множества является функцией из множества конечных множеств в множество неотрицательных целых чисел. Ядро этой функции — это отношение равномощности, которое является, таким образом, отношением эквивалентности.

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 1.4.4–1.4.7. Здесь вводятся точные определения, которые предполагаются известными в остальной части книги.

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 полным (линейным), и тогда оно называется отношением *полного*, или *линейного порядка*. Отношение порядка может не обладать свойством полноты (линейности), и тогда оно называется отношением *частичного порядка*.

Обычно отношение строгого порядка (полного 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: $\forall a, b\ a < b \Longrightarrow \neg(b < a)$.

Пример

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

Множество, на котором определено отношение частичного порядка, называется *частично упорядоченным*. Множество, на котором определено отношение полного порядка, называется *вполне упорядоченным*.

Пример

Множество чисел упорядочено линейно, а булеан упорядочен частично.

1.8.2. Минимальные элементы

Элемент x множества M с отношением порядка $\prec$ называется *минимальным*, если 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 *Во всяком конечном непустом частично упорядоченном множестве существует минимальный элемент.*

Доказательство

От противного. Пусть $\neg(\exists x \in M\ \neg\exists y \in M\ y \prec x)$. 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

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

ТЕОРЕМА *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 полного 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 1.7. Алгоритм топологической сортировки

Вход: частично упорядоченное множество U.
Выход: вполне упорядоченное множество W.

```
while U ≠ ∅ do
  m := M(U) { функция M возвращает минимальный элемент }
  yield m { такой элемент m существует по теореме предыдущего раздела }
  U := U \ {m}
end while
```

ЗАМЕЧАНИЕ

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.

Обоснование

Существование функции M обеспечивается первой теоремой раздела 1.8.2. □

ЗАМЕЧАНИЕ

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

1.8.4. Монотонные функции

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

Тогда если $a_1 < a_2 \Longrightarrow f(a_1) \leqslant f(a_2)$, то функция f называется *монотонной*, а если $a_1 < a_2 \Longrightarrow f(a_1) < f(a_2)$, то функция f называется *строго монотонной*.

Пример

Функция мощности конечного множества $|\,|\colon 2^M \to \mathrm{N}$ является монотонной.

1.9. 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 определенные 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.

1.9.1. Замыкание отношения относительно свойства

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: $C(R'') \,\&\, R \subset R'' \Longrightarrow R' \subset R''$.

1.9.2. Транзитивное и рефлексивное транзитивное замыкания

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

$$R^+ = \bigcup_{i=1}^{\infty} R^i, \qquad R^* := \bigcup_{i=0}^{\infty} R^i$$

ТЕОРЕМА *R^+ — транзитивное замыкание R.*

Доказательство

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

1. 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, $aRc_1R \ldots Rc_{n-1}RbRd_1R \ldots Rd_{m-1}Rc \Longrightarrow aR^{n+m+1}c \Longrightarrow aR^+c$.
2. $R = R^1 \Longrightarrow R \subset \bigcup_{i=1}^{\infty} R^i \Longrightarrow R \subset R^+$.
3. $aR^+b \Longrightarrow \exists k\ aR^kb \Longrightarrow \exists c_1, \ldots, c_{k-1}\ aRc_1R \ldots Rc_{k-1}Rb$;
   $R \subset R'' \Longrightarrow aR''c_1R'' \ldots R''c_{k-1}R''b \Longrightarrow aR''b$.
   Таким образом, $R^+ \subset R''$. □

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

Вычислить транзитивное замыкание заданного отношения можно следующим образом:

$$R^+ = \bigcup_{i=1}^{\infty} R^i = \bigcup_{i=1}^{\cdot\cdot} R^i \Longrightarrow \boxed{R^+} = \bigvee_{i=1}^{\cdot\cdot} \boxed{R^i} = \bigvee_{i=1}^{\cdot\cdot} \boxed{R}^i$$

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

1.9.3. Алгоритм Уоршалла

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

Алгоритм 1.8. Алгоритм Уоршалла
Вход: отношение, заданное матрицей R.
Выход: транзитивное замыкание отношения, заданное матрицей T.

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

```
    for j from 1 to n do
      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 от 1 до i, связанных отношением R. Действительно, последовательность промежуточных элементов с номерами в диапазоне от 1 до 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 до $i-1$ для пары элементов с номерами j и k, либо существуют *две* последовательности элементов с номерами в диапазоне от 1 до $i-1$ — 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. □

Комментарии

Основные сведения, изложенные в этой главе, можно найти в *любом* учебнике по (дискретной) математике. Алгоритм 1.2 описан в [14]. Алгоритмы 1.3, 1.4, 1.6 — общеизвестный программистский фольклор. Алгоритм 1.7 описан в фундаментальной книге [8], которая входит в «золотой список» обязательного чтения любого программиста. Алгоритм 1.8 описан во многих источниках, например в [1].

Упражнения

1.1. Существует ли множество всех множеств?

1.2. Доказать, что $|A \cup B| = |A| + |B| - |A \cap B|$.

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

1.4. Доказать, что $Q(2^k + m) = Q(2^k - m)$ для $0 \leqslant m \leqslant 2^k - 1$, где Q — функция из алгоритма 1.2.

1.5. Пусть $Q := \{(m,n) \mid m,n \in \mathbb{N} \,\&\, m = n^2\}$. Какими свойствами обладает отношение Q?

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

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

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

1.9. Доказать вторую теорему из подраздела 1.9.2.

ГЛАВА 2 Алгебраические структуры

Алгебраические методы описания моделей находят самое широкое применение при формализации различных предметных областей. Грубо говоря, при построении модели предметной области все 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. В начале 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.

2.1. Операции и алгебры

Словом «алгебра» обозначают, вообще говоря, не только отрасль математики, но и один из конкретных объектов, изучаемых в этой отрасли. К счастью, «алгебры» в узком смысле здесь не рассматриваются, а потому для краткости и без риска возникновения недоразумений слово «алгебра» используется как родовое понятие для обозначения разнообразных алгебраических структур.

2.1.1. Алгебраические структуры

Всюду определенная (тотальная) функция $\varphi\colon M^n \to M$ называется *n-арной (n-местной) операцией* на M.

Если операция φ — бинарная (то есть $\varphi\colon M \times M \to M$), то будем писать $a\varphi b$ вместо $\varphi(a, b)$ или $a \circ b$, где $\circ$ — знак операции.

ЗАМЕЧАНИЕ

Такая форма записи называется *инфиксной*.

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*. 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 $\langle M; \varphi_1, \ldots, \varphi_m\rangle$.

ЗАМЕЧАНИЕ

Операции φ_i *конечноместны (финитарны)*, сигнатура Σ конечна. Носитель не обязательно конечен, но не пуст.

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, причем $\varphi_i\colon M_{i_1} \times \cdots \times M_{i_{n_i}} \to M_j$. Тогда $\langle M; \Sigma\rangle$ называется *многоосновной* алгеброй. Другими словами, многоосновная алгебра имеет несколько носителей, а каждая операция сигнатуры действует из прямого произведения некоторых носителей в некоторый носитель.

2.1.2. Замыкания и подалгебры

Подмножество $X \subset M$ 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 $\Sigma_X = \{\varphi_i^X\}$, $\varphi_i^X = \varphi_i\big|_{X^k}$, $k = n_i$.

Пример

1. Алгебра $\langle \mathbb{R}; +, \cdot\rangle$ — *поле действительных чисел*. Тип — (2,2). Все конечные подмножества, кроме $\{0\}$, не замкнуты относительно обеих операций. *Поле рациональных чисел* $\langle \mathbb{Q}; +, \cdot\rangle$ образует подалгебру.
2. Алгебра $\langle 2^M; \cup, \cap, {}^-\rangle$ — *алгебра подмножеств* над множеством M. Тип — (2,2,1). При этом $\langle 2^X; \cup, \cap, {}^-\rangle$ для любого подмножества X множества M образует подалгебру.
3. Алгебра $\left\langle \{f \mid f\colon \mathbb{R} \to \mathbb{R}\}; \frac{d}{dx}\right\rangle$, где $\frac{d}{dx}$ — операция дифференцирования. Множество элементарных функций образует подалгебру.

ТЕОРЕМА *Непустое пересечение подалгебр образует подалгебру.*

Доказательство

Пусть $\langle X_i; \Sigma_{X_i}\rangle$ — подалгебра $\langle M; \Sigma\rangle$. Тогда $\forall i\ \forall j\ \varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in X_i \Longrightarrow$ $\Longrightarrow \forall j\ \varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in \bigcap X_i$. □

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

Пример

В алгебре *целых чисел* $\langle \mathbb{Z}; +, \cdot \rangle$ замыканием числа 2 являются четные 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. $[X] \cup [Y] \subset [X \cup Y]$.

2.1.3. Алгебра термов

Пусть заданы сигнатура $\Sigma = (\varphi_1, \ldots, \varphi_m)$ типа $N = (n_1, \ldots, n_m)$ и множество переменных $V = \{x_1, x_2, \ldots\}$. Определим множество *термов* T в сигнатуре Σ следующим образом:

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

Алгебра $\langle T; \Sigma \rangle$ называется *свободной алгеброй термов*, или *Σ-алгеброй*.

ОТСТУПЛЕНИЕ

Носителем Σ-алгебры является множество термов, то есть формальных выражений, построенных с помощью знаков операций сигнатуры Σ. Таким образом, с Σ-алгеброй не связано никакое конкретное множество, являющееся носителем. Поэтому Σ-алгебры используются в программировании для определения абстрактных типов данных.

2.1.4. 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 она называется *конечно-порожденной*. Бесконечные алгебры могут иметь конечные системы образующих.

Пример

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

2.1.5. 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* слева: $a\diamond(b\circ c)=(a\diamond b)\circ(a\diamond c)$;
4. *Дистрибутивность* справа: $(a\circ b)\diamond c=(a\diamond c)\circ(b\diamond c)$;
5. *Поглощение*: $(a\circ b)\diamond a=a$;
6. *Идемпотентность*: $a\circ a=a$.

Пример

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.
3. Дистрибутивные операции: умножение относительно сложения чисел. Недистрибутивные операции: возведение в степень дистрибутивно относительно умножения справа, но не слева: $\left((ab)^c=a^cb^c, a^{bc}\neq a^ba^c\right)$.
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.

2.2. Морфизмы

Понятие изоморфизма, введенное в этом разделе, является одним из ключевых. Поэтому следует обратить особое внимание на посвященный изоморфизму подраздел 2.2.2.

2.2.1. Гомоморфизм

Алгебры с различными типами имеют различное *строение*.

Пусть $\mathcal{A}=\langle A;\varphi_1,\ldots,\varphi_m\rangle$ и $\mathcal{B}=\langle B;\psi_1,\ldots,\psi_m\rangle$ — две алгебры одинакового типа. Если существует функция $f\colon A\to B$, такая что

$$\forall i\in 1..m\ f\left(\varphi_i(a_1,\ldots,a_n)\right)=\psi_i\left(f(a_1),\ldots,f(a_n)\right), \tag{2.1}$$

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

Пусть $\mathcal{A} = \langle A; \varphi \rangle$, $\mathcal{B} = \langle B; \psi \rangle$, тип $= (1)$ и $f\colon A \to B$. Действие этих функций 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 (2.1) можно переписать так:

$$\varphi \circ f = f \circ \psi.$$

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 $f := a \mod 10$ — 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* (или *эпиоморфизмом*).
- Гомоморфизм, который является биекцией, называется *изоморфизмом*.
- Если $\mathcal{A} = \mathcal{B}$, то гомоморфизм называется *эндоморфизмом*, а изоморфизм называется *автоморфизмом*.

2.2.2. Изоморфизм

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

Доказательство

Рассмотрим произвольную операцию φ из сигнатуры $\mathcal{A}$ и соответствующую ей операцию ψ из сигнатуры $\mathcal{B}$. Имеем:

$$f(\varphi(a_1, \ldots, a_n)) = \psi(f(a_1), \ldots, f(a_n)),$$

кроме того, f — биекция. Обозначим $b_1 := f(a_1), \ldots, b_n := f(a_n)$, при этом

$$a_1 = f^{-1}(b_1), \ldots, a_n = f^{-1}(b_n).$$

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: $\mathcal{A} \overset{f}{\sim} \mathcal{B}$.

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

Доказательство

1. Рефлексивность: $\mathcal{A} \overset{I}{\sim} \mathcal{A}$, $f := I$.
2. Симметричность: $\mathcal{A} \overset{f}{\sim} \mathcal{B} \Longrightarrow \mathcal{B} \overset{f^{-1}}{\sim} \mathcal{A}$.
3. Транзитивность: $\mathcal{A} \overset{f}{\sim} \mathcal{B} \,\&\, \mathcal{B} \overset{g}{\sim} \mathcal{G} \Longrightarrow \mathcal{A} \overset{f\circ g}{\sim} \mathcal{G}$. $\square$

Пример

1. Пусть $\mathcal{A} = \langle \mathbb{N}; + \rangle$, $\mathcal{B} = \langle \{n \mid n = 2k, k \in \mathbb{N}\}; + \rangle$ — *четные* 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. $\mathcal{A} = \langle \mathbb{R}_+; \cdot \rangle \overset{\ln}{\sim} \mathcal{B} = \langle \mathbb{R}; + \rangle$.

Понятие изоморфизма является одним из центральных понятий, обеспечивающих 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 на соответствующие операции 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*, то есть рассматривать классы эквивалентности по отношению изоморфизма.

2.3. Алгебры с одной операцией

Естественно начать изучение алгебраических структур с наиболее простых. Самой простой структурой является алгебра с одной унарной операцией, но этот

случай настолько тривиален, что про него нечего сказать. Следующим по порядку является случай алгебры с одной бинарной операцией $\circ\colon M \times M \to M$, который рассматривается в этом разделе.

2.3.1. Полугруппы

Полугруппа — это алгебра с одной ассоциативной бинарной операцией:

$$a \circ (b \circ c) = (a \circ b) \circ c.$$

Пример

1. Множество слов A^+ в алфавите A образует полугруппу относительно операции *конкатенации*.
2. Всякое множество функций, 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.

Пусть $P = \langle M; \circ \rangle$ — полугруппа с конечной системой образующих

$$A = \{a_1, \ldots, a_n\}.$$

Тогда $\forall x \in M\ \exists y_1, \ldots, y_k \in A\ x = y_1 \circ \cdots \circ y_k$. Если опустить обозначение операции $\circ$, 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 различны, 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 полугруппа может быть получена из свободной введением определяющих соотношений. Отношение равенства слов в полугруппе с определяющими соотношениями является отношением эквивалентности.

Пример

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.

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

Доказательство

Без доказательства. □

2.3.2. Моноиды

Моноид это полугруппа с *единицей*:

$$\exists e\ \forall a\ a \circ e = e \circ a = a.$$

Пример

1. Множество слов A^* в алфавите A вместе с пустым словом Λ 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, причем $v_i \notin t_i$. Результатом применения подстановки σ к терму t (обозначается $t\sigma$) называется терм, который получается заменой *всех* вхождений переменных v_i на соответствующие термы 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 $\sigma := \sigma_1 \circ \sigma_2 := \{t_k//v_k\}_{k\in K}$, 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, причем тождественная подстановка $\{v_i//v_i\}$ является единицей.

ТЕОРЕМА *Единица единственна.*

Доказательство

Пусть $\exists e_1, e_2\ \forall a\ \ a \circ e_1 = e_1 \circ a = a \,\&\, a \circ e_2 = e_2 \circ a = a$. Тогда
$$e_1 \circ e_2 = e_1 \,\&\, e_1 \circ e_2 = e_2 \Longrightarrow e_1 = e_2.$$ □

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

[1] А. А. Марков (1903–1979)

Доказательство

Пусть $\mu = \langle M; \bullet \rangle$ — моноид над $M = \{e, a, b, c, \dots\}$. Построим $\mathcal{F} = \langle F; \circ \rangle$ — моноид преобразований над M, где $F := \{f_m : M \to M \mid f_m(x) := x \bullet m\}_{m \in M}$, а $\circ$ — суперпозиция функций, $h : M \to F$, $h(m) := f_m$. Тогда:

1. $\mathcal{F}$ — моноид, поскольку суперпозиция функций ассоциативна, f_e — тождественная функция (так как $f_e(x) = x \bullet e = x$), и F замкнуто относительно $\circ$ (так как $f_a \circ f_b = f_{a \bullet b}$: $f_{a \bullet b}(x) = x \bullet (a \bullet b) = (x \bullet a) \bullet b = f_a(x) \bullet b = f_b(f_a(x))$.
2. h — гомоморфизм (так как $h(a \bullet b) = f_{a \bullet b} = f_a \circ f_b = h(a) \circ h(b)$).
3. h — биекция, поскольку h — сюръекция по построению, и h — инъекция (так как $(f_a(e) = e \bullet a = a \,\&\, f_b(e) = e \bullet b = b) \Longrightarrow (a \neq b \Longrightarrow f_a \neq f_b)$). □

2.3.3. Группы

Группа — это моноид, в котором

$$\forall a \; \exists a^{-1} \; a \circ a^{-1} = a^{-1} \circ a = e.$$

Элемент a^{-1} называется *обратным*.

Пример

1. Множество невырожденных квадратных матриц порядка n образует группу относительно операции умножения матриц. Единицей группы является единичная матрица. Обратным элементом является обратная матрица.
2. Множество подстановок 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 *Обратный элемент единственен.*

Доказательство

Пусть $a \circ a^{-1} = a^{-1} \circ a = e \,\&\, a \circ b = b \circ a = e$. Тогда

$$a^{-1} = a^{-1} \circ e = a^{-1} \circ (a \circ b) = (a^{-1} \circ a) \circ b = e \circ b = b.$$ □

ТЕОРЕМА *В группе выполняются следующие соотношения:*

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

Доказательство

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

ТЕОРЕМА *В группе можно однозначно решить уравнение*

$$a \circ x = b, \qquad (\text{решение: } x = a^{-1} \circ b).$$

Доказательство

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

Коммутативная группа, то есть группа, в которой

$$a \circ b = b \circ a,$$

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

Пример

1. $\langle \mathbb{Z}; + \rangle$ — множество целых чисел образует абелеву группу относительно сложения. Нулем группы является число 0. Обратным элементом является число с противоположным знаком: $x^{-1} := -x$.
2. $\langle \mathbb{Q}_+; \cdot \rangle$ — множество положительных рациональных чисел образует абелеву группу относительно умножения. Нулем группы является число 1. Обратным элементом является обратное число: $(m/n)^{-1} := n/m$.
3. $\langle 2^M; \Delta \rangle$ — булеан образует абелеву группу относительно симметрической разности. Нулем группы является пустое множество $\varnothing$. Обратным элементом является дополнение: $X^{-1} := M \setminus X$.

2.4. Алгебры с двумя операциями

В этом разделе рассматриваются алгебры с двумя бинарными операциями:

$$\oplus, \otimes \colon M \times M \to M,$$

которые условно называются «сложением» и «умножением», соответственно.

[1] Нильс Хенрик Абель (1802–1929)

2.4.1. Кольца

Кольцо — это множество M с двумя бинарными операциями $\oplus$ и $\otimes$, в котором:

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

Кольцо называется *коммутативным*, если

7. $a \otimes b = b \otimes a$ — умножение коммутативно.

Коммутативное кольцо называется кольцом с *единицей*, если

8. $\exists 1 \in M\ a \otimes 1 = 1 \otimes a = a$ — существует единица;
   то есть кольцо с единицей — моноид по умножению.

ТЕОРЕМА *В кольце выполняются следующие соотношения:*

1. $0 \otimes a = a \otimes 0 = 0$;
2. $a \otimes (-b) = (-a) \otimes b = -(a \otimes b)$;
3. $(-a) \otimes (-b) = a \otimes b$.

Доказательство

1. $0 \otimes a = (0 \oplus 0) \otimes a = (0 \otimes a) \oplus (0 \otimes a) \Longrightarrow$
   $\Longrightarrow -(0 \otimes a) \oplus (0 \otimes a) = -(0 \otimes a) \oplus \big((0 \otimes a) \oplus (0 \otimes a)\big) = \big(-(0 \otimes a) \oplus (0 \otimes a)\big) \oplus (0 \otimes a) \Longrightarrow$
   $\Longrightarrow 0 = 0 \oplus (0 \otimes a) = 0 \otimes a.$
2. $\big(a \otimes (-b)\big) \oplus (a \otimes b) = a \otimes (-b \oplus b) = a \otimes 0 = 0 \Longrightarrow$
   $\Longrightarrow a \otimes (-b) = -(a \otimes b),\ (a \otimes b) \oplus \big((-a) \otimes b\big) = \big(a \oplus (-a)\big) \otimes b = 0 \otimes b = 0 \Longrightarrow$
   $\Longrightarrow (-a) \otimes b = -(a \otimes b).$
3. $(-a) \otimes (-b) = -\big(a \otimes (-b)\big) = -\big(-(a \otimes b)\big) = a \otimes b.$ □

Пример

$\langle \mathbb{Z}; +, * \rangle$ — коммутативное кольцо с единицей. Для любого натурального 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.

2.4.2. Области целостности

Если в кольце $\exists x \neq 0\ \exists y \neq 0\ x \otimes y = 0$, то x называется *левым*, а y — *правым делителем нуля*.

Пример

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

В группе $a \circ b = a \circ c \Longrightarrow b = c$, однако в произвольном кольце это не так.

ТЕОРЕМА *Пусть* $a \neq 0$. *Тогда*

$$\left.\begin{array}{l}(a \otimes b = a \otimes c \Longrightarrow b = c)\\ (b \otimes a = c \otimes a \Longrightarrow b = c)\end{array}\right\} \Longleftrightarrow (x \neq 0 \,\&\, y \neq 0 \Longrightarrow x \otimes y \neq 0).$$

Доказательство

$\Longrightarrow$: От противного. Пусть $x \otimes y = 0$. Тогда $x \neq 0 \,\&\, x \otimes y = 0 \,\&\, x \otimes 0 = 0 \Longrightarrow y = 0$, $y \neq 0 \,\&\, x \otimes y = 0 \,\&\, 0 \otimes y = 0 \Longrightarrow x = 0$.

$\Longleftarrow$: $0 = (a \otimes b) \oplus \big(-(a \otimes b)\big) = (a \otimes b) \oplus \big(-(a \otimes c)\big) = (a \otimes b) \oplus \big(a \otimes (-c)\big) = a \otimes \big(b \oplus (-c)\big)$, $a \otimes \big(b \oplus (-c)\big) = 0 \,\&\, a \neq 0 \Longrightarrow b \oplus (-c) = 0 \Longrightarrow b = c$. □

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.

2.4.3. Поля

Поле — это множество M с двумя бинарными операциями $\oplus$ и $\otimes$, такими что:

1. $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ — сложение ассоциативно;
2. $\exists 0 \in M\ a \oplus 0 = 0 \oplus a = a$ — существует нуль;
3. $\forall a\ \exists -a\ a \oplus -a = 0$ — существует обратный элемент по сложению;
4. $a \oplus b = b \oplus a$ — сложение коммутативно, то есть поле — абелева группа по сложению;
5. $a \otimes (b \otimes c) = (a \otimes b) \otimes c$ — умножение ассоциативно;
6. $\exists 1 \in M\ a \otimes 1 = 1 \otimes a = a$ — существует единица;
7. $\forall a \neq 0\ \exists a^{-1}\ a^{-1} \otimes a = 1$ — существует обратный элемент по умножению;
8. $a \otimes b = b \otimes a$ — умножение коммутативно, то есть поле — абелева группа по умножению;

9. $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ умножение дистрибутивно относительно сложения.

Пример

1. $\langle \mathbb{R}; +, \cdot \rangle$ — поле вещественных чисел.
2. $\langle \mathbb{Q}; +, \cdot \rangle$ — поле рациональных чисел.
3. Пусть $E_2 := \{0, 1\}$. Определим операции $\oplus, \cdot : E_2 \times E_2 \to E_2$ следующим образом: $0 \cdot 0 = 0, 0 \cdot 1 = 0, 1 \cdot 0 = 0, 1 \cdot 1 = 1, 0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$. Тогда $\mathcal{E}_2 := \langle E_2; \oplus, \cdot \rangle$ является полем и называется *двоичной арифметикой*.

ТЕОРЕМА *В поле выполняются следующие соотношения.*

1. $(-a) = a \otimes (-1)$;
2. $-(a \oplus b) = (-a) \oplus (-b)$;
3. $a \neq 0 \Longrightarrow (a^{-1})^{-1} = a$;
4. $a \otimes b = 0 \Longrightarrow a = 0 \vee b = 0$.

Доказательство

1. $\big(a \otimes (-1)\big) \oplus a = \big(a \otimes (-1)\big) \oplus (a \otimes 1) = a \otimes (-1 \oplus 1) = a \otimes 0 = 0$.
2. $(a \oplus b) \oplus \big((-a) \oplus (-b)\big) = (a \oplus b) \oplus \big((-b) \oplus (-a)\big) = a \oplus \big(b \oplus (-b)\big) \oplus (-a) = a \oplus 0 \oplus (-a) = a \oplus (-a) = 0$.
3. $a^{-1} \otimes a = 1$.
4. $a \otimes b = 0 \,\&\, a \neq 0 \Longrightarrow$
   $\Longrightarrow b = 1 \otimes b = (a^{-1} \otimes a) \otimes b = a^{-1} \otimes (a \otimes b) = a^{-1} \otimes 0 = 0$,
   $a \otimes b = 0 \,\&\, b \neq 0 \Longrightarrow$
   $\Longrightarrow a = 1 \otimes a = (b^{-1} \otimes b) \otimes a = b^{-1} \otimes (b \otimes a) = b^{-1} \otimes (a \otimes b) = b^{-1} \otimes 0 = 0$. □

ТЕОРЕМА *Если $a \neq 0$, то в поле единственным образом разрешимо уравнение*

$$a \otimes x \oplus b = 0, \qquad \big(x = -(a^{-1}) \otimes b\big).$$

Доказательство

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

2.5. Векторные пространства

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.

2.5.1. Определения

Пусть $\mathcal{F} = \langle F; +, \cdot \rangle$ — некоторое поле с операцией сложения $+$, операцией умножения $\cdot$, *аддитивной* единицей 0 и *мультипликативной* единицей 1. Пусть $\mathcal{V} = \langle V; + \rangle$ — некоторая абелева группа с операцией $+$ и единицей $\mathbf{0}$. 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*, а необозначенная операция $F \times V \to V$ называется *умножением вектора на скаляр*.

Пример

1. Пусть $\mathcal{F} = \langle F; +, \cdot \rangle$ — некоторое поле. Рассмотрим множество кортежей F^n. Тогда $\mathcal{F}^n = \langle F^n; + \rangle$, где $(a_1, \ldots, a_n) + (b_1, \ldots, b_n) := (a_1 + b_1, \ldots, a_n + b_n)$, является абелевой группой, если $-(a_1, \ldots, a_n) := (-a_1, \ldots, -a_n)$ и $\mathbf{0} := (0, \ldots, 0)$. Положим $a(a_1, \ldots, a_n) := (a \cdot a_1, \ldots, a \cdot a_n)$. 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 $\mathcal{E}_2 = \langle E_2; \oplus, \cdot \rangle$ является полем, а булеан $\langle 2^M; \triangle \rangle$ с симметрической разностью является абелевой группой. Положим $1X := X$, $0X := \varnothing$. Тогда булеан с симметрической разностью является векторным пространством над двоичной арифметикой.

2.5.2. Свойства нуль-вектора

Единица группы $\mathcal{V}$ (если $\mathcal{V}$ — векторное пространство) называется *нуль-вектором* и обозначается $\mathbf{0}$.

ТЕОРЕМА $\forall \mathbf{x} \in V \; 0\mathbf{x} = \mathbf{0}, \qquad \forall a \in F \; a\mathbf{0} = \mathbf{0}.$

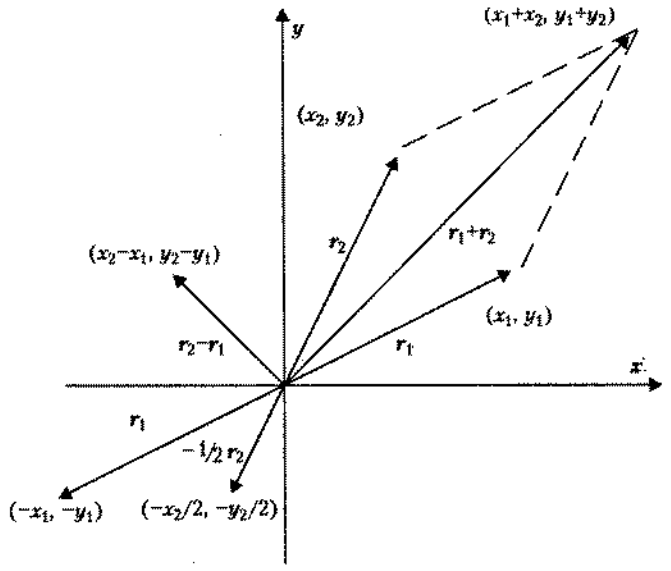


Рис. 2.1. Операции над векторами в $\mathbb{R}^2$

Доказательство

1. $0\mathbf{x} = (1-1)\mathbf{x} = 1\mathbf{x} - 1\mathbf{x} = \mathbf{x} - \mathbf{x} = \mathbf{0}$.
2. $a\mathbf{0} = a(\mathbf{0}-\mathbf{0}) = a\mathbf{0} - a\mathbf{0} = (a-a)\mathbf{0} = 0\mathbf{0} = \mathbf{0}$. □

2.5.3. Линейные комбинации

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

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

называется *линейной комбинацией* векторов из S.

Пусть $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ — конечное множество векторов. Если

$$\sum_{i=1}^{k} a_i \mathbf{x}_i = \mathbf{0} \Longrightarrow a_1 = a_2 = \cdots = a_k = 0,$$

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

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

то множество X называется *линейно зависимым*.

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

Доказательство

Пусть $S = \{\mathbf{0}, \mathbf{x_2}, \ldots, \mathbf{x_k}\}$, $\mathbf{x_1} := \mathbf{0}$. Положим $a_1 \quad :1$, $a_2 := 0$, $\quad a_k := 0$.

Тогда $\sum_{i=1}^{n} a_i \mathbf{x_i} = 10 + 0\mathbf{x_2} + \cdots + 0\mathbf{x_k} = \mathbf{0} + \mathbf{0} + \quad + \mathbf{0} = \mathbf{0}$. □

2.5.4. 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 $\mathcal{V}$. Конечное линейно независимое порождающее множество называется *базисом* векторного пространства.

Пусть векторное пространство $\mathcal{V}$ имеет базис $B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$.

ТЕОРЕМА *Каждый элемент векторного пространства имеет единственное представление в данном базисе.*

Доказательство

Пусть $\mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{e_i}$ и $\mathbf{x} = \sum_{i=1}^{n} b_i \mathbf{e_i}$.

Тогда $\mathbf{0} = \mathbf{x} - \mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{e_i} - \sum_{i=1}^{n} b_i \mathbf{e_i} = \sum_{i=1}^{n} (a_i - b_i)\mathbf{e_i} \Longrightarrow$

$\forall i \in 1..n \; a_i - b_i = 0 \Longrightarrow \forall i \; a_i = b_i$. □

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

Доказательство

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

$$\exists a_1, \ldots, a_n \in F \; \mathbf{x_1} = a_1\mathbf{e_1} + \quad + a_n\mathbf{e_n}$$

Имеем $\mathbf{x_1} \neq \mathbf{0} \Longrightarrow \bigvee_{i=1}^{n} a_i \neq 0$. Пусть для определенности $a_1 \neq 0$. Тогда (свойства поля)

$$\mathbf{e_1} = a_1^{-1}\mathbf{x_1} - (a_1^{-1}a_2)\mathbf{e_2} - (a_1^{-1}a_n)\mathbf{e_n}.$$

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,

$$\mathbf{x}_{n+1} = \sum_{i=1}^{n} b_i \mathbf{x_i} \;\&\; \bigvee_{i=1}^{n} b_i \neq 0 \Longrightarrow \mathbf{x}_{n+1} - \sum_{i=1}^{n} b_i \mathbf{x_i} = \mathbf{0} \Longrightarrow$$

X — линейно зависимое множество. □

ТЕОРЕМА *Пусть B_1 и B_2 — базисы векторного пространства $\mathcal{V}$, тогда*

$$|B_1| = |B_2|.$$

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 $|B_1| = |B_2|$. □

Мощность любого базиса векторного пространства $\mathcal{V}$ называется *размерностью* векторного пространства и обозначается $\dim(\mathcal{V})$. Векторное пространство, имеющее базис, называется *конечномерным*.

Пример

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

2.6. Решетки

Решетки иногда называют «структурами», но слово «структура» перегружено, и мы не будем использовать его в этом значении. Решетки сами по себе часто встречаются в разных программистских задачах, но еще важнее то, что понятие решетки непосредственно подводит нас к понятию булевой алгебры, которое имеет множество приложений в программировании и вычислительной технике.

2.6.1. Определения

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

1. идемпотентность:
   $a \cap a = a$, $a \cup a = a$;
2. коммутативность:
   $a \cap b = b \cap a$ $a \cup b = b \cup a$;
3. ассоциативность:
   $(a \cap b) \cap c = a \cap (b \cap c)$, $(a \cup b) \cup c = a \cup (b \cup c)$;
4. поглощение:
   $(a \cap b) \cup a = a$, $(a \cup b) \cap a = a$;
5. Решетка называется *дистрибутивной*, если
   $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$, $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$.

2.6.2. Ограниченные решетки

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

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

Доказательство

Пусть $0'$ — еще один нуль решетки. Тогда $0 \cap 0' \quad 0'$ и $0' \cap 0 = 0$.
Следовательно $0 = 0'$. □

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

Доказательство

$\Longrightarrow$: Пусть $a \cap b = b$. Тогда $a \cup b = a \cup (a \cap b) = (a \cap b) \cup a = a$.

$\Longleftarrow$: Пусть $a \cup b = a$. Тогда $a \cap b = (a \cup b) \cap b = (b \cup a) \cap b = b$. □

СЛЕДСТВИЕ $0 \cap a = 0 \iff 0 \cup a = a$, $1 \cup a = 1 \iff 1 \cap a = a$.

2.6.3. Решетка с дополнением

В ограниченной решетке 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 решетка называется *решеткой с дополнением*. Вообще говоря, дополнение не обязано существовать и не обязано быть единственным.

ТЕОРЕМА (о свойствах дополнения) *В ограниченной дистрибутивной решетке 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: $(a \cup b)' = a' \cap b'$, $(a \cap b)' = a' \cup b'$.*

Доказательство

1. Пусть x, y — дополнения a. Тогда $a \cap x = 0$, $a \cup x = 1$, $a \cap y = 0$, $a \cup y = 1$. Имеем: $(x = x \cap 1 = x \cap (a \cup y) = (x \cap a) \cup (x \cap y) = 0 \cup (x \cap y) = x \cap y$, $y = y \cap 1 = y \cap (a \cup x) = (y \cap a) \cup (y \cap x) = 0 \cup (y \cap x) = y \cap x = x \cap y) \Longrightarrow x = y$.
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. $(a \cap b) \cap (a' \cup b') = (a \cap b \cap a') \cup (a \cap b \cap b') = (0 \cap b) \cup (a \cap 0) = 0 \cup 0 = 0$, $(a \cap b) \cup (a' \cup b') = (a \cup a' \cup b') \cap (b' \cup a' \cup b') = (1 \cup b') \cap (1 \cup a') = 1 \cap 1 = 1$. □

2.6.4. Частичный порядок в решетке

В любой решетке можно естественным образом ввести нестрогий частичный порядок, а именно: $a \prec b := a \cap b = a$.

ТЕОРЕМА *Пусть $a \prec b := a \cap b = a$. Тогда $\prec$ является отношением частичного порядка.*

Доказательство

1. Рефлексивность: $a \cap a = a \Longrightarrow a \prec a$.
2. Антисимметричность: $a \prec b \,\&\, b \prec a \Longrightarrow a \cap b = a \,\&\, b \cap a = b \Longrightarrow a = a \cap b = b \cap a = b$.
3. Транзитивность: $a \prec b \,\&\, b \prec c \Longrightarrow a \cap b = a \,\&\, b \cap c = b \Longrightarrow a \cap c = (a \cap b) \cap c = a \cap (b \cap c) = a \cap b = a \Longrightarrow a \prec c$. □

Наличие частичного порядка в решетке не случайно, это ее характеристическое свойство. Более того, обычно решетку определяют, начиная с частичного порядка, следующим образом.

Пусть M — частично упорядоченное множество с частичным порядком $\prec$. 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 $v \prec x$. 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 $y \prec u$. Обозначение: $y = \sup(a, b)$.

ТЕОРЕМА *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 *Если в частично упорядоченном множестве для любых двух элементов существуют нижняя и верхняя грани, то это множество образует решетку относительно* inf *и* sup *(то есть $x \cap y := \inf(x, y)$, $x \cup y := \sup(x, y)$).*

Доказательство

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. $\inf\bigl(x,\inf(y,z)\bigr)=\inf\bigl(\inf(x,y),z\bigr) \Longrightarrow x\cap(y\cap z)=(x\cap y)\cap z$,
   $\sup\bigl(x,\sup(y,z)\bigr)=\sup\bigl(\sup(x,y),z\bigr) \Longrightarrow x\cup(y\cup z)=(x\cup y)\cup z$;
4. $\sup\bigl(\inf(a,b),a\bigr)=a \Longrightarrow (a\cap b)\cup a=a$,
   $\inf\bigl(\sup(a,b),a\bigr)=a \Longrightarrow (a\cup b)\cap a=a$. □

2.6.5. Булевы алгебры

Дистрибутивная ограниченная решетка, в которой для каждого элемента существует дополнение, называется *булевой алгеброй.*

Свойства булевой алгебры:

1. $a\cup a=a$, по определению решетки; $\quad a\cap a=a$
2. $a\cup b=b\cup a$, по определению решетки; $\quad a\cap b=b\cap a$
3. $a\cup(b\cup c)=(a\cup b)\cup c$, по определению решетки; $\quad a\cap(b\cap c)=(a\cap b)\cap c$
4. $(a\cap b)\cup a=a$, по определению решетки; $\quad (a\cup b)\cap a=a$
5. $a\cup(b\cap c)=(a\cup b)\cap(a\cup c)$, по свойству дистрибутивности; $\quad a\cap(b\cup c)=(a\cap b)\cup(a\cap c)$
6. $a\cup 1=1$, по свойству ограниченности; $\quad a\cap 0=0$
7. $a\cup 0=a$, по следствию из теоремы ограниченности; $\quad a\cap 1=a$
8. $a''=a$ по теореме о свойствах дополнения;
9. $(a\cap b)'=a'\cup b'$, по теореме о свойствах дополнения; $\quad (a\cup b)'=a'\cap b'$
10. $a\cup a'=1$, так как дополнение существует. $\quad a\cap a'=0$

Пример

1. $\langle 2^M;\cap,\cup,^-\rangle$ — булева алгебра, $1=U$, $0=\varnothing$, $\prec=\subset$.
2. $\langle E_2;\&,\vee,\neg\rangle$ — булева алгебра, $1=1$, $0=1$, $\prec=\Longrightarrow$.

3. Пусть $\pi(p) := \{q \mid q \in \mathbb{N} \,\&\, q - \text{простое} \,\&\, q \leqslant p\}$ — множество простых чисел, не превосходящих p,

$$P(p) := \left\{ n \mid n = \prod_{m \in M \subset \pi(p)} m \right\} -$$

множество произведений различных чисел из $\pi(p)$.

Тогда $\langle P; \text{Н.О.Д.}, \text{Н.О.К.}, \text{ДОП} \rangle$ — булева алгебра, где Н.О.Д. — наибольший общий делитель, Н.О.К. — наименьшее общее кратное,

$$\text{ДОП}(n) := \frac{1}{n} \prod_{q \in \pi(p)} q.$$

$1 := \prod_{q \in \pi(p)} q$, $0 := 1$, $m \prec n := n$ делится на m.

2.7. Матроиды

Матроиды, рассматриваемые в этом разделе, вообще говоря, не являются алгебраическими структурами в том смысле, который был придан этому понятию в первом разделе данной главы. Однако, во-первых, матроиды имеют много общего с рассмотренными алгебраическими структурами (в частности, с линейно независимыми множествами в векторных пространствах) и изучаются сходными методами, а во-вторых, они являются теоретической основой для изучения и анализа «жадных» алгоритмов. Ясное понимание природы и области применимости жадных алгоритмов совершенно необходимо всякому программисту.

2.7.1. Определения

Матроидом $M = \langle E, \mathcal{E} \rangle$ называется конечное множество E, $|E| = n$ и семейство его подмножеств $\mathcal{E} \subset 2^E$, такое 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: $A, B \in \mathcal{E} \,\&\, |B| = |A| + 1 \Longrightarrow \exists e \in B \setminus A \; A \cup \{e\} \in \mathcal{E}$.

Элементы множества $\mathcal{E}$ называются *независимыми*, а остальные подмножества E $(2^E \setminus \mathcal{E})$ — *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 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. Если бы все векторы из множества B выражались в виде линейной комбинации векторов из множества A, то множество B было бы линейно зависимым. 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 исследований линейной независимости в векторных пространствах.

2.7.2. Максимальные независимые подмножества

Пусть $X \subset E$ — произвольное множество. *Максимальным* независимым подмножеством множества X называется множество Y, такое что

$$Y \subset X \,\&\, Y \in \mathcal{E} \,\&\, \forall Z \in \mathcal{E}\ Z \subset X \Longrightarrow Z \subset Y.$$

Множество максимальных независимых подмножеств множества X обозначим $\underline{X}$.

Рассмотрим следующее утверждение:

M_4: $\forall X\ (Y \in \underline{X} \,\&\, Z \in \underline{X} \Longrightarrow |Y| = |Z|)$,

то есть 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*

$$(A, B \in \mathcal{E} \,\&\, |B| = |A| + 1 \Longrightarrow \exists e \in B \setminus A\ A \cup \{e\} \in \mathcal{E})$$

тогда и только тогда, когда

$$\forall X\ (Y, Z \in \underline{X} \Longrightarrow |Y| = |Z|).$$

Доказательство

Необходимость. 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 определенности $|Y| > |Z|$. Возьмем $Y' \subset Y$, так что $|Y'| = |Z| + 1$. Тогда по свойству M_3 имеем: $\exists e \in Y' \setminus Z\ W := Z \cup \{e\} \in \mathcal{E}$. Таким образом, имеем $W \in \mathcal{E}$, $Z \subset W$, $W \subseteq X$, что противоречит предположению $Z \in \underline{X}$.

Достаточность. 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 $\neg\exists e \in B \setminus A\ A \cup \{e\} \in \mathcal{E}$, то есть $\forall e \in B \setminus A\ A \cup \{e\} \notin \mathcal{E}$. 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.

2.7.3. Базисы

Максимальные независимые подмножества множества E называются *базами*, или *базисами*, матроида $M = \langle E, \mathcal{E} \rangle$. Всякий матроид имеет базисы, что видно из следующего алгоритма.

Алгоритм 2.1. Алгоритм построения 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

Алгоритм вычисляет базу матроида M.
Действительно, пусть $B_0 = \varnothing$, $B_1, \ldots, B_k = B$ — 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 $(B_{i-1} \cup \{e\}) \notin \mathcal{E}$. Но $e \in B'' \,\&\, B_{i-1} \subset B'' \Longrightarrow (B_{i-1} \cup \{e\}) \in \mathcal{E}$. По аксиоме M_2 имеем: $(B_{i-1} \cup \{e\}) \in \mathcal{E}$ — противоречие. □

СЛЕДСТВИЕ *Все базы матроида равномощны.*

2.7.4. Ранг

Мощность максимального независимого подмножества данного множества X называется *рангом* множества:

$$r(X) := \max_{A \subset X \,\&\, A \in \mathcal{E}} |A|.$$

ЗАМЕЧАНИЕ

Это определение корректно, потому что все максимальные независимые подмножества данного множества равномощны.

ЛЕММА $X \in \mathcal{E} \,\&\, Y \in \underline{X} \Longrightarrow X = Y$.

Доказательство

$X \subset X \,\&\, X \in \mathcal{E} \,\&\, Y \in \underline{X} \Longrightarrow X \subset Y \subset X \Longrightarrow X = Y$. □

ТЕОРЕМА $X \in \mathcal{E} \iff r(X) = |X|$.

Доказательство

$X \in \mathcal{E} \iff (X \in \mathcal{E} \,\&\, Y \in \underline{X} \Longrightarrow X = Y) \iff (r(x) = |X|)$. □

ТЕОРЕМА $\forall A, B \subseteq E \; \forall e_1, e_2 \in E$

$R_1 : 0 \leqslant r(A) \leqslant |A|$
$R_2 : A \leqslant B \Longrightarrow r(A) \leqslant r(B)$
$R_3 : r(A \cup B) + r(A \cap B) \leqslant r(A) + r(B)$
$R_4 : r(A) \leqslant r(A \cup \{e\}) \leqslant r(A) + 1$
$R_5 : r(A \cup \{e_1\}) = r(A \cup \{e_2\}) = r(A) \Longrightarrow r(A \cup \{e_1, e_2\}) = r(A)$

Доказательство

R_1 : очевидно;
R_2 : очевидно;
R_3 : Пусть $\{e_1, \ldots, e_i\} \in \underline{A \cap B}$. Тогда $\{e_1, \ldots, e_i, d_1, \ldots, d_j\} \in \underline{A}$ и затем $\{e_1, \ldots, e_i, d_1, \ldots, d_j, c_1, \ldots, c_k\} \in \underline{A \cap B}$. Имеем $i = r(A \cap B)$, $B^{i+j} = r(A)$, $i + k \leqslant i + j + k = r(A \cup B) \Longrightarrow r(A \cup B) + r(A \cap B) = (i + j + k) + i = (i + j) + (i + k) \leqslant r(A) + r(B)$;
R_4 : очевидно;
R_5 : $r(A) \leqslant r\big(A \cup \{e_1, e_2\}\big) = r\big(A \cup \{e_1\} \cup \{e_2\}\big) \leqslant r\big(A \cup \{e_1\}\big) + r\big(A \cup \{e_2\}\big) - r\Big(\big(A \cup \{e_1\}\big) \cap \big(A \cup \{e_2\}\big)\Big) = r(A) + r(A) - r(A) = r(A)$. □

2.7.5. Жадный алгоритм

Пусть имеются конечное множество E, $|E| = n$, *весовая* функция $w\colon E \to \mathbb{R}_+$ и семейство $\mathcal{E} \subset 2^E$.

Рассмотрим следующую задачу: найти $X \in \mathcal{E}$, такое что

$$w(X) = \max_{Y \in \mathcal{E}} w(Y), \quad \text{где } w(Z) := \sum_{e \in Z \subset E} w(e).$$

Другими словами, необходимо выбрать в указанном семействе подмножество наибольшего веса.

Не ограничивая общности, можно считать, что $w(e_1) \geqslant \cdots \geqslant w(e_n) > 0$.

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 упорядочено в порядке убывания весов элементов.
Выход: подмножество X.

```
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 $X \cup \{e_i\} \in \mathcal{E}$.) Возникает вопрос: в каких случаях жадный алгоритм действительно решает задачу, поставленную в начале раздела?

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; если же $M = \langle E, \mathcal{E} \rangle$ не является матроидом, то существует такая функция w, что множество X, найденное жадным алгоритмом, не будет максимальным.*

Доказательство

Пусть $M = \langle E, \mathcal{E} \rangle$ — матроид, и пусть $X = \{x_1, \ldots, x_k\}$ — множество, построенное жадным алгоритмом. По построению $w(x_1) \geqslant \cdots \geqslant w(x_k) > 0$. Согласно алгоритму 2.1, X — база M. Пусть теперь $Y = \{y_1, \ldots, y_m\} \in \mathcal{E}$ — некоторое независимое множество. Имеем $m \leqslant k$, так как X — база. Покажем, что $w(y_i) \leqslant w(x_i)$.

От противного. Пусть $w(y_i) > w(x_i)$. 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

$$w(y_j) \geqslant w(y_i) > w(x_i) \Longrightarrow \exists p \leqslant i \; w(x_1) \geqslant \cdots \geqslant w(x_{p-1}) \geqslant w(y_j) \geqslant w(x_p),$$

что противоречит тому, что x_p — элемент с наибольшим весом, добавление которого к $\{x_1, \ldots, x_{p-1}\}$ не нарушает независимости. Следовательно,

$$\forall i \; w(y_i) \leqslant w(x_i) \Longrightarrow w(Y) \leqslant w(X)$$

Обратно. 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 $A \notin \mathcal{E} \Longrightarrow A \not\subset X \Longrightarrow |X| < |A|$. Если же условие M_2 выполнено, но нарушено условие M_3, то $\exists A, B \; |A| = k \,\&\, |B| = k + 1$ и $\forall e \in B \setminus A \; A \cup \{e\}$ — 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 $B \setminus A$. В результате будет выбрано множество X, вес которого меньше веса множества B. Действительно,

$$w(X) = w(A) = k(1 + \varepsilon) = (k - p)(1 + \varepsilon) + p(1 + \varepsilon) \leqslant$$
$$\leqslant (k - p)(1 + 1/(k - p)) + p(1 + \varepsilon) = (k - p + 1) + p(1 + \varepsilon) = w(B). \quad \square$$

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 еще более эффективного алгоритма. С другой стороны, если семейство $\mathcal{E}$ не является матроидом, то это еще 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 удается 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 труднорешаемой. 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.

2.7.6. 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, $r(A) = |A|$.
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 $\mathcal{E} := \{A \subseteq E \mid |A \cap E_i| \leqslant 1\}$, 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 $A \cup \{e\} \in \mathcal{E}$. 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 2.5.4.
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. [14]).

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 2.3–2.6 описаны в [10, 12, 22]. Книги [22] и [12] могут быть использованы как введение в общую алгебру (разделы 2.1–2.2); первая из них вполне элементарна. Монография [3] исчерпывающим образом освещает важные для приложений вопросы, бегло затронутые в подразделе 2.6.5. Особого внимания заслуживает материал последнего раздела главы, который (с некоторыми сокращениями) следует книге [14].

Упражнения

2.1. Является ли группа целых чисел $\langle \mathbb{Z}; + \rangle$ конечно-порожденной? Какова ее система образующих?

2.2. Пусть $\mathcal{A} := \langle A; f \rangle$ и $\mathcal{B} := \langle B; g \rangle$ — две алгебры типа (2).
Тогда алгебра $\mathcal{C} := \langle C; h \rangle$ называется *прямым произведением* алгебр $\mathcal{A}$ и $\mathcal{B}$ (и обозначается $\mathcal{C} := \mathcal{A} \times \mathcal{B}$), если $C = A \times B$ и операция $h\colon C \times C \to C$ определена следующим образом:

$$h((a_1, b_1)(a_2, b_2)) := (f(a_1, a_2), g(b_1, b_2)).$$

Пусть $\mathcal{B} := \mathcal{A} \times \mathcal{A}$. Доказать, что существуют гомоморфизмы $\alpha\colon \mathcal{A} \to \mathcal{B}$ и $\beta\colon \mathcal{B} \to \mathcal{A}$, такие что $\alpha \circ \beta\colon A \to A$ — тождественная функция.

2.3. Пусть $L := \{ax + b \mid a, b \in \mathbb{R}\}$ — множество линейных функций. Операция *линейной замены переменной* определяется следующим образом:

$$(ax + b) \circ (cx + d) := (a(cx + d) + b) = (ac)x + (ad + b)).$$

Доказать, что множество линейных функций образует группу относительно линейной замены переменной.

2.4. Пусть на множестве пар вещественных чисел $\mathbb{R}^2$ определены операции

$$\oplus, \otimes\colon \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$$

следующим образом: $(a, b) \oplus (c, d) := (a + c, b + d)$ и $(a, b) \otimes (c, d) := (ac, bd)$. Доказать, что $\mathcal{K} := \langle \mathbb{R}^2; \oplus, \otimes \rangle$ образует коммутативное кольцо.

2.5. Поле $\mathcal{F} := \langle F; +, \cdot \rangle$ называется *упорядоченным*, если существует такое подмножество P носителя F, что для любого элемента $x \in F$ выполняется одно и только одно из следующих трех утверждений: $x \in P \setminus \{0\}$, $x = 0$ или $-x \in P \setminus \{0\}$. Определим отношение $\leqslant$ на упорядоченном поле следующим образом: $a \leqslant b := b - a \in P$. Доказать, что $\leqslant$ является отношением порядка на F.

2.6. Доказать, что в решетке из взаимного поглощения следует идемпотентность обеих операций.

2.7. Доказать, что семейство $\mathcal{B} \subset 2^E$ является базой некоторого матроида над E тогда и только тогда, когда

$$\forall B_1, B_2 \in \mathcal{B}\ e \in B_1 \setminus B_2 \Longrightarrow \exists f \in B_2 \setminus B_1\ (B_1 \setminus \{e\} \cup \{f\}) \in \mathcal{E}.$$

ГЛАВА 3 Булевы функции

Данная глава имеет двойное назначение. С одной стороны, это развернутый пример к предыдущей главе в том смысле, что здесь демонстрируются эффективность и действенность алгебраических методов. 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», которые должны 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 (sin, log и т. д.), 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 $E_2 := \{0,1\}$, называются *функциями алгебры логики*, или *булевыми функциями*, по имени Дж. Буля[1]. Множество булевых функций от n переменных обозначим P_n, $P_n := \{f \mid f\colon E_2^n \to E_2\}$.

Булеву функцию от n переменных можно задать *таблицей истинности*:

[1] G. Boole (1815–1864)

x_1,	...,	x_{n-1},	x_n	$f(x_1, \ldots, x_n)$
0	...	0	0	
0	...	0	1	
0	...	1	0	
...	...	...	...	
1	...	1	1	

Если число переменных n, то в таблице истинности имеется 2^n строк, соответствующих всем различным комбинациям значений переменных, которым можно сопоставить 2^{2^n} различных столбцов, соответствующих различным функциям. Таким образом, число булевых функций от n переменных с ростом n растет весьма быстро:

$$|P_n| = 2^{2^n}$$

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

$$f(a_1, \quad , a_{i-1}, 0, a_{i+1}, \quad , a_n) \neq f(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \quad , a_n).$$

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 следующей 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	

Для этих функций переменная x_1 — существенная, а переменная x_2 несущественная.

По определению булевы функции равны, если одна из другой получается введением (или удалением) несущественных переменных. 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.

3.1.3. Булевы функции одной переменной

	Переменная x	0	1	
Название	Обозначение			Фиктивные
нуль	0	0	0	x
тождественная	x	0	1	
отрицание	$\neg x$, $\bar{x}$, x'	1	0	
единица	1	1	1	x

3.1.4. Булевы функции двух переменных

	Переменная x	0	0	1	1	
	Переменная y	0	1	0	1	
Название	Обозначение					Фиктивные
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	$+$, $\not\equiv$, $\oplus$, $\triangle$	0	1	1	0	
дизъюнкция	$\vee$	0	1	1	1	
стрелка Пирса	$\downarrow$	1	0	0	0	
эквивалентность	$\sim$, $\equiv$	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	$\mid$	1	1	1	0	
единица	1	1	1	1	1	x,y

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 учета некоторых тонкостей, связанных с данными понятиями, которые целесообразно явно указать и обсудить.

3.2.1. Реализация функций формулами

Пусть $F = \{f_1, \ldots, f_m\}$ — множество булевых функций. *Формулой* над F называется выражение 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*.

ЗАМЕЧАНИЕ

Обычно для элементарных булевых функций используется инфиксная форма записи, устанавливается приоритет ($\neg$, &, $\vee$, $\to$) и лишние скобки опускаются.

Всякой формуле $\mathcal{F}$ однозначно соответствует некоторая функция f. Это соответствие задается алгоритмом *интерпретации*, который позволяет вычислить значение формулы при заданных значениях переменных.

Алгоритм 3.1. Алгоритм интерпретации формул — рекурсивная функция Eval

Вход: формула $\mathcal{F}$, множество функций базиса F, значения переменных $x_1, \ldots, x_n$.

Выход: значение формулы $\mathcal{F}$ на значениях $x_1, \ldots, x_n$, или значение **fail**, если значение 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 }
  end for
  return f(y_1, ..., y_n) { значение главной операции, примененной к значениям аргументов }
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 определен (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.
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* неопределенным, 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.

Если формула $\mathcal{F}$ и базис F заданы (причем $\mathcal{F}$ является правильно построенной формулой над базисом F), то процедура $\mathrm{Eval}(\mathcal{F}, F, x_1, \ldots, x_n)$ является некоторой булевой функций f переменных $x_1, \ldots, x_n$. В этом случае говорят, что формула $\mathcal{F}$ *реализует* функцию f:

$$\mathrm{func}\,\mathcal{F} = f.$$

ЗАМЕЧАНИЕ

Для обозначения реализуемости применяют и другие приемы. Выбранное обозначение обладает тем достоинством, что не путается с другими обозначениями в книге.

Зная таблицы истинности для функций базиса, можно вычислить таблицу истинности той функции, которую реализует данная формула.

Пример

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*:

$$\mathcal{F}_1 = \mathcal{F}_2 := \operatorname{func}\mathcal{F}_1 = f \,\&\, \operatorname{func}\mathcal{F}_2 = f.$$

Отношение равносильности формул является эквивалентностью. Имеют место 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.

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. Если подстановка производится вместо *всех* вхождений заменяемой переменной (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: $\mathcal{F}(\ldots\mathcal{G}_1\ldots)\{\mathcal{G}_2/\mathcal{G}_1\}$.

Пример

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: $x \vee y \vee z\{\neg x/y \vee z\} = x \vee \neg x$.

Правило подстановки: если в равносильных формулах вместо *всех* вхождений некоторой переменной x подставить одну и ту же формулу, то получатся равносильные формулы:

$$\forall \mathcal{G}\ (\mathcal{F}_1(\ldots x\ldots) = \mathcal{F}_2(\ldots x\ldots) \Longrightarrow \mathcal{F}_1(\ldots x\ldots)\{\mathcal{G}//x\} = \mathcal{F}_2(\ldots x\ldots)\{\mathcal{G}//x\}).$$

ЗАМЕЧАНИЕ

В правиле подстановки условие замены *всех* вхождений существенно: например, $x \vee \neg x = 1$ и $x \vee \neg x = 1\{y//x\} = y \vee \neg y = 1$, но $x \vee \neg x\{y/x\} = y \vee \neg x \neq 1$!

Правило замены: если в формуле заменить некоторую подформулу на равносильную, то получится равносильная формула:

$$\forall \mathcal{F}(\ldots\mathcal{G}_1\ldots)\ \mathcal{G}_1 = \mathcal{G}_2 \Longrightarrow \mathcal{F}(\ldots\mathcal{G}_1\ldots) = \mathcal{F}(\ldots\mathcal{G}_1\ldots)\{\mathcal{G}_2/\mathcal{G}_1\}.$$

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) являются операциями 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:

$$\operatorname{func}\mathcal{F}_1 = f, \qquad \operatorname{func}\mathcal{F}_2 = f, \qquad \operatorname{func}\mathcal{G}_1 = g, \qquad \operatorname{func}\mathcal{G}_2 = g.$$

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

$$f \wedge g := \operatorname{func}(\mathcal{F} \wedge \mathcal{G}), \qquad f \vee g := \operatorname{func}(\mathcal{F} \vee \mathcal{G}), \qquad \neg f := \operatorname{func}(\neg\mathcal{F})$$

Алгебраическая структура $\langle P_n; \vee, \wedge, \neg\rangle$ называется *алгеброй булевых функций*. Алгебра булевых функций является булевой алгеброй. 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. Таким образом, 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 $\mathcal{K} := \{[\mathcal{F}]\}_{\mathcal{F}}$. 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:

$$[\mathcal{F}_1] \vee [\mathcal{F}_2] := [\mathcal{F}_1 \vee \mathcal{F}_2], \quad [\mathcal{F}_1] \wedge [\mathcal{F}_2] := [\mathcal{F}_1 \wedge \mathcal{F}_2], \quad \neg[\mathcal{F}_1] := [\neg\mathcal{F}_2]$$

Тогда алгебра классов равносильных формул $\langle \mathcal{K}; \wedge, \vee, \neg\rangle$ (*алгебра Линденбаума-Тарского*) 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 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.

3.3. Принцип двойственности

Пусть $f(x_1, \ldots, x_n) \in P_n$ — булева функция. Тогда функция $f^*(x_1, \ldots, x_n)$, определенная следующим образом:

$$f^*(x_1, \ldots, x_n) := \overline{f(\overline{x}_1, \ldots, \overline{x}_n)},$$

называется *двойственной* к функции f. Из определения видно, что двойственность инволютивна: $f^{**} = f$.

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.

ЗАМЕЧАНИЕ

Далее используется обозначение: $\overline{f}(\cdots) := \overline{f(\cdots)}$.

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

$$\begin{aligned}\varphi^*(x_1, \ldots, x_n) &= \overline{\varphi}(\overline{x}_1, \ldots, \overline{x}_n) = \overline{f}(f_1(\overline{x}_1, \ldots, \overline{x}_n), \ldots, f_n(\overline{x}_1, \ldots, \overline{x}_n)) = \\ &= \overline{f}(\overline{f}_1(\overline{x}_1, \ldots, \overline{x}_n), \ldots, (\overline{f}_n(\overline{x}_1, \ldots, \overline{x}_n)) = \\ &= \overline{f}({f_1}^*(x_1, \ldots, x_n), \ldots, {f_n}^*(x_1, \ldots, x_n)) = \\ &= f^*(f_1^*(x_1, \ldots, x_n), \ldots, f_n^*(x_1, \ldots, x_n)). \quad \square\end{aligned}$$

ТЕОРЕМА (Принцип двойственности) *Пусть $F = \{f_1, \ldots, f_m\}$. Положим $F^* := \{{f^*}_1, \ldots, {f^*}_m\}$. Тогда если формула $\mathcal{F}$ над базисом F реализует функцию f, то формула $\mathcal{F}^*$ над базисом F^*, полученная из формулы $\mathcal{F}$ заменой функций f_i на двойственные функции ${f_i}^*$, реализует функцию f^*:*

$$\operatorname{func}\mathcal{F}[F] = f \Longrightarrow \operatorname{func}\mathcal{F}^*[F^*] = f^*, \text{ где } \mathcal{F}^*[F^*] := \mathcal{F}[F]\{f_i^* // f_i\}_{i=1}^m.$$

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. $\square$

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. Например, в доказательстве предыдущей теоремы был использован принцип математической индукции в следующей форме. Пусть задана некоторая иерархия (ориентированное дерево). Тогда если

1. некоторое утверждение P справедливо для всех узлов иерархии нижнего уровня (листьев дерева) и
2. из того, что утверждение P справедливо для всех узлов, подчиненных данному узлу, следует, что утверждение P справедливо для данного узла,

то 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 $x^y := x \cdot y \vee \overline{x} \cdot \overline{y}$ (здесь $\cdot$ обозначает конъюнкцию). Очевидно, что

$$x^y = \begin{cases} \overline{x}, & y = 0, \\ x, & y = 1, \end{cases} \qquad x^y = \begin{cases} 1, & x = y, \\ 0, & x \neq y, \end{cases} \qquad x^y = x \equiv y.$$

ТЕОРЕМА (О разложении булевой функции по переменным)

$$f(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) = \bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \wedge \cdots \wedge x_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n),$$

где дизъюнкция берется по всем возможным наборам $(\sigma_1, \ldots, \sigma_m)$.

Доказательство

$$\Big(\bigvee_{(\sigma_1,\ldots,\sigma_n)} x_1^{\sigma_1}\wedge\cdots\wedge x_m^{\sigma_m}\wedge f(\sigma_1,\ldots,\sigma_m,x_{m+1},\ldots,x_n)\Big)(a_1,\ldots,a_n)=$$

$$=\bigvee_{(\sigma_1,\ldots,\sigma_n)} a_1^{\sigma_1}\wedge\cdots\wedge a_m^{\sigma_m}\wedge f(\sigma_1,\ldots,\sigma_m,a_{m+1},\ldots,a_n)=$$

$$=a_1^{a_1}\wedge\cdots\wedge a_n^{a_n}\wedge f(a_1,\ldots,a_m,a_{m+1},\ldots,a_n)=f(a_1,\ldots,a_n).\quad\square$$

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

$$f(x_1,\ldots,x_{n-1},x_n)=x_n\wedge f(x_1,\ldots,x_{n-1},1)\vee\bar{x}_n\wedge f(x_1,\ldots,x_{n-1},0).$$

СЛЕДСТВИЕ $f(x_1,\ldots,x_n)=\bigvee\limits_{f(\sigma_1,\ldots,\sigma_n)=1} x_1^{\sigma_1}\wedge\cdots\wedge x_n^{\sigma_n}\wedge f(\sigma_1,\ldots,\sigma_n).$

3.4.2. Совершенные нормальные формы

Представление булевой функции $f(x_1,\ldots,x_n)$ в виде

$$\bigvee x_1^{\sigma_1}\wedge\cdots\wedge x_n^{\sigma_n}$$

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 в следующем отступлении.

ТЕОРЕМА *Всякая булева функция (кроме 0) имеет единственную СДНФ.*

Доказательство

$$f(x_1,\ldots,x_n)=\bigvee_{\sigma_1,\ldots,\sigma_n} x_1^{\sigma_1}\wedge\cdots\wedge x_n^{\sigma_n}\wedge f(\sigma_1,\ldots,\sigma_n)=$$

$$=\bigvee_{f(\sigma_1,\ldots,\sigma_n)=1} x_1^{\sigma_1}\wedge\cdots\wedge x_n^{\sigma_n}\wedge f(\sigma_1,\ldots,\sigma_n)=$$

$$=\bigvee_{f(\sigma_1,\ldots,\sigma_n)=1} x_1^{\sigma_1}\wedge\cdots\wedge x_n^{\sigma_n}.\quad\square$$

ТЕОРЕМА *Всякая булева функция может быть выражена через дизъюнкцию, конъюнкцию и отрицание:*

$$\forall f \in P_n \; \exists \mathcal{F}[\{\vee, \wedge, \neg\}] \; f = \operatorname{func} \mathcal{F}.$$

Доказательство

Если $f = 0$, то $0 = x \wedge \overline{x}$. Если $f \neq 0$, то см. предыдущую теорему. □

ТЕОРЕМА *Всякая булева функция (кроме 1) может быть единственным образом выражена в виде совершенной конъюнктивной нормальной формы (СКНФ):*

$$f(x_1, \ldots, x_n) = \bigwedge_{f^*(\sigma_1, \ldots, \sigma_n)=1} x_1^{\sigma_1} \vee \ldots \vee x_n^{\sigma_n}.$$

Доказательство

По принципу двойственности из предыдущей теоремы. □

ОТСТУПЛЕНИЕ

Говорят, что некоторый класс формул $\mathcal{K}$ имеет *нормальную форму*, если существует другой класс формул $\mathcal{K}'$, которые называются нормальными формами, такой что любая формула класса $\mathcal{K}$ имеет единственную равносильную формулу из класса $\mathcal{K}'$. Наличие у класса формул нормальной формы очень редкое, сильное и полезное свойство. Оно обеспечивает *разрешимость*, то есть наличие алгоритма проверки равносильности. Один и тот же класс $\mathcal{K}$ может иметь несколько различных нормальных форм, то есть несколько различных классов $\mathcal{K}'$.

3.4.3. Построение СДНФ

СДНФ булевой функции может быть построена по заданной таблице истинности с помощью следующего алгоритма.

Алгоритм 3.2. Построение СДНФ

Вход: вектор X : **array** [1..n] **of string** идентификаторов переменных,
матрица V : **array** [1..2^n, 1..n] **of** 0..1 всех различных наборов значений переменных,
вектор F : **array** [1..2^n] **of** 0..1 соответствующих значений функции.

Выход: последовательность символов, образующих запись формулы СДНФ для заданной функции.

```
f := false { признак присутствия левого операнда дизъюнкции }
for i from 1 to 2^n do
  if F[i] = 1 then
    if f then
      yield '∨' { добавление в формулу знака дизъюнкции }
    else
      f := true
    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
      end if
      if V[i, j] = 0 then
        yield '¬' { добавление в формулу знака отрицания }
      end if
      yield X[j] { добавление в формулу идентификатора переменной }
    end for
  end if
end for
```

ЗАМЕЧАНИЕ

Если зафиксировать порядок перечисления переменных в таблице истинности и порядок перечисления кортежей значений, то алгоритм построения СДНФ дает синтаксически однозначный результат. В этой книге переменные всегда перечисляются в лексикографическом порядке, а кортежи булевских значений — в порядке возрастания целых чисел, задаваемых кортежами как двоичными шкалами (см. алгоритм 1.1). Такой порядок далее считается *установленным*.

3.4.4. 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 **array** $[1..k, 1..n]$ **of** 0..1, где строка $f[i, *]$ содержит набор значений $\sigma_1, \ldots, \sigma_n$, для которого $f(\sigma_1, \ldots, \sigma_n) = 1$, $i \in 1..k$, $k \leqslant n$.

ОТСТУПЛЕНИЕ

Быстрое вычисление значения СДНФ имеет не только теоретическое, но и 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, причем клетки одного столбца считаются соединенными 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 3.3. Алгоритм вычисления 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} = 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 = 0 }
```

ЗАМЕЧАНИЕ

В алгоритме использован оператор **next**, которому здесь придается следующая семантика: выполнение текущего цикла прерывается, а выполнение программы продолжается со следующего шага цикла, указанного в операторе **next**. Такого рода операторы называются *операторами структурного перехода*. Операторы структурного перехода присутствуют в некоторых реальных языках программирования (например оператор **continue** в языке C), хотя обычно имеют более ограниченную семантику по сравнению с использованным здесь оператором **next**.

Этот алгоритм в худшем случае выполняет $k \cdot n$ сравнений, а в среднем — гораздо меньше, то есть он существенно эффективнее общего алгоритма интерпретации.

3.4.5. Эквивалентные преобразования

Используя уже доказанные равносильности, можно преобразовывать по правилу замены одни формулы в другие, равносильные им. Преобразование формулы в равносильную называются *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, 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 $\mathcal{F}_2$ с помощью равносильностей, указанных в подразделе 3.2.2.*

Доказательство

Любую формулу (кроме той, которая реализует 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 на подформулу, содержащую только три базисные операции. Например, элиминация импликации выполняется с помощью равносильности $x_1 \to x_2 = \neg x_1 \vee x_2$. В результате этого 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 этого шага отрицания могут присутствовать в формуле только непосредственно перед переменными.

3. *Раскрытие скобок.* По дистибутивности конъюнкции относительно дизъюнкции раскрываются все скобки, являющиеся операндами конъюнкции. В результате этого шага формула приобретает вид *дизъюнктивной формы*:

$$\bigvee (A_i \wedge \quad \wedge A_j),$$

где A_k — это либо переменная, либо отрицание переменной.

4. *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 этого шага формула не содержит «лишних» переменных и «лишних» конъюнктивных слагаемых.

5. *Расщепление переменных.* По правилу расщепления в каждую конъюнкцию, которая содержит не все переменные, добавляются недостающие. В результате этого шага формула становится «совершенной», то есть в каждой конъюнкции содержатся все переменные.

6. *Сортировка.* С помощью коммутативности переменные в каждой конъюнкции, а затем конъюнкции в дизъюнкции сортируются в установленном порядке (см. подраздел 3.4.3). В результате этого шага формула приобретает вид СДНФ.

Заметим, что указанные преобразования обратимы. Таким образом, если 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. □

3.5. Замкнутые классы

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 и следующий разделы посвящены ответу на вопрос: существуют ли (и если существуют, то какие) другие системы булевых функций, обладающих тем свойством, что с их помощью можно выразить все другие функции.

3.5.1. Замыкание множества булевых функций

Пусть $F = \{f_1, \ldots, f_m\}$, $f_i \in P_n$. *Замыканием* F (обозначается $[F]$) называется множество всех булевых функций, реализуемых формулами над F:

$$[F] := \{f \in P_n \mid f = \operatorname{func} \mathfrak{F}[F]\}.$$

Свойства замыкания:

1. $F \subset [F]$;
2. $[[F]] = [F]$;
3. $F_1 \subset F_2 \Longrightarrow [F_1] \subset [F_2]$;
4. $([F] \cup [F_2]) \subset [F_1 \cup F_2]$.

3.5.2. Некоторые замкнутые классы

Класс (множество) функций F называется *замкнутым*, если $[F] = F$.

Рассмотрим следующие классы функций:

1. Класс функций, сохраняющих 0:
   $T_0 := \{f \mid f(0, \ldots, 0) = 0\}$.
2. Класс функций, сохраняющих 1:
   $T_1 := \{f \mid f(1, \ldots, 1) = 1\}$.
3. Класс самодвойственных функций:
   $T_* := \{f \mid f = f^*\}$.
4. Класс монотонных функций:
   $T_\leqslant := \{f \mid \alpha \leqslant \beta \Longrightarrow f(\alpha) \leqslant f(\beta)\}$,
   где $\alpha = (a_1, \ldots, a_n)$, $\beta = (b_1, \ldots, b_n)$, $a_i, b_i \in E_2$, $\alpha \leqslant \beta := \forall i\ a_i \leqslant b_i$.
5. Класс линейных функций:
   $T_L := \{f \mid f = c_0 + c_1 x_1 + \cdots + c_n x_n\}$,
   где $+$ обозначает сложение по модулю 2, а знак конъюнкции опущен.

ТЕОРЕМА *Классы T_0, T_1, T_*, $T_\leqslant$, T_L замкнуты.*

Доказательство

Чтобы доказать, что некоторый класс F — замкнутый, 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. подраздел 3.3). База индукции очевидна: функции из F реализованы как тривиальные формулы над F. Таким образом, осталось обосновать индукционные переходы для пяти рассматриваемых классов.

1. 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, $\Phi \in T_0$.
2. 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, $\Phi \in T_1$.
3. 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, $\Phi \in T_*$.
4. 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, $\Phi \in T_\leqslant$.
5. 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
   $$\begin{aligned}f &= c_0 + c_1 x_1 + \cdots + c_n x_n, \\ f_1 &= c_0^1 + c_1^1 x_1 + \cdots + c_n^1 x_n, \\ &\vdots \\ f_n &= c_0^n + c_1^n x_1 + \cdots + c_n^n x_n.\end{aligned}$$
   Подставим эти формулы в формулу для Φ. Имеем:
   $$\begin{aligned}\Phi(x_1, \ldots, x_n) &= c_0 + c_1(c_0^1 + c_1^1 x_1 + \cdots + c_n^1 x_n) + \cdots + c_n(c_0^n + c_1^n x_1 + \cdots + c_n^n) = \\ &= d_0 + d_1 x_1 + \cdots + d_n x_n.\end{aligned}$$
   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 попарно различны, не пусты и не совпадают с P_n.

3.6. Полнота

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:*

$$F = \{f_1, \ldots, f_m\} \text{ и } G = \{g_1, \ldots, g_k\}.$$

Тогда, если система F полна и все функции из F реализуемы формулами над G, то система G также полна:

$$([F] = P_n \,\&\, \forall i\ f_i = \operatorname{func} \mathcal{G}_i[G]) \Longrightarrow [G] = P_n.$$

Доказательство

Пусть h — произвольная функция, $h \in P_n$.
Тогда $[F] = P_n \Longrightarrow h = \operatorname{func} \mathcal{F}[F] \Longrightarrow \mathcal{F}\{\mathcal{G}_i//f_i\}$ — формула над G. Следовательно, $h = \operatorname{func} \mathcal{G}[G]$. □

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*. 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 Σ — сложение по модулю 2, знак $\cdot$ обозначает конъюнкцию и $a_{i_1, \ldots, i_s} \in E_2$.

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:*

$$[F] = P_n \Longleftrightarrow \neg(F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L).$$

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 3.5.2.

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

$$\exists F' = \{f_0, f_1, f_*, f_\leqslant, f_L\}\ f_0 \notin T_0 \,\&\, f_1 \notin T_1 \,\&\, f_* \notin T_* \,\&\, f_\leqslant \notin T_\leqslant \,\&\, f_L \notin T_L.$$

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. 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.

1. 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: $\varphi(1) = 1$ и $\varphi(1) = 1$.

1) $\varphi(1) = 1$. В этом случае формула φ реализует 1.

2) $\varphi(1) = 0$. В этом случае формула φ реализует отрицание. Тогда рассмотрим функцию f_*. Имеем:

$$f_* \notin T_* \Longrightarrow \exists a_1, \ldots, a_n\ f_*(a_1, \ldots, a_n) \neq \overline{f}_*(\overline{a}_1, \ldots, \overline{a}_n).$$

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 $\psi = 1 \vee \psi = 0$. Если $\psi = 1$, то требуемая константа 1 построена. В противном случае ψ реализует 0, и значит $\varphi(\psi(x)) = \psi(x)$ реализует 1.

Построение 0 аналогично, только вместо f_0 нужно использовать f_1.

2. 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: $\varphi(0) > \varphi(1) \Longrightarrow \varphi(0) = 1 \,\&\, \varphi(1) = 0 \Longrightarrow \varphi(x) = \overline{x}$.

3. Построим формулу, реализующую конъюнкцию. Рассмотрим функцию f_L. Имеем: $f_L \in P_n \Longrightarrow f_L = \sum_{i_1, \ldots, i_s} a_{a_{i_1}, \ldots, a_{i_s}} x_{i1}, \ldots, x_{is}$. Но $f_L \notin T_L$, следовательно, в полиноме Жегалкина существует нелинейное слагаемое, содержащее конъюнкцию по крайней мере двух переменных. Пусть, для определенности, это x_1 и x_2. Тогда

$$f_L = x_1 \cdot x_2 \cdot f_a(x_3, \ldots, x_n) + x_1 \cdot f_b(x_3, \ldots, x_n) + \\ + x_2 \cdot f_c(x_3, \ldots, x_n) + f_d(x_3, \ldots, x_n),$$

причем $f_a(x_3, \ldots, x_n) \neq 0$. 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

$$\psi(x_1, x_2) = (x_1 + c) \cdot (x_2 + b) + b \cdot (x_1 + c) + c \cdot (x_2 + b) + d + b \cdot c + d = \\ = x_1 \cdot x_2 + c \cdot x_2 + b \cdot x_1 + b \cdot c + b \cdot x_1 + b \cdot c + c \cdot x_2 + b \cdot c + c \cdot c + \\ + d + b \cdot c + d = x_1 \cdot x_2.$$

(Функции $x + a$ выразимы, так как $x + 1 = \overline{x}$, $x + 0 = x$, а константы 0, 1 и отрицание уже построены.) □

Комментарии

Прекрасным руководством по булевым функциям являются книги [25] и [17], в которых можно найти обширный дополнительный материал, в частности, опущенные за недостатком места различные алгоритмы построения, упрощения и минимизации нормальных форм.

Упражнения

3.1. Доказать, что число булевых функций от n переменных, среди которых k фиктивных, равно $2^{2^{n-k}}$.

3.2. Проверить равносильности подраздела 3.2.2 путем построения таблиц истинности.

3.3. Какие функции являются двойственными для $+$, $\equiv$, $|$, $\downarrow$?

3.4. Построить СДНФ для $x_1 \mid x_2$, $x_1 \downarrow x_2$, $x_1 \to x_2$, $x_1 + x_2$.

3.5. Проверить принадлежность классам T_0, T_1, T_*, $T_\leqslant$, T_L функций $\downarrow$, $|$, $\vee$, $\to$, $\oplus$.

3.6. Доказать, что $f \notin T_0 \Longrightarrow f \notin T_* \vee (f \notin T_1 \,\&\, f \notin T_\leqslant)$.

ГЛАВА 4 Логические исчисления

С древнейших времен человечеству известна логика, или искусство правильно рассуждать. Вообще, 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. Он рассмотрел конкретные виды рассуждений, которые назвал *силлогизмами*. А именно, Аристотель рассмотрел так называемые *категорические утверждения* четырех видов:

- все A обладают свойством B (все A суть B);
- некоторые A обладают свойством B (некоторые A суть B);
- все A не обладают свойством B (все A суть не B);
- 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.

Пример

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 женщины раскрашивают свои лица. Следовательно, некоторые современные женщины — 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.

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 его 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 — ввести специфическую «логическую» терминологию и указать на ее связь с материалом предшествующих глав.

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*, принимающими *истинностные значения* «И» и «Л». Из простых высказываний с помощью *логических связок* могут быть построены составные высказывания. Обычно рассматривают следующие логические связки:

Название	Прочтение	Обозначение
Отрицание	не	
Конъюнкция	и	
Дизъюнкция	или	
Импликация	если ... то	

4.1.2. Формулы

Правильно построенные составные высказывания называются *(пропозициональными) формулами*. Формулы имеют следующий синтаксис:

⟨формула⟩ ::= И | Л |
⟨пропозициональная переменная⟩ |
(¬⟨формула⟩) |
(⟨формула⟩ &⟨формула⟩) |
(⟨формула⟩ ∨ ⟨формула⟩) |
(⟨формула⟩ → ⟨формула⟩)

Для упрощения записи вводится старшинство связок (¬, &, ∨, →) и лишние скобки опускаются.

Истинностное значение формулы определяется через истинностные значения ее составляющих в соответствии со следующей таблицей:

A	B	$\neg A$	$A \& B$	$A \vee B$	$A \to B$
Л	Л	И	Л	Л	И
Л	И	И	Л	И	И
И	Л	Л	Л	И	Л
И	И	Л	И	И	И

4.1.3. Интерпретация

Пусть $A(x_1, \ldots, x_n)$ — пропозициональная формула, где $x_1, \ldots, x_n$ — входящие в нее пропозициональные переменные. Конкретный набор истинностных значений, приписанных переменным $x_1, \ldots, x_n$, называется *интерпретацией* формулы A. Формула может быть истинной (иметь значение И) при одной интерпретации и ложной (иметь значение Л) 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*).

Пример

$A \vee \neg A$ — тавтология, $A \& \neg A$ — противоречие, $A \to \neg A$ — выполнимая формула, она истинна при $I(A) = \text{Л}$.

ТЕОРЕМА *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.*

Доказательство

Очевидно из определений. □

ТЕОРЕМА *Если формулы A и $A \to B$ — тавтологии, то формула B — тавтология.*

Доказательство

От противного. Пусть $I(B) = \text{Л}$. Но $I(A) = \text{И}$, так как A — тавтология. Значит, $I(A \to B) = \text{Л}$, 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 (обозначается $A \Longrightarrow B$), если формула B имеет значение И при всех интерпретациях, при которых формула A имеет значение И.

Говорят, что формулы A и B *логически эквивалентны* (обозначается $A \Longleftrightarrow B$ или просто $A = B$), если они являются логическим следствием друг друга. Логически эквивалентные формулы имеют одинаковые значения при любой интерпретации.

ТЕОРЕМА $(P \to Q) \Longleftrightarrow (\neg P \vee Q)$.

Доказательство

Для доказательства достаточно проверить, что формулы действительно имеют одинаковые истинностные значения при всех интерпретациях.

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$
И	И	И	Л	И
Л	И	И	И	И
И	Л	Л	Л	Л
Л	Л	И	И	И

□

ТЕОРЕМА *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.	$A \vee Л = A$,	$A \,\&\, Л = Л$;
7.	$A \vee И = И$,	$A \,\&\, И = A$;
8.	$\neg(\neg A) = A$;	
9.	$\neg(A \,\&\, B) = \neg A \vee \neg B$,	$\neg(A \vee B) = \neg A \,\&\, \neg B$;
10.	$A \vee \neg A = И$,	$A \,\&\, \neg A = Л$.

Доказательство

Непосредственно проверяется построением таблиц истинности. □

ЗАМЕЧАНИЕ

Таким образом, алгебра $\langle \{И, Л\}; \vee, \&, \neg \rangle$ является булевой алгеброй, которая называется *алгеброй высказываний*.

ТЕОРЕМА *$P_1 \& \ldots \& P_n \Longrightarrow Q$ тогда и только тогда, когда $(P_1 \& \ldots \& P_n) \to Q$ — тавтология.*

Доказательство

Необходимость. Пусть $I(P_1 \& \ldots \& P_n) = $ И. Тогда

$$I(Q) = \text{И и } I(P_1 \& \ldots \& P_n \to Q) = \text{И}.$$

Пусть $I(P_1 \& \ldots \& P_n) = $ Л. Тогда $I(P_1 \& \ldots \& P_n \to Q) = $ И при любой интерпретации I. Таким образом, формула $P_1 \& \ldots \& P_n \to Q$ общезначима.

Достаточность. Пусть $I(P_1 \& \ldots \& P_n) = $ И. Тогда $I(Q) = $ И, иначе бы формула

$$P_1 \& \ldots \& P_n \to Q$$

не была бы тавтологией. Таким образом формула Q — логическое следствие формулы

$$P_1 \& \ldots \& P_n. \quad \square$$

ТЕОРЕМА *$P_1 \& \ldots \& P_n \Longrightarrow Q$ 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. Подстановки

Пусть A — некоторая формула, в которую входит переменная x (обозначается $A(\ldots x \ldots)$) или некоторая подформула B (обозначается $A(\ldots B \ldots)$), и пусть C — некоторая формула. Тогда

$$A(\ldots x \ldots)\{C//x\}$$

обозначает формулу, полученную из формулы A подстановкой формулы C вместо *всех* вхождений переменной x, а $A(\ldots B \ldots)\{C/B\}$ обозначает формулу, полученную из формулы A подстановкой формулы C вместо *некоторых* (в частности, вместо одного) вхождений подформулы B.

ТЕОРЕМА *Если $A(\ldots x \ldots)$ — тавтология и B — любая формула, то $A(\ldots x \ldots)\{B//x\}$ — тавтология.*

Доказательство

Пусть $C := A(\ldots x \ldots)\{B//x\}$. Пусть I — интерпретация C (она не содержит x). Пусть $I' := I \cup \{x := I(C)\}$. Тогда $I'(A) = I(C)$, но $I'(A) =$ И, следовательно $I(C) =$ И. □

ТЕОРЕМА *Если* $A(\ldots B \ldots)$ *и* $B = C$, *а* $D := A(\ldots B \ldots)\{C/B\}$, *то* $A = D$.

Доказательство

Пусть I — любая интерпретация. Тогда $I(B) = I(C)$, значит $I(A) = I(D)$. □

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 развернутых примеров таких специальных исчислений (они довольно объемны), но, тем не менее, такие примеры существуют.

4.2.1. Определение формальной теории

Формальная теория $\mathcal{T}$ — это:

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 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, как правило, оно задается с помощью конечного множества *схем* аксиом и правил порождения конкретных аксиом из схемы аксиом. Обычно 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 $\mathcal{R}$, как правило, 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:

$$\frac{F_1, \ldots, F_n}{G} R,$$

где формулы $F_1, \ldots, F_n$ называются *посылками*, а формула G — *заключением*.

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:

$$F_1, \ldots, F_n \vdash_{\mathcal{T}} G,$$

где формулы $F_1, \ldots, F_n$ называются *гипотезами* вывода. Если теория $\mathcal{T}$ подразумевается, то ее 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 $\Gamma, \Delta \vdash_{\mathcal{T}} G$, где Γ и Δ — любые множества формул (то есть при добавлении лишних гипотез выводимость сохраняется).

ЗАМЕЧАНИЕ

При изучении формальных теорий нужно различать теоремы формальной теории и теоремы о формальной теории, или *метатеоремы*. Это различие не всегда явно формализуется, 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

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 счете нас интересуют такие формальные теории, которые описывают какие-то реальные объекты и связи между ними. Речь идет 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 ее теорема не является противоречием. Таким образом, формальная теория пригодна для описания тех множеств (алгебраических систем), которые являются ее 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.

ОТСТУПЛЕНИЕ

Последние утверждения являются доказуемыми метатеоремами, доказательства которых опускаются из-за технических сложностей.

4.2.5. Полнота, независимость и разрешимость

Пусть множество M является моделью формальной теории $\mathcal{T}$. Формальная теория $\mathcal{T}$ называется *полной* (или *адекватной*), если каждому истинному высказыванию M соответствует теорема теории $\mathcal{T}$.

Если для множества (алгебраической системы) M существует формальная полная непротиворечивая теория $\mathcal{T}$, то M называется *аксиоматизируемым* (или *формализуемым*).

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 и B — *любые* формулы. Таким образом, множество аксиом теории $\mathcal{L}$ 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*.

При записи формул лишние скобки опускаются, если это не вызывает недоразумений. Другие связки вводятся определениями (а не аксиомами):

$$A \,\&\, B := \neg(A \to \neg B), \qquad A \vee B := \neg A \to B.$$

Любая формула, содержащая эти связки, рассматривается как синтаксическое сокращение собственной формулы теории $\mathcal{L}$.

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 в ЭВМ. В следующих трех разделах вводится определение исчисления высказываний, более удобное для реализации на ЭВМ.

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:

$$B := A(\ldots, x_i, \ldots)\{B_i//x_i\}_{i=1}^n.$$

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

$$C = A(\ldots, x_i, \ldots)\{X_i//x_i\}_{i=1}^n \,\&\, C = B(\ldots, x_i, \ldots)\{X_i//x_i\}_{i=1}^n.$$

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 определенным 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 (полагаем, 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. Алгоритм унификации — 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 остается 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. подраздел 4.3.1). Аксиомы — три конкретные формулы:

A_1: $(a \to (b \to a))$;
A_1: $((a \to (b \to c)) \to ((a \to b) \to (a \to c)))$;
A_1: $((\neg b \to \neg a) \to ((\neg b \to a) \to b))$.

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 $\mathcal{L}$ — это достаточно богатая формальная теория, в которой выводимы многие важные теоремы.

ЗАМЕЧАНИЕ

Выводимость формул в теории $\mathcal{L}$ доказывается путем 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 она является гипотезой или получена из схемы аксиом указанной подстановкой, или получена из предшествующих формул по указанному правилу вывода и т. д.).

ТЕОРЕМА $\vdash_{\mathcal{L}} A \to A$

Доказательство

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 $A \vdash_{\mathcal{L}} B \to A$

Доказательство

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) *Если* $\Gamma, A \vdash_{\mathcal{L}} B$, *то* $\Gamma \vdash_{\mathcal{L}} A \to B$ *и обратно.*

Доказательство

Туда. 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 раздела $\vdash_{\mathcal{L}} E_1 \to E_1$, 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, причем $i, j < k$ и $E_j = E_i \to E_k$. Для первых трех случаев имеем $\Gamma \vdash_{\mathcal{L}} A \to E_k$ аналогичным образом. Для четвертого 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 $\Gamma \vdash_{\mathcal{L}} A \to B$.

Обратно. 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 для более широкого класса теорий, чем $\mathcal{L}$.

СЛЕДСТВИЕ 1 *Если* $A \vdash_{\mathcal{L}} B$, *то* $\vdash_{\mathcal{L}} A \to B$ *и обратно.*

Доказательство

$\Gamma := \varnothing$ □

СЛЕДСТВИЕ 2 $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$.

Доказательство

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 $A \to (B \to C), B \vdash_{\mathcal{L}} A \to C$.

Доказательство

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. $A \to (B \to C), B, A \vdash_{\mathcal{L}} C$	1–5
7. $A \to (B \to C), B \vdash_{\mathcal{L}} A \to C$	по теореме дедукции. □

ЗАМЕЧАНИЕ

Это производное правило называется *правилом сечения.*

4.3.7. Некоторые теоремы теории $\mathcal{L}$

Множество теорем теории $\mathcal{L}$ 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 $\mathcal{L}$ выводимы следующие теоремы:*

а) $\vdash_{\mathcal{L}} \neg\neg A \to A$

б) $\vdash_{\mathcal{L}} A \to \neg\neg A$

в) $\vdash_{\mathcal{L}} \neg A \to (A \to B)$

г) $\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$

д) $\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$

е) $\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$

ж) $\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$

Доказательство

Доказательство теоремы *а*: $\vdash_{\mathcal{L}} \neg\neg A \to A$.

№	Утверждение	Обоснование
1.	$(\neg A \to \neg\neg A) \to ((\neg A \to \neg A) \to A)$	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; $\{\neg A \to \neg\neg A/A, \neg A \to \neg A/B, A/C\}$
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; $\{\neg\neg A/A, \neg A \to \neg\neg A/B, A/C\}$

Доказательство теоремы *б*: $\vdash_{\mathcal{L}} A \to \neg\neg A$.

№	Утверждение	Обоснование
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; $\{\neg\neg\neg A/B, A/A, \neg\neg A/C\}$

Доказательство теоремы $в$: $\vdash_{\mathcal{L}} \neg A \to (A \to B)$.

№	Утверждение	Обоснование
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

Доказательство теоремы $г$: $\vdash_{\mathcal{L}} \neg B \to \neg A) \to (A \to B)$.

№	Утверждение	Обоснование
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

Доказательство теоремы $д$: $\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$.

№	Утверждение	Обоснование
1.	$A \to B$	гипотеза
2.	$\neg\neg A \to A$	$а$
3.	$\neg\neg A \to B$	Сл. 2; $\{A/B, \neg\neg A/A, A/C\}$
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

Доказательство теоремы *в*: $\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$

№	Утверждение	Обоснование
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.	$\vdash_{\mathcal{L}} ((A \to B) \to B) \to (\neg B \to \neg(A \to B))$	∂; $\{A \to B/A\}$
8.	$\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$	Сл. 2; $\{((A \to B) \to B)/B, (\neg B \to \neg(A \to B))/C\}$

Доказательство теоремы *ж*: $\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$.

№	Утверждение	Обоснование
1.	$A \to B$	гипотеза
2.	$\neg A \to B$	гипотеза
3.	$(A \to B) \to (\neg B \to \neg A)$	∂
4.	$\neg B \to \neg A$	MP; 1, 3
5.	$(\neg A \to B) \to (\neg B \to \neg\neg A)$	∂; $\{\neg A/A\}$
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 теории $\mathcal{L}$

Пусть формула A содержит переменные $a_1, \ldots, a_n$, и пусть задана некоторая интерпретация I формулы A, то есть приписаны истинностные значения переменным $a_1, \ldots, a_n$. Обозначим

$$A'_i := \begin{cases} a_i, & \text{если } a_i = \text{И} \\ \neg a_i, & \text{если } a_i = \text{Л} \end{cases} \qquad A' := \begin{array}{ll} A, & \text{если } I(A) = \text{И} \\ \neg A, & \text{если } I(A) = \text{Л} \end{array}$$

в данной интерпретации

ЛЕММА $A'_1 \quad , A'_n \vdash_{\mathcal{L}} A'$.

Доказательство

Индукция по структуре формулы A.

1. Переменная. Пусть $A = a$. Тогда $a \vdash_{\mathfrak{L}} a$ и $\neg a \vdash_{\mathfrak{L}} \neg a$.
2. Отрицание. Пусть $A = \neg B$.
   - Пусть $I(B) =$ И. Тогда $I(A) =$ Л и $A' = \neg A = \neg\neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} B$. Но $\vdash_{\mathfrak{L}} B \to \neg\neg B$ по теореме 4.3.7, *б*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} \neg\neg B = A'$.
   - Пусть $I(B) =$ Л. Тогда $I(A) =$ И и $A' = A = \neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} \neg B = A'$.
3. Импликация. Пусть $A = (B \to C)$. По индукционному предположению
$$A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} B' \text{ и } A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} C'.$$
   - Пусть $I(B) =$ Л. Тогда, независимо от значения $I(C)$, имеем:
$$I(A) = \text{И и } B' = \neg B,\ A' = A.$$
   Но $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} \neg B, \vdash_{\mathfrak{L}} \neg B \to (B \to C)$ по теореме 4.3.7, *в*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} B \to C = A'$.
   - Пусть $I(B) =$ И и $I(C) =$ И. Тогда $I(A) =$ И и $C' = C$, $A' = A = B \to C$. Имеем: $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} C, \vdash_{\mathfrak{L}} C \to (B \to C)$ (аксиома A_1 с подстановкой $\{C/A\}$), следовательно, $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} B \to C = A'$.
   - Пусть $I(B) =$ И и $I(C) =$ Л. Тогда $A' = \neg A = \neg(B \to C)$, $B' = B$ и $C' = \neg C$. Имеем: $A'_1, \ldots, A'_n \vdash B$, $A'_1, \ldots, A'_n \vdash C$, $\vdash_{\mathfrak{L}} B \to (\neg C \to \neg(B \to C))$ по теореме 4.3.7, *е*. Следовательно, $A'_1, \ldots, A'_n \vdash_{\mathfrak{L}} \neg(B \to C) = A'$. □

ТЕОРЕМА *Теоремами теории $\mathfrak{L}$ являются общезначимые формулы и только они:*
$$\vdash_{\mathfrak{L}} A \iff A - \textit{тавтология}.$$

Доказательство

$\Longleftarrow$: Пусть A — тавтология. Тогда $A'_1, \ldots, A'_n \vdash A$ в любой интерпретации. Таким образом, имеется 2^n различных выводимостей $A'_1, \ldots, A'_n \vdash A$. Среди них есть две, которые различаются в A'_n: $A'_1, \ldots, A'_{n-1}, a_n \vdash A$ и $A'_1, \ldots, A'_{n-1}, \neg a_n \vdash A$. По теореме дедукции $A'_1, \ldots, A'_{n-1} \vdash_{\mathfrak{L}} a_n \to A$ и $A'_1, \ldots, A'_{n-1} \vdash \neg a_n \to A$, но $\vdash_{\mathfrak{L}} (a_n \to A) \to ((\neg a_n \to A) \to A)$ по теореме 4.3.7(g), следовательно, $A'_1, \ldots, A'_{n-1} \vdash A$. Повторив этот процесс еще $n-1$ раз, имеем $\vdash_{\mathfrak{L}} A$.

$\Longrightarrow$: Аксиомы A_1, A_2, A_3 суть тавтологии. MP сохраняет тавтологичность, следовательно, теоремы теории $\mathfrak{L}$ суть тавтологии. □

СЛЕДСТВИЕ *Теория $\mathfrak{L}$ формально непротиворечива.*

Доказательство

Все теоремы $\mathfrak{L}$ суть тавтологии. Отрицание тавтологии не есть тавтология. Следовательно, в $\mathfrak{L}$ нет теоремы и ее отрицания. □

4.3.9. Другие аксиоматизации исчисления высказываний

Теория $\mathfrak{L}$ не является единственной возможной аксиоматизацией исчисления высказываний. Ее основное достоинство — лаконичность при сохранении определенной 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 и Аккерман, 1938.
   Связки: $\vee, \neg, \quad (A \to B := \neg A \vee B)$.
   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, 1953.
   Связки: $\&, \neg, \quad (A \to B := \neg(A \& \neg B))$.
   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, 1952.
   Связки: $\neg, \&, \vee, \to$.
   Аксиомы:
   $A \to (B \to A)$,
   $(A \to (B \to C)) \to ((A \to B) \to .(A \to C))$,
   $A \& B \to A$,
   $A \& B \to B$,
   $A \to (B \to (A \& B))$,
   $A \to (A \vee B)$,
   $B \to (A \vee B)$,
   $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$,
   $(A \to B) \to ((A \to \neg B) \to \neg A)$,
   $\neg\neg A \to A$.
   Правило: Modus ponens.
4. Никод, 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.

1. Алфавит:

связки	основные	$\neg \rightarrow$
	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*	$P, Q, \ldots$
	функторы	$f, g, \ldots$

С каждым предикатом и функтором связано некоторое натуральное число, которое называется *арностью* , или *местностью*.

2. Формулы имеют следующий синтаксис:

⟨формула⟩	::=	⟨атом⟩ | ¬⟨формула⟩ | (⟨формула⟩ → ⟨формула⟩) | ∀⟨переменная⟩ ⟨формула⟩ | ∃⟨переменная⟩ ⟨формула⟩
⟨атом⟩	::=	⟨предикат⟩(⟨список термов⟩)
⟨список термов⟩	::=	⟨терм⟩ | ⟨терм⟩, ⟨список термов⟩
⟨терм⟩	::=	⟨константа⟩ | ⟨переменная⟩ | ⟨функтор⟩(⟨список термов⟩)

При этом должны быть выполнены следующие контекстные условия: в *терме* $f(t_1, \ldots, t_n)$ функтор 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 остаются свободными в формулах $\neg A$ и $A \rightarrow B$. 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, остаются 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. Формула, не содержащая свободных вхождений переменных, называется *замкнутой*.

Пример

Рассмотрим формулу $\forall x\ (P(x) \to \exists y\ Q(x,y))$ и ее подформулы. В подформулу $\exists y\ Q(x,y)$ переменная x входит свободно, а оба 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 вхождением 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 $\mathcal{L}$ не содержит кванторов, поэтому понятия свободного и связанного вхождения переменных не применимы непосредственно. Обычно для удобства полагают, что все формулы теории $\mathcal{L}$ 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 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

1. Терм y свободен для переменной x в формуле $P(x)$, но тот же терм y не свободен для переменной x в формуле $\forall y\ P(x)$.
2. 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: $\forall x\ A(x) \to A(t)$,
P_2: $A(t) \to \exists x\ A(x)$,

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, функторов, предикатов и собственных аксиом, называется *чистым*. Исчисление предикатов, которое содержит предметные константы и/или функторы и/или предикаты и связывающие их собственные аксиомы, называется *прикладным*.

Исчисление предикатов, в котором кванторы могут связывать только предметные переменные, но не могут связывать функторы или предикаты, называется исчислением *первого порядка*. Исчисления, в которых кванторы могут связывать не только предметные переменные, но и функторы, 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

- каждой предметной константе a элемент носителя $I(a)$, $I(a) \in M$;
- каждому n-местному функтору f операцию $I(f)$ на носителе, $I(f)\colon M^n \to M$;
- 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:

$$s^*(a) := I(a), \quad s^*(x_i) := s_i, \quad s^*(f(t_1, \dots, t_n)) := I(f)(s^*(t_1), \dots, s^*(t_n)).$$

Всякий атом $P(t_1, \dots, t_n)$ имеет на s истинностное значение $s^*(P)$, определяемое следующим образом:

$$s^*(P(t_1, \dots, t_n)) := (s^*(t_1), \dots, s^*(t_n)) \in I(P).$$

Если $s^*(P) = \text{И}$, 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 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 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 Г, если все формулы из Г 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 ее *замыкание* $\forall x\ \forall y\ \forall z \cdots A(x, y, z, \dots)$ истинно в данной интерпретации. 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^*.
Пусть $s^*(t_1) = a_1$ и пусть $t(x)$ — некоторый терм, а $t' := t(\dots x \dots)\{t_1//x\}$. Тогда $s^*(t) = s_1^*(t')$, где s_1 имеет значение a_1 на месте x.
Пусть $A(x)$ — формула, а терм t свободен для x в A. Положим

$$A(t) := A(\dots x \dots)\{t//x\}.$$

Имеем: $s^*(A(t)) = \text{И} \iff s_1^*(A(x)) = \text{И}$, где s_1 имеет значение $s^*(t)$ на месте x. Если $s^*(\forall x\ A(x)) = \text{И}$ и терм t свободен для x в A, то $s^*(A(t)) = \text{И}$. 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. Полнота чистого исчисления предикатов

Следующие две метатеоремы устанавливают свойства исчисления предикатов, аналогичные тем, которые установлены для исчисления высказываний в подразделе 4.3.8. Теоремы приводятся без доказательств, которые технически довольно сложны.

ТЕОРЕМА *Всякая теорема чистого исчисления предикатов первого порядка общезначима.*

ТЕОРЕМА *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. Имеют место следующие логические следования и эквавалентности:

1. $\neg\forall x\ A(x) = \exists x\ \neg A(x)$, $\quad \neg\exists x\ A(x) = \forall x\ \neg A(x)$,
2. $\forall x\ (A(x)\,\&\,B(x)) = \forall x\ A(x)\,\&\,\forall x\ B(x)$, $\quad \exists x\ (A(x)\vee B(x)) = \exists x\ A(x)\vee\exists x\ B(x)$,
3. $\exists x\ (A(x)\,\&\,B(x)) \Longrightarrow \exists x\ A(x)\,\&\,\exists x\ B(x)$, $\quad \forall x\ A(x)\vee\forall x\ B(x) \Longrightarrow \forall x\ (A(x)\vee B(x))$,
4. $\forall x\,\forall y\ A(x,y) = \forall y\,\forall x\ A(x,y)$, $\quad \exists x\,\exists y\ A(x,y) = \exists y\,\exists x\ A(x,y)$,
5. $\forall x\ (A(x)\,\&\,C) = \forall x\ A(x)\,\&\,C$, $\quad \forall x\ (A(x)\vee C) = \forall x\ A(x)\vee C$,
6. $\exists x\ (A(x)\,\&\,C) = \exists x\ A(x)\,\&\,C$, $\quad \exists x\ (A(x)\vee C) = \exists x\ A(x)\vee C$,
7. $C\to\forall x\ A(x) = \forall x\ (C\to A(x))$, $\quad C\to\exists x\ A(x) = \exists x\ (C\to A(x))$,
8. $\forall x\ A(x)\to C \Longrightarrow \exists x\ (A(x)\to C)$, $\quad \exists x\ A(x)\to C \Longrightarrow \forall x\ (A(x)\to C)$,

где формула C не содержит никаких вхождений переменной x.

Для всякой формулы A существует логически эквивалентная ей формула A' в *предваренной форме*:

$$A' := Q_1\,x_1\ldots Q_n\,x_n\ \tilde{A}(x_1,\ldots,x_n),$$

где $Q_1,\ldots,Q_n$ — некоторые кванторы, а $\tilde{A}$ — *бескванторная* формула.

4.4.6. Теория равенства

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. $\vdash_\varepsilon x = y \to (y = z \to x = z)$.

Доказательство

1. Из E_1 и P_1 по MP.
2. Имеем: $(x = y) \to (x = x \to y = x)$ из E_2 при подстановке $\{x = x/A(x)\}$. Значит, $x = y, x = x \vdash_\varepsilon y = x$. Но $\vdash_\varepsilon x = x$, следовательно, $x = y \vdash_\varepsilon y = x$. По теореме дедукции $\vdash_\varepsilon x = y \to y = x$.
3. Имеем: $y = x \to (y = z \to x = z)$ из E_2 при подстановке $\{y/x, x/y\}$, значит, $y = x \vdash_\varepsilon (y = z \to x = z)$. Но $x = y \vdash_\varepsilon y = x$, по транзитивности $x = y \vdash_\varepsilon (y = z \to x = z)$, по теореме дедукции $\vdash_\varepsilon (x = y) \to (y = z \to x = z)$. □

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 функторы $+$ и $\cdot$, одноместный функтор $'$
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.

4.4.8. Теория (абелевых) групп

Теория групп $\mathcal{G}$ — это прикладное исчисление предикатов *с равенством*, в котором имеются:

1. Предметная константа 0.
2. Двухместный функтор $+$.

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: $\forall x\ \exists y\ x + y = 0$.

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: $\forall x, y\ x + y = y + x$.

Абелева группа называется группой *конечного порядка* n, если выполнена собственная аксиома

G_5: $\forall x\ \exists k \leqslant n\ kx = 0$, где kx — это сокращение для $x + x + \cdots + x$ (k слагаемых).

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: $\forall x\ (x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0)$.

Абелева группа называется *полной*, если выполнена собственная аксиома

G_6: $\forall n \geqslant 1\ \forall x\ \exists y\ ny = x$.

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: $\forall x\ \exists y\ 2y = x$,
$\forall x\ \exists y\ 3y = x$,
...

Но любое *конечное* множество формул, истинное во всех полных абелевых группах, истинно и в некоторой неполной абелевой группе, то есть теория полных абелевых групп не является *конечно* аксиоматизируемой.

Абелева группа называется *периодической*, если выполнена собственная аксиома

G_7: $\forall x\ \exists n \geqslant 1\ nx = 0$.

Эта формула 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'_9: $\forall x\ (x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0 \vee \ldots)$, которая не является допустимой формулой исчисления предикатов и тем более теории $\mathcal{G}$. Таким образом, периодическая абелева группа 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. 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 функторами содержат «внелогические» предикаты и функторы, 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 функторов, которые и обеспечивают практическую применимость системы логического программирования.

4.4.9. Теоремы Гёделя о неполноте

В настоящее время точно 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 изложение этих фактов выходит далеко за рамки данного учебника. Поэтому ниже приводятся для сведения (без доказательства и в упрощенной 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 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.) Теорема Гёделя утверждает, что непротиворечивость достаточно богатой формальной теории не может быть установлена средствами самой этой теории (см. еще 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 популярного метода автоматического доказательства теорем.

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 сложных формальных теорий $\mathcal{T}$. В некоторых случаях удается построить алгоритм автоматического доказательства теорем, который применим не ко всем формулам теории (то есть частичный алгоритм, см. подраздел 4.2.5).

Для некоторых простых формальных теорий (например исчисление высказываний) и некоторых простых классов формул (например прикладное исчисление предикатов с одним одноместным предикатом) алгоритмы автоматического доказательства теорем известны.

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 И, то проверяемая формула — тавтология, и, следовательно, является теоремой теории $\mathcal{L}$. Если же хотя бы в одном случае получится значение Л, то проверяемая формула не является тавтологией и, следовательно, не является теоремой теории $\mathcal{L}$.

ЗАМЕЧАНИЕ

Приведенный выше пример является алгоритмом автоматического доказательства теорем в теории $\mathcal{L}$, хотя и не является алгоритмом автоматического поиска вывода теорем из аксиом теории $\mathcal{L}$.

Наиболее известный классический алгоритм автоматического доказательства теорем называется *методом резолюций*. Для любого прикладного исчисления предикатов первого порядка $\mathcal{T}$, любой формулы S и множества формул Γ теории $\mathcal{T}$ метод резолюций выдает ответ «Да», если $\Gamma \vdash_{\mathcal{T}} S$, и выдает ответ «Нет» или не выдает никакого ответа (то есть зацикливается), если неверно, что $\Gamma \vdash_{\mathcal{T}} S$.

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 $\mathcal{L}$.) $\Gamma, \neg S \vdash F \iff \Gamma \& \neg S \to F$ — 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 истинной ни в какой интерпретации и, по определению, является противоречием. В качестве формулы F при доказательстве от противного по методу резолюций принято использовать пустую формулу, которая обозначается □.

4.5.3. Сведение 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 $\Longrightarrow$ используется для обозначения способа преобразования формул).

1. *Элиминация импликации.* Преобразование: $A \to B \Longrightarrow \neg A \vee B$. После первого этапа формула содержит только $\neg, \vee, \&, \forall, \exists$.
2. *Протаскивание отрицаний.* Преобразования: $\neg \forall x\ A \Longrightarrow \exists x\ \neg A$, $\neg \exists x\ A \Longrightarrow \forall x\ \neg A$, $\neg\neg A \Longrightarrow A$, $\neg(A \vee B) \Longrightarrow \neg A \,\&\, \neg B$, $\neg(A \,\&\, B) \Longrightarrow \neg A \vee \neg B$. После второго этапа формула содержит отрицания только перед атомами.
3. *Разделение связанных переменных.* Преобразование:
$$Q_1 x\ A(\ldots Q_2 x\ B(\ldots x \ldots)\ldots) \Longrightarrow Q_1 x\ A(\ldots Q_2 y\ B(\ldots y \ldots)\ldots),$$
где Q_1 и Q_2 — любые кванторы. После третьего этапа формула не содержит случайно совпадающих связанных переменных.
4. *Приведение к предваренной форме.* Преобразования:
$$Q x\ A \vee B \Longrightarrow Q x\ (A \vee B), \quad Q x\ A \,\&\, B \Longrightarrow Q x\ (A \,\&\, B),$$
где Q — любой квантор. После четвертого этапа формула находится в предваренной форме.
5. *Элиминация кванторов существования (сколемизация).* Преобразования:
$$\exists x_1\ Q_2 x_2\ \ldots Q_n x_n\ A(x_1, x_2 \ldots, x_n) \Longrightarrow Q_2 x_2 \ldots Q_n x_n\ A(a, x_2, \ldots, x_n),$$
$$\forall x_1\ \ldots \forall x_{i-1}\ \exists x_i\ Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, x_i, \ldots, x_n) \Longrightarrow$$
$$\Longrightarrow \forall x_1\ \ldots \forall x_{i-1} Q_{i+1}\ x_{i+1} \ldots Q_n\ x_n A(x_1, \ldots, f(x_1, \ldots, x_{i-1}), \ldots, x_n),$$
где a — новая предметная константа, f — новый функтор, а $Q_1, Q_2, \ldots, Q_n$ — любые кванторы. После пятого этапа формула содержит только кванторы всеобщности.
6. *Элиминация кванторов всеобщности.* Преобразование: $\forall x\ A(x) \Longrightarrow A(x)$. После шестого этапа формула не содержит кванторов.
7. *Приведение к конъюнктивной нормальной форме.* Преобразование:
$$A \vee (B \,\&\, C) \Longrightarrow (A \vee B) \,\&\, (A \vee C), \quad (A \,\&\, B) \vee C \Longrightarrow (A \vee C) \,\&\, (B \vee C).$$
После седьмого этапа формула находится в конъюнктивной нормальной форме.
8. *Элиминация конъюнкции.* Преобразование: $A \,\&\, B \Longrightarrow A, B$. 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
$$F = \forall x_1 \ldots \forall x_{i-1}\ \exists x_i Q_{i+1} x_{i+1} \ldots Q_n x_n\ A(x_1, \ldots, x_n)$$
Положим
$$F' := \forall x_1\ \ldots \forall x_{i-1}\ Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, x_{i-1}, f(x_1, \ldots, x_{i-1}), x_{i+1}, \ldots, x_n).$$

Пусть F — противоречие, а F' — не противоречие. Тогда существует интерпретация I и набор значений $s = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, такой что $s^*(F') = \text{И}$. 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 $s_1^*(F) = \text{И}$ и F — выполнимая формула. □

ЗАМЕЧАНИЕ

Множество формул Г невыполнимо — это означает, что множество Г не имеет модели, то есть не существует интерпретации, в которой все формулы Г имели бы значение И.

4.5.4. Правило резолюции для исчисления высказываний

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.

Правило резолюции — это очень мощное правило вывода. Многие ранее рассмотренные правила являются частными случаями правила резолюции:

$$\frac{A,\ A \to B}{B}\ \text{Modus ponens} \qquad \frac{A,\ \neg A \vee B}{B}\,R$$

$$\frac{A \to B,\ B \to C}{A \to C}\ \text{Транзитивность} \qquad \frac{\neg A \vee B,\ \neg B \vee C}{\neg A \vee C}\,R$$

$$\frac{A \vee B,\ A \to B}{B}\ \text{Слияние} \qquad \frac{A \vee B,\ \neg A \vee B}{B}\,R$$

ТЕОРЕМА *Правило резолюции логично, то есть резольвента является логическим следствием резольвируемых предложений.*

Доказательство

Пусть $I(C_1) = \text{И}$ и $I(C_2) = \text{И}$. Тогда, если $I(P) = \text{И}$, то $C_2' \neq \varnothing$ и $I(C_2') = \text{И}$, а значит, $I(C_1' \vee C_2') = \text{И}$. Если же $I(P) = \text{Л}$, то $C_1' \neq \varnothing$ и $I(C_1') = \text{И}$, а значит, $I(C_1' \vee C_2') = \text{И}$. □

ЗАМЕЧАНИЕ

Резольвента является предложением.

4.5.5. Правило резолюции для исчисления предикатоа

Для применения правила резолюции нужны контрарные литералы в резольвируемых предложениях. 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 $C_2 = \neg P_2 \vee C_2'$, причем атомарные формулы P_1 и P_2 являются унифицируемыми 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 резолюций и резольвента добавляется в множество до тех пор, пока не будет получено пустое предложение. При этом возможны три случая:

1. Среди текущего множества предложений нет резольвируемых. 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 все 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. $\square$ 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: 1 — если G выводимо из S, 0 — в противном случае.

```
M := C { M — текущее множество предложений }
while □ ∉ M do
  Choose(M, c1, c2, p1, p2, σ) { выбор родительских предложений }
  if c1, c2 = ∅ then
    return 0 { нечего резольвировать }
  end if
  c := R(c1, c2, p1, p2, σ) { вычисление резольвенты }
  M := M ∪ {c} { пополнение текущего множества }
end while
return 1 { 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. 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

Здесь изложены элементарные сведения из математической логики, причем технически сложные доказательства опущены. Более подробное изложение можно найти в многочисленных классических учебниках, например [15]. Классическое описание автоматического доказательства теорем методом резолюций имеется в [24]. Эта книга содержит описание различных стратегий и усовершенствований метода резолюций, а также многочисленные примеры применений. Алгоритмы 4.3.3 и 4.5.7 являются упрощенными модификациями алгоритмов, описанных в [24].

Упражнения

4.1. Пусть $x \equiv y := x \to y \,\&\, y \to x$. Доказать, что формула $\mathcal{L}$, содержащая только связку $\equiv$, является тавтологией тогда и только тогда, когда каждая переменная входит в нее четное число раз.

4.2. Описать процедуру, которая перечисляет множество термов для заданных конечных множеств переменных, констант и функторов.

4.3. Пусть

- формула A теории $\mathcal{L}$ — не тавтология.
- $\tilde{A}$ — множество всех частных случаев A.
- $\mathcal{L}^+ := \mathcal{L} \cup \tilde{A}$.

Доказать, что $\mathcal{L}^+$ противоречива, то есть, 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 идет 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, причем каждая запись содержит одно предложение (что имеется) и один запрос (что требуется) относительно объектов недвижимости. Требуется найти все такие пары записей, в которых предложение первой записи совпадает с запросом второй, и, наоборот, 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, что мы оценили только трудоемкость подбора *прямых* вариантов, а существуют еще варианты, когда число участников сделки больше двух ...

Этот пример показывает, что комбинаторные вычисления требуют предварительного анализа и количественной оценки исходных задач и используемых алгоритмов. 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 *временную сложность*), то есть количество необходимых шагов алгоритма, и *сложность по памяти* (или *ёмкостную сложность*), то есть объем памяти, необходимый для работы алгоритма.

Во всех случаях основным инструментом такого анализа оказываются формулы и методы, рассматриваемые в этой главе.

5.1. Комбинаторные конфигурации

Во многих практических случаях возникает необходимость подсчитать количество возможных комбинаций объектов, удовлетворяющих определенным условиям. Такие задачи называются *комбинаторными*. Разнообразие комбинаторных задач не поддается исчерпывающему описанию, но среди них есть целый ряд особенно часто встречающихся, для которых известны способы подсчета.

5.1.1. Комбинаторные задачи

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

$$F\colon X \to Y, \quad \text{где } |X| = n,\ |Y| = m,\ X = \{1, \ldots, n\}.$$

Не ограничивая общности, можно считать, что

$$Y = \{1, \ldots, m\},\ F = \langle F(1), \ldots, F(n)\rangle,\ 1 \leqslant F(i) \leqslant m.$$

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 подсчета (вывод формулы) можно провести на любом языке. Если сведение одной модели к другой не очевидно, то оно включается в доказательство.

5.1.2. Размещени

Число всех функций (при отсутствии ограничений), или число всех возможных способов разместить n предметов по m ящикам называется, *числом размвщений* и обозначается $U(m, n)$.

ТЕОРЕМА $U(m,n)=m^n$

Доказательство

Каждый из n предметов можно разместить m способами. □

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 $(6+6)$ дает двенадцать очков. Вероятность $1/36$.

5.1.3. Размещения без повторений

Число инъективных функций, или число всех возможных 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,

$$A(m,n)=m\cdot(m-1)\cdot\ldots\cdot(m-n+1)=\frac{m!}{(m-n)!}.$$ □

По определению считают, что $A(m,n):=0$ при $n>m$ и $A(m,0):=1$

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

Число взаимнооднозначных функций, или *число перестановок* n предметов, обозначается $P(n)$.

ТЕОРЕМА $P(n) = n!$

Доказательство

$P(n) = A(n,n) = n \cdot (n-1) \cdot \quad (n-n+1) = n \cdot (n-1) \quad 1 = n!.$ □

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 монотонных 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

1. Число размещений без повторений нужно разделить на число перестановок, поскольку предметы неразличимы.
2. Число сочетаний является числом строго монотонных функций, потому что строго монотонная функция $F\colon 1..n \to 1..m$ определяется набором своих значений, причем $1 \leqslant F(1) < \cdots < F(n) \leqslant m$. Другими словами, каждая строго монотонная функция определяется выбором n чисел из диапазона $1..m$. Таким образом, число строго монотонных 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 $C(m,n) := 0$ при $n > m$.

Пример

В начале игры в домино каждому играющему выдается 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:

$$C(28,7) = \frac{28!}{7!(28-7)!} = \frac{28 \cdot 27 \cdot 26 \cdot 25 \cdot 24 \cdot 23 \cdot 22}{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 1184040.$$

5.1.6. Сочетания с повторениями

Число монотонных функций, или число размещений n неразличимых предметов по m ящикам, называется *числом сочетаний с повторениями* и обозначается $V(m,n)$.

ТЕОРЕМА $V(m,n) = C(n+m- \quad n)$

Доказательство

Монотонной функции $f\colon \{1,\ldots,n\} \to \{1,\ldots,m\}$ однозначно соответствует строго монотонная функция $f'\colon \{1,\ldots,n\} \to \{1,\ldots,n+m-1\}$. Это соответствие устанавливается следующей процедурой.

```
f'(1) := f(1) { первые элементы совпадают }
for i from 2 to n do
  if f(i) = f(i-1) then
    f'(i) := f'(i-1) + 1 { плато }
  else
    f'(i) := f'(i-1) + f(i) - f(i-1) { подъем }
  end if
end for
```

□

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, это можно сделать
$V(m,n) = C(m+n-1,n) = \frac{(m+n-1)!}{n!(m-1)!}$ способами.

5.2. Подстановки

В этом разделе рассматриваются подстановки и перестановки, которые на самом деле являются равнообъемными понятиями. Для вычисления количества

перестановок в подразделе 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 растет быстрее экспоненты. Действительно, используя известную из математического анализа *формулу Стирлинга*

$$n! \approx \sqrt{2\pi n} n^n e^{-n}$$

или, более точно

$$\sqrt{2\pi n} n^n e^{-n} < n! < \sqrt{2\pi n} n^n e^{-n+1/(12n)}$$

нетрудно показать, что

$$\lim_{n \to +\infty} \frac{n!}{2^n} = +\infty.$$

5.2.1. Группа подстановок

Взаимнооднозначная функция $f: X \to X$ называется *подстановкой* на X.

ЗАМЕЧАНИЕ

Если множество X конечно ($|X| = n$), то, не ограничивая общности, можно считать, что $X = 1..n$. В этом случае подстановку $f: 1..n \to 1..n$ удобно задавать таблицей из двух строк. В первой строке — значения аргументов, во второй — соответствующие значения функции.

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 подстановок f и g называется их суперпозиция $f \circ g$.

Пример

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

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

Пример

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

Обратная подстановка — это обратная функция, которая всегда существует, поскольку подстановка 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, множество подстановок образует группу относительно операции суперпозиции. Эта группа называется *симметрической* степени n.

5.2.2. Графическое представление подстановок

Подстановки удобно представлять в графической форме, проводя стрелки от каждого элемента x к элементу $f(x)$.

Пример

Графическое представление подстановки

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

представлено на рис. 5.1.

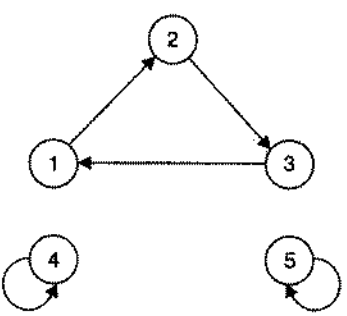


Рис. 5.1. Графическое представление подстановки

5.2.3. Циклы

Цикл — это 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 подстановки наглядно видно происхождение термина «цикл».

5.2.4. Подстановки и перестановки

В таблице подстановки нижняя строка (значения функции) является перестановкой элементов верхней строки (значения аргумента). Если приинять соглашение, что элементы верхней строки (аргументы) всегда располагаются в определенном порядке (например, по возрастанию), то верхнюю строку можно не указывать — подстановка определяется одной нижней строкой. Таким образом, подстановки взаимно однозначно соответствуют перестановкам.

Перестановку (и соответствующую ей подстановку) элементов $1,\ldots,n$ будем обозначать $\langle a_1,\ldots,a_n\rangle$, где все a_i — различные числа из диапазона $1..n$.

5.2.5. 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 подстановку 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 ее 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, $f\circ t_1\circ\cdots\circ t_n=e$ и по свойству группы $f=t_n^{-1}\circ\cdots\circ t_1^{-1}$, причем $|t_1|+|t_2|+\cdots+|t_n|=I(f)$. □

СЛЕДСТВИЕ *Всякая сортировка может быть выполнена перестановкой соседних элементов.*

ОТСТУПЛЕНИЕ

Доказанная теорема утверждает, что произвольную перестановку можно представить в виде композиции определенного 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 свое место транспозициями соседних элементов все элементы остаются на своих местах, кроме перемещаемого элемента и того элемента, который стоит на целевом месте, а эти элементы 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 1 to n − 1 do
  m := i { индекс кандидата в минимальные элементы }
  for j from i + 1 to n do
    if A[j] < A[m] then
      m := j { новый кандидат в минимальные }
    end if
  end for
  A[i] ↔ A[m] { ставим минимальный элемент на место }
end for
```

5.2.6. Генерация перестановок

На множестве перестановок естественным образом можно определить упорядоченность 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 1.9.3). 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]
  j := j + 1
  k := k − 1
end while
```

Обоснование

Заметим следующее. Искомую последовательность перестановок n элементов можно получить из последовательности перестановок $n-1$ элемента следующим образом. Нужно выписать n блоков по $(n-1)!$ перестановок в каждом, соответствующих последовательности перестановок $n-1$ элемента 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 $n = 3$: $(1,2,3)$, $(2,1,3)$, $(1,3,2)$, $(3,1,2)$, $(2,3,1)$, $(3,2,1)$.

5.3. Биномиальные коэффициенты

Число сочетаний $C(m,n)$ — это число различных n-элементных подмножеств m-элементного множества (см. подраздел 5.1.5). Числа $C(m,n)$ встречаются в формулах решения многих комбинаторных задач. Действительно, рассмотрим следующую типовую схему рассуждений при решении комбинаторной задачи. Пусть нужно определить число подмножеств m-элементного множества, удовлетворяющих некоторому условию. Разобьем задачу на подзадачи: рассмотрим отдельно 1-элементные подмножества, 2-элементные и т. д., а затем сложим полученные результаты. К счастью, числа $C(m,n)$ обладают целым рядом свойств, рассматриваемых в этом разделе, которые оказываются очень полезными при выкладках.

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. $$C(m-1,n) + C(m-1,n-1) = \frac{(m-1)!}{n!\,(m-n-1)!} + \frac{(m-1)!}{(n-1)!\,(m-1-(n-1))!} =$$
$$= \frac{(m-1)!}{n(n-1)!\,(m-n-1)!} + \frac{(m-1)!}{(n-1)!\,(m-n)(m-n-1)!} =$$
$$= \frac{(m-n)(m-1)! + n(m-1)!}{n(n-1)!\,(m-n)(m-n-1)!} = \frac{(m-n+n)(m-1)!}{n!\,(m-n)!} = \frac{m!}{n!\,(m-n)!} = C(m,n).$$

$$C(n,i)C(i,m) = \frac{n!}{i!(n-i)!} \cdot \frac{i!}{m!(i-m)!} = \frac{n!}{m!(i-m)!(n-i)!} =$$
$$= \frac{n!(n-m)!}{m!(i-m)!(n-i)!(n-m)!} =$$

$$= \frac{n!}{m!(n-m)!} \cdot \frac{(n-m)!}{(i-m)!(n-i)!} =$$
$$= C(n,m)C(n-m,i-m). \qquad \square$$

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*.

ТЕОРЕМА $(x+y)^m \quad \sum_{n=0}^{m} C(m,n)x^n y^{m-n}$

Доказательство

По индукции. База, $m =$

$$(x+y)^1 = x+y = 1x^1y^0 + 1x^0y^1 = C(1,0)x^1y^0 + C(1,1)x^0y^1 = \sum_{n=0}^{1} C(1,n)x^n y$$

Индукционный переход:

$$(x+y)^m = (x+y)(x+y)^{m-1} = (x+y)\sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n-1}$$
$$= \sum_{n=0}^{m-1} xC(m-1,n)x^n y^{m-n-1} + \sum_{n=0}^{m-1} yC(m-1,n)x^n y^{m-n-}$$
$$= \sum_{n=0}^{m-1} C(m-1,n)x^{n+1} y^{m-n-1} + \sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n}$$
$$= \sum_{n=0}^{m-1} \left(C(m-1,n-1) + C(m-1,n)\right)x^n y^{m-n} + C(m-1,m \quad 1)x^m y^0$$
$$= \sum_{n=0}^{m-1} C(m,n)x^n y^{m-n} + C(m,m)x^m y^{m-m} = \sum_{n=0}^{m} C(m,n)x^n y^{m-n}. \qquad \square$$

СЛЕДСТВИЕ $\sum_{n=0}^{m} C(m,n) = 2^m$

Доказательство

$$2^m = (1+1)^m = \sum_{n=0}^{n} C(m,n)1^n 1^{m-n} = \sum_{n=0}^{m} C(m,n). \qquad \square$$

СЛЕДСТВИЕ $\sum_{n=0}^{m}(-1)^n C(m,n)=0.$

Доказательство

$$0=(-1+1)^m=\sum_{n=0}^{m}C(m,n)(-1)^n 1^{m-n}=\sum_{n=0}^{m}(-1)^n C(m,n).$$ □

5.3.3. Свойства биномиальных коэффициентов

Биномиальные коэффициенты обладают целым рядом замечательных свойств.

ТЕОРЕМА

1. $\sum_{n=0}^{m} nC(m,n)=m2^{m-1}$,
2. $C(m+n,k)=\sum_{i=0}^{k}C(m,i)C(n,k-i)$.

Доказательство

1. Рассмотрим следующую последовательность из чисел $1,\ldots,m$. Сначала 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.
2. $C(n+m,k)$ — 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*.

5.3.4. Треугольник Паскаля

Из второй формулы теоремы 5.3.1 вытекает эффективный способ реккурентного вычисления значений биномиальных коэффициентов, который можно представить в графической форме, известной как *треугольник Паскаля*[1].

[1] Блез Паскаль (1623–1662)

- $999 : (5 * 7) = 28$ 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. Прииции включеиия и исключеиия

Следующая формула, известная как *принцип включения и исключения*, позволяет вычислить мощность объединения множеств, если известны их мощностн и мощности всех пересечений.

ТЕОРЕМА

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\leqslant i<j\leqslant n} |A_i \cap A_j| + \sum_{1\leqslant i<j<k\leqslant n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1}|A_1 \cap \cdots \cap A_n|.$$

Доказательство

Индукция по n. Для $n = 2, 3$ теорема проверена в предыдущем подразделе. Пусть

$$\left|\bigcup_{i=1}^{n-1} A_i\right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \cdots + (-1)^{n-2}|A_i \cap \cdots \cap A_{n-1}|.$$

Заметим, что $\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n = \bigcup_{i=1}^{n-1} (A_i \cap A_n)$, и по индукционному предположению

$$\left|\bigcup_{i=1}^{n-1} (A_i \cap A_n)\right| = \sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \ldots + (-1)^{n-2}|A_1 \cap \cdots \cap A_{n-1} \cap A_n|.$$

Тогда

$$\left|\bigcup_{i=1}^{n} A_i\right| = \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n\right| = \left|\bigcup_{i=1}^{n-1} A_i\right| + |A_n| - \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n\right| =$$

$$= \left(\sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \cdots + (-1)^{n-2}|A_1 \cap \cdots \cap A_{n-1}|\right) +$$

$$+ |A_n| - \left(\sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \ldots + (-1)^{n-2}|A_1 \cap \cdots \cap A_{n-1} \cap A_n|\right) =$$

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 на идущие подряд целые числа, 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 $m = 4$: $(1,2,3)$, $(1,2,4)$, $(1,3,4)$, $(2,3,4)$.

5.4. Разбиения

Разбиения не были рассмотрены среди типовых комбинаторных конфигураций в разделе 5.1, потому что получить для них явную формулу не так просто, как для остальных. В этом разделе исследуются основные свойства разбиений, а окончательные формулы приведены в подразделе 5.6.3.

5.4.1. Определения

Пусть $\mathcal{B} = \{B_1, \quad, B_n\}$ есть разбиение множества X из m элементов на n подмножеств:

$$B_i \subset X, \qquad \bigcup_{i=1} B_i = X, \qquad B_i \neq \varnothing, \qquad B_i \cap B_j = \varnothing \quad \text{при } i \quad j.$$

Подмножества B_i называются *блоками* разбиения.

Между разбиениями и отношениями эквивалентности существует взаимнооднозначное соответствие (см. подраздел 1.7.2). 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

$$S(m,0) := 0 \quad \text{при } m > 0,$$
$$S(m,m) := 1,$$
$$S(0,0) := 1,$$
$$S(m,n) := 0 \quad \text{при } n > m.$$

ТЕОРЕМА $S(m,n) \quad S(m \quad 1, n-1) + nS(m-1,n)$

Доказательство

Пусть $\mathcal{B}$ — множество всех разбиений множества $\{1, \quad m\}$ 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: берем 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 $S(m,n) = \sum_{i=n-1}^{m-1} C(m-1,i)S(i,n-1).$

Доказательство

Пусть $\mathcal{B}$ — множество всех разбиений множества $\{1,\ldots,m\}$ на n блоков. Рассмотрим семейство $\overline{B} := \left\{B \subset 2^{\{1,\ldots,m\}} \mid m \in B\right\}$. Тогда

$$\mathcal{B} = \bigcup_{B \in \overline{B}} \mathcal{B}_B,$$

где $\mathcal{B}_B := \{X \mid X \in \mathcal{B} \,\&\, B \in X\}$, причем $\mathcal{B}_{B'} \cap \mathcal{B}_{B>>} = \varnothing$, если $B' \neq B >>$. Пусть $B \in \overline{B}$ и $b := |B|$. Тогда $|\mathcal{B}_B| = S(m-b,n-1)$. Заметим, что

$$|\left\{B \in \overline{B} \mid |B| = b\right\}| = C(m-1,b-1).$$

[1] Джеймс Стирлинг (1699-1770)

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 $i := m - b$. □

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 $s(m,n)$.

Каждое разбиение множества $\{1,\ldots,m\}$ соответствует ядру сюръективной функции и обратно (см. подраздел 1.7.4). Таким образом, число различных ядер 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 ядром равно числу перестановок множества значений функции.

5.4.4. Число Белла

Число всех разбиений m-элементного множества называется *числом Белла* и обозначается $B(m)$.

$$B(m) := \sum_{n=0}^{m} S(m,n), \quad B(0) := 1.$$

ТЕОРЕМА $B(m+1) = \sum_{i=0}^{m} C(m,i) B(i)$.

Доказательство

Пусть $\mathcal{B}$ — множество всех разбиений множества $1..m+1$. Рассмотрим множество подмножеств множества $1..m+1$, содержащих элемент $m+1$:

$$\overline{B} := \left\{ B \subset 2^{\{1,\ldots,m+1\}} \mid m+1 \in B \right\}.$$

Тогда $\mathcal{B} = \bigcup_{B \in \overline{B}} \mathcal{B}_B$, 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 $|\{B \in \overline{B} \ \ |B| = b\}| = C(m,b \quad 1)$. 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. Принцип включения и исключения

Приведенные в предыдущих четырех разделах формулы и алгоритмы дают способы вычисления комбинаторных чисел для некоторых распространенных комбинаторных конфигураций. Практические задачи не всегда прямо сводятся к 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 сферу применения принципа включения и исключения.

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:

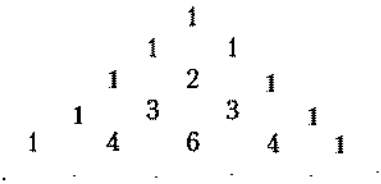
$$|A \cup B| = |A| + |B| - |A \cap B|,$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|.$$

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,



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.

5.3.5. Генерация подмножеств

Элементы множества $\{1,\ldots,m\}$ упорядочены. Поэтому каждое n-элементное подмножество также можно рассматривать как упорядоченную последовательность. На множестве таких последовательностей естественным образом 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
    yield A[1..n] { единственное подмножество }
    exit
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
```

$$= \left(\sum_{i=1}^{n-1}|A_i| + |A_n|\right) - \left(\sum_{1\leqslant i<j\leqslant n-1}|A_i\cap A_j| + \sum_{i=1}^{n-1}|A_i\cap A_n|\right) + \ldots$$

$$-(-1)^{n-2}|A_1\cap\cdots\cap A_{n-1}\cap A_n| =$$

$$= \sum_{i=1}^{n}|A_i| - \sum_{1\leqslant i<j\leqslant n}|A_i\cap A_j| + \cdots + (-1)^{n-1}|A_1\cap\cdots\cap A_n|.$$

□

5.5.3. Число булевых функций, существенно зависящих от всех своих переменных

Рассмотрим применение принципа включения и исключения на примере следующей задачи. Пусть $p_n := |P_n| = 2^{2^n}$ — число всех булевых функций n переменных, а $\widetilde{p}_n$ — число булевых функций, существенно зависящих от всех n переменных (см. подраздел 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:

$$\widetilde{p}_n = |P_n \setminus (P_n^1\cup\cdots\cup P_n^n)| = \left|P_n \setminus \bigcup_{n=1}^{n} P_n^i\right|.$$

С другой стороны, $|P_n^i| = 2^{2^{n-1}}$, более того, $|P_n^{i_1}\cap\cdots\cap P_n^{i_k}| = 2^{2^{n-k}}$. Следовательно,

$$\widetilde{p}_n = 2^{2^n} - \left(\sum_{i=1}^{n}|P_n^i| - \sum_{1\leqslant i<j\leqslant n}|P_n^i\cap P_n^j| + \cdots + (-1)^{n-1}|P_n^1\cap\cdots\cap P_n^n|\right) =$$

$$= 2^{2^n} - \left(C(n,1)2^{2^{n-1}} - C(n,2)2^{2^{n-2}} + \cdots + (-1)^{n-1}C(n,n)2\right) =$$

$$= \sum_{i=0}^{n}(-1)^i C(n,i)2^{2^{n-i}}.$$

5.6. Формулы обращения

Очень полезную, но специфическую группу приемов 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, причем $0 \leqslant k \leqslant n$. 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*

$$\forall n\ \forall k \leqslant n\ a_{n,k} = \sum_{i=0}^{n} \lambda_{n,k,i} b_{n,i}$$

и пусть

$$\exists \mu_{n,k,i}\ \forall k \leqslant n\ \forall m \leqslant n\ \sum_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m} = \begin{cases} 1, & m = k, \\ 0, & m \neq k. \end{cases}$$

Тогда

$$\forall k \leqslant n\ b_{n,k} = \sum_{i=0}^{n} \mu_{n,k,i} a_{n,i}.$$

Доказательство

$$\sum_{i=0}^{n} \mu_{n,k,i} a_{n,i} = \sum_{i=0}^{n} \mu_{n,k,i} \left(\sum_{m=0}^{n} \lambda_{n,i,m} b_{n,m} \right) =$$

$$= \sum_{m=0}^{n} \left(\sum_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m} \right) b_{n,m} = b_{n,k}.$$ □

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

$$\sum_{i=0}^{n} (-1)^{i-m} C(n,i) C(i,m) = \sum_{i=0}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) =$$

$$= \sum_{i=m}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) =$$

$$= C(n,m) \sum_{i=m}^{n} (-1)^{i-m} C(n-m, i-m).$$

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 *Если* $a_{n,k} = \sum_{i=0}^{k} C(k,i)b_{n,k}$, *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:

$$\begin{aligned}\sum_{i=0}^{n}\mu_{n,k,i}\lambda_{n,i,m} &= \sum_{i=0}^{n}(-1)^{k-i}C(k,i)C(i,m) = \\ &= \sum_{i=0}^{k}(-1)^{k-i+m-m}C(k,i)C(i,m) = \\ &= (-1)^{k-m}\sum_{i=0}^{k}(-1)^{i-m}C(k,i)C(i,m) = \\ &= \begin{cases}1, & k = m,\\ 0, & k \neq m.\end{cases}\end{aligned} \qquad \square$$

ТЕОРЕМА *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:

$$\begin{aligned}\sum_{i=0}^{n}\mu_{n,k,i}\lambda_{n,i,m} &= \sum_{i=0}^{n}(-1)^{i-k}C(i,k)C(m,i) = \\ &= \sum_{i=0}^{n}(-1)^{i-k}C(m,i)C(i,k) = \\ &= \begin{cases}1, & k = n,\\ 0, & k \neq n.\end{cases}\end{aligned} \qquad \square$$

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 $s(i,n)$,

поскольку $s(i,n)$ — это число сюръективных функций $f: 1..n \to 1..i$. Но область значений функции (при заданном i) можно выбрать $C(k,i)$ способами. Поэтому

$$k^n = \sum_{i=0}^{k} C(k,i)s(i,n).$$

Обозначив $a_{n,k} := k^n$ и $b_{n,i} := s(i,n)$, имеем по первой теореме предыдущего подраздела,

$$s(k,n) = \sum_{i=0}^{n} (-1)^{k-i} C(i,k) i^n.$$

Учитывая связь чисел Стирлинга первого и второго рода, имеем:

$$S(k,n) = \frac{1}{n!} \sum_{i=0}^{n} (-1)^{k-i} C(i,k) i^n.$$

5.7. Производящие функции

Для решения комбинаторных задач в некоторых случаях можно использовать методы математического анализа. Разнообразие применяемых здесь приемов весьма велико, и не может быть в полном объеме рассмотрено в рамках этой книги. В данном разделе рассматривается только основная идея метода производящих функций, применение которой иллюстрируется простым примером. Более детальное рассмотрение можно найти в литературе, например в [20].

5.7.1. Основная идея

Пусть есть последовательность комбинаторных чисел a_i и последовательность функций $\varphi_i(x)$. Рассмотрим формальный ряд:

$$\mathcal{F}(x) := \sum_i a_i \varphi_i(x).$$

$\mathcal{F}(x)$ называется *производящей функцией* (для заданной последовательности комбинаторных чисел a_i относительно заданной последовательности функций $\varphi_i(x)$).

Обычно используют $\varphi_i(x) := x^i$ или $\varphi_i(x) := x^i/i!$.

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 неопределенных коэффициентов

Из математического анализа известно, что если

$$\mathcal{F}(x) := \sum_i a_i \varphi_i(x) \text{ и } \mathcal{F}(x) := \sum_i b_i \varphi_i(x),$$

то $\forall i\ a_i = b_i$ (для рассматриваемых здесь систем функций φ_i).

В качестве примера применения производящих функций рассмотрим доказательство следующего тождества.

ТЕОРЕМА $C(2n, n) = \sum_{k=0}^{n} C(n, k)^2$.

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:

$$C(2n, n) = \sum_{k=0}^{n} C(n, k) C(n, n-k) = \sum_{k=0}^{n} C(n, k)^2. \qquad \square$$

Комментарии

Сведения из области комбинаторного анализа в том или ином объеме приводятся в любом учебнике по дискретной математике (см., например, [25, 18]). 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 книгах [19] и [20]. Все алгоритмы этой главы заимствованы (в модифицированном виде) из книги [14].

Упражнения

5.1. Доказать, что $A(m, n) = A(m-1, n) + nA(m-1, n-1)$.

5.2. Доказать, что множество перестановок с четным числом инверсий образует группу.

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. *Числа Фибоначчи* $F(n)$ определяются следующим образом:

$$F(0) := 1,\ F(1) := 1,\ F(n+2) := F(n+1) + F(n).$$

Найти выражение для $F(n)$ через n (указание: рассмотреть производящую функцию $1/(1-x-x^2)$).

ГЛАВА 6 Кодирование

Вопросы кодирования издавна играли заметную роль в математике.

Пример

1. *Десятичная позиционная система счисления* — это способ кодирования натуральных чисел. Римские цифры — другой способ кодирования натуральных чисел, причем гораздо более наглядный и естественный: палец — I, пятерня — V, две пятерни — X. Однако при этом способе кодирования труднее 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:

- 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.

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 $F\colon S \to B^*$, где S — некоторое множество слов в алфавите A, $S \subset A^*$. Тогда функция F иазывается *кодированием*, элементы множества S — *сообщениями*, а элементы $\beta = F(\alpha)$, $\alpha \in S$, $\beta \in B^*$ — *кодами* (соответствующих сообщений). Обратная фуикция F^{-1} (если она существует!) называется *декодированием*.

Если $|B| = m$, то F называется *m-ичным кодированием*. Наиболее распространеиный 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 сообщеиий S найти такое кодирование F, которое обладает определенными свойствами (то есть удовлетворяет заданным ограничеииям) 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 — это очень естественное свойство, одиако даже оно требуется не всегда. Например, трансляция программы на языке высокого уровня в машинные команды — это кодирование, для которого не требуется одиозначного декодирования;
- помехоустойчивость, или исправление ошибок: функция декодирования F^{-1} обладает таким свойством, что $F^{-1}(\beta) = F^{-1}(\beta')$, если β' в определеиием смысле близко к β (см. раздел 6.3);
- заданная сложность (или простота) кодирования и декодирования. Например, в криптографии изучаются такие способы кодирования, при которых имеется просто вычисляемая фуикция F, но определение функции F^{-1} требует очень сложных вычислеиий (см. подраздел 6.5.5).

Большое значение для задач кодирования пмеет природа множества сообщеиий S. При одних и тех же алфавитах A, B и требуемых свойствах кодирования F оптимальные решения могут кардинально отличаться для разных S. 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, напрпмер $S = A^*$, и заданы вероятности p_i появления букв 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 упомянутых здесь приемов.

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. Префикс и постфикс слова

Пусть задано конечное множество $A = \{a_1, \ldots, a_n\}$, которое называется *алфавитом*. Элементы алфавита называются *буквами*. Последовательность букв называется *словом* (в данном алфавите). 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 Λ: $\Lambda \in A^*$, $|\Lambda| = 0$, $\Lambda \notin A$.

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 $\alpha_1 \neq \Lambda$ (соответственно, $\alpha_2 \neq \Lambda$), то α_1 (соответственно, α_2) называется *собственным началом* (соответственно, *собственным окончанием*) слова α.

6.1.2. Таблица кодов

Алфавитное (или *побуквенное*) кодирование задается *схемой* (или *таблицей кодов*) σ:

$$\sigma := \langle a_1 \to \beta_1, \ldots, a_n \to \beta_n \rangle, \qquad a_i \in A, \quad \beta_i \in B^*.$$

Множество кодов букв $V := \{\beta_i\}$ называется множеством *элементарных кодов*.

Алфавитное кодирование пригодно для любого множества сообщений S:

$$F\colon A^* \to B^*, \qquad a_{i_1} \ldots a_{i_k} = \alpha \in A^*, \qquad F(\alpha) := \beta_{i_1} \ldots \beta_{i_k}.$$

Пример

Рассмотрим алфавиты $A := \{0,1,2,3,4,5,6,7,8,9\}$, $B := \{0,1\}$ и схему

$$\delta := \langle 0 \to 0, 1 \to 1, 2 \to 10, 3 \to 11, 4 \to 100, 5 \to 101, 6 \to 110,$$
$$7 \to 111, 8 \to 1000, 9 \to 1001 \rangle.$$

Эта схема однозначна, но кодирование не является взаимно однозначным:

$$F_\delta(333) = 111111 = F_\delta(77),$$

а значит, невозможно декодирование. С другой стороны, схема

$$\delta := \langle 0 \to 0000, 1 \to 0001, 2 \to 0010, 3 \to 0011, 4 \to 0100, 5 \to 0101, 6 \to 0110,$$
$$7 \to 0111, 8 \to 1000, 9 \to 1001 \rangle,$$

известная под названием «*двоично-десятичное* кодирование», допускает однозначное декодирование.

6.1.3. Разделимые схемы

Рассмотрим схему алфавитного кодирования σ и различные слова, составленные из элементарных кодов. Схема σ называется *разделимой*, если

$$\beta_{i_1} \dots \beta_{i_k} = \beta_{j_1} \dots \beta_{j_l} \Longrightarrow k = l \,\&\, \forall t \in 1..k \; i_t = j_t,$$

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

$$\forall i \neq j \; \beta_i, \beta_j \in V \Longrightarrow \beta_i \neq \beta_j.$$

6.1.4. Префиксные схемы

Схема σ называется *префиксной*, если элементарный код одной буквы не является префиксом элементарного кода другой буквы:

$$(\forall i \neq j \; \beta_i, \beta_j \in V) \Longrightarrow (\forall \beta \in B^* \; \beta_i \neq \beta_j \beta).$$

ТЕОРЕМА *Префиксная схема является разделимой.*

Доказательство

От противного. Пусть кодирование со схемой σ не является разделимым. Тогда существует такое слово $\beta \in F_\sigma(A^*)$, что

$$\beta = \beta_{i_1} \dots \beta_{i_k} = \beta_{j_1} \dots \beta_{j_l} \,\&(\exists t \; \forall s \; (s < t \Longrightarrow \beta_{i_s} = \beta_{j_s} \,\&\, \beta_{i_t} \neq \beta_{j_t})).$$

Поскольку $\beta_{i_t} \dots \beta_{i_k} = \beta_{j_t} \dots \beta_{j_l}$, 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 является достаточным, но не является необходимым для разделимости схемы.

Пример

Разделимая, но не префиксная схема: $A = \{a, b\}$, $B = \{0, 1\}$, $\delta = \{a \to 0, b \to 01\}$.

6.1.5. Неравенство Макмиллана

Чтобы схема алфавитного кодирования была разделимой, необходимо, чтобы длины элементарных кодов удовлетворяли определенному соотношению, известному как *неравенство Макмиллана*.

ТЕОРЕМА *Если схема* $\sigma = (a_i \to \beta_i)_{i=1}^n$ *разделима, то*

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leqslant 1, \qquad \text{где } l_i := |\beta_i|.$$

Доказательство

Обозначим $l := \max\limits_{i=1}^{n} l_i$. Рассмотрим n-ю степень левой части неравенства

$$\Big(\sum_{i=1}^{n} 2^{-l_i}\Big)^n.$$

Раскрывая скобки, имеем сумму

$$\sum_{(i_1,\ldots,i_n)} (2^{l_{i_1}+\cdots+l_{i_n}})^{-1},$$

где $i_1, \ldots, i_n$ — различные наборы номеров элементарных кодов. Обозначим через $\nu(n,t)$ количество входящих в эту сумму слагаемых вида $1/2^t$, где $t = l_{i_1}+\cdots+l_{i_n}$. Для некоторых t может быть, что $\nu(n,t) = 0$. Приводя подобные, имеем сумму

$$\sum_{t=1}^{nl} \frac{\nu(n,t)}{2^t}.$$

Каждому слагаемому вида $(2^{l_{i_1}+\cdots+l_{i_n}})^{-1}$ можно однозначно сопоставить код (слово в алфавите B) вида $\beta_{i_1}\ldots\beta_{i_n}$. Это слово состоит из n элементарных кодов и имеет длину t.

Таким образом, $\nu(n,t)$ — это число некоторых слов вида $\beta_{i_1}\ldots\beta_{i_n}$, таких что $|\beta_{i_1}\ldots\beta_{i_n}| = t$. В силу разделимости схемы $\nu(n,t) \leqslant 2^t$, в противном случае заведомо существовали бы два одинаковых слова $\beta_{i_1}\ldots\beta_{i_n} = \beta_{j_1}\ldots\beta_{j_n}$, допускающих различное разложение. Имеем

$$\sum_{t=1}^{nl} \frac{\nu(n,t)}{2^t} \leqslant \sum_{t=1}^{nl} \frac{2^t}{2^t} = nl.$$

Следовательно, $\forall n\ \Big(\sum\limits_{i=1}^{n} 2^{-l_i}\Big)^n \leqslant nl$, и значит, $\sum\limits_{i=1}^{n} 2^{-l_i} \leqslant \sqrt[n]{nl}$, откуда

$$\sum_{i=1}^{n} 2^{-l_i} \leqslant \lim_{n\to\infty} \sqrt[n]{nl} = 1. \qquad \square$$

Неравенство Макмиллана является не только необходимым, но и в некотором смысле достаточным условием разделимости схемы алфавитного кодирования.

ТЕОРЕМА *Если числа $l_1, \ldots, l_n$ удовлетворяют неравенству*

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leqslant 1,$$

то существует разделимая схема алфавитного кодирования $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$, где $\forall i\ l_i = |\beta_i|$.

Доказательство

Без ограничений общности можно считать, что $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_n$. Разобьем множество $\{l_1, \ldots, l_n\}$ на классы эквивалентности по равенству $\{L_1, \ldots, L_m\}$, $m \leqslant n$. Пусть

$$\lambda_i \in L_i,\ \mu_i := |L_i|,\ \sum_{i=1}^{m} \mu_i = n,\ \lambda_1 < \lambda_2 < \cdots < \lambda_m.$$

Тогда 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:

$$\frac{\mu_1}{2^{\lambda_1}} \leqslant 1 \Longrightarrow \mu_1 \leqslant 2^{\lambda_1} \tag{1}$$

$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} \leqslant 1 \Longrightarrow \mu_2 \leqslant 2^{\lambda_2} - \mu_1 2^{\lambda_2 - \lambda_1} \tag{2}$$

$$\cdots$$

$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} + \cdots + \frac{\mu_m}{2^{\lambda_m}} \leqslant 1 \Longrightarrow \mu_m \leqslant 2^{\lambda_m} - \mu_1 2^{\lambda_m - \lambda_1} - \mu_2 2^{\lambda_m - \lambda_2} - \cdots - \mu_{m-1} 2^{\lambda_m - \lambda_{m-1}}. \tag{m}$$

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 так далее, используя неравенства для частичных сумм, мы будем на i-м шаге выбирать μ_i слов длины λ_i, $\beta_{\mu_1+\mu_2+\cdots+\mu_{i-1}+1}, \ldots, \beta_{\mu_1+\mu_2+\cdots+\mu_{i-1}+\mu_i}$, причем эти слова не будут начинаться с тех слов, которые были выбраны раньше. В то же время длины этих слов все время растут (так как $\lambda_1 < \lambda_2 < \cdots < \lambda_m$), поэтому они не могут быть префиксами тех слов, которые выбраны раньше. Итак, в конце имеем набор из n слов $\beta_1, \ldots, \beta_{\mu_1+\cdots+\mu_m} = \beta_n$, $|\beta_1| = l_1, \ldots, |\beta_n| = l_n$, 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 — 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:

$$1/4 + 1/16 + 1/16 + 1/8 + 1/2 + 1/16 + 1/8 + 1/16 + 1/4 + 1/16 +$$
$$+ 1/8 + 1/16 + 1/4 + 1/4 + 1/8 + 1/16 + 1/16 + 1/8 + 1/8 +$$
$$+ 1/2 + 1/8 + 1/16 + 1/8 + 1/16 + 1/16 + 1/16 =$$
$$= 2/2 + 4/4 + 7/8 + 12/16 = 3 + 5/8 > 1.$$

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 прием и передача сообщений с помощью азбуки Морзе, особенно с высокой скоростью, является некоторым искусством, а не простой технической процедурой.

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 $S = A^*$. 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 $\langle \beta'_1, \ldots, \beta'_n \rangle$ является перестановкой $\langle \beta_1, \ldots, \beta_n \rangle$, 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 элемеитарные 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, 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 будет минимальна: нужно отсортировать буквы 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 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 $p_i + \cdots + p_n = 1$ и $p_1 \geqslant \cdots \geqslant p_n > 0$ (то есть можно сразу исключить буквы, которые не могут появиться в сообщении, и упорядочить буквы по убыванию вероятности их появления).

Для каждой (разделимой) схемы $\sigma = \{a_i \to \beta_i\}_{i=1}^n$ алфавитного кодирования математическое ожидание коэффициента увеличения длины сообщения при кодировании (обозначается l_σ) определяется следующим образом:

$$l_\sigma(P) := \sum_{i=1}^{n} p_i l_i, \qquad \text{где} \quad l_i := |\beta_i|$$

и называется средней *ценой* (или *длиной*) кодирования σ при распределении вероятностей P.

Пример

Для разделимой схемы $A = \{a, b\}$, $B = \{0, 1\}$, $\delta = \{a \to 0, b \to 01\}$ 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 $0.9 * 1 + 0.1 * 2 = 1.1$.

Обозначим

$$P) := \inf_{\sigma} l_\sigma(P), \qquad p_* := \min_{i=1}^{n} p_i. \qquad L_i = \lfloor \log_2(n-1) \rfloor + 1$$

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 $l_*(P) \leqslant l_\sigma(P) \leqslant L$. Следовательно, 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

Следующий рекурсивный алгоритм 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; $1 \geqslant P[1] \geqslant \cdots \geqslant P[n] > 0$, $P[1] + \cdots + P[n] = 1$.

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 }
repeat
$d := S_b - S_e$ { разность сумм первой и второй части }
$m := m - 1$ { сдвигаем границу медианы вниз }
$S_b := S_b - P[m]; S_e := S_e + P[m]$
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
0.20	00	2
0.20	010	3
0.19	011	3
0.12	100	3
0.11	101	3
0.09	110	3
0.09	111	3
$l_\sigma(P)$		2.80

6.2.4. Оптимальное кодирование

Оптимальное кодирование обладает определенными свойствами, которые можно использовать для его построения.

ЛЕММА *Пусть* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *— схема оптимального кодирования для распределения вероятностей* $P = p_1 \geqslant \cdots \geqslant p_n > 0$. *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

$$\sigma' = \{a_1 \to \beta_1, \ldots, a_i \to \beta_j, \ldots, a_j \to \beta_i, \ldots, a_n \to \beta_n\}.$$

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 σ — оптимально. □

Таким образом, не ограничивая общности, можно считать, что $l_1 \leqslant \cdots \leqslant l_n$.

ЛЕММА *Если $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \cdots \geqslant p_n > 0$, то среди элементарных кодов, имеющих максимальную длину, имеются два, которые различаются только в последнем разряде.*

Доказательство

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, причем $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ.
2. Пусть теперь два кодовых слова β_{n-1} и β_n максимальной длины отличаются не в последнем разряде, то есть $\beta_{n-1} = \beta' b'$, $\beta_n = \beta'' b''$, $\beta' \neq \beta''$, причем 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, причем $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ. □

ТЕОРЕМА *Если $\sigma_{n-1} = \langle a_i \to \beta_i \rangle_{i=1}^{n-1}$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \cdots \geqslant p_{n-1} > 0$ и $p_j = q' + q''$, причем*

$$p_1 \geqslant \cdots \geqslant p_{j-1} \geqslant p_{j+1} \geqslant \cdots \geqslant p_{n-1} \geqslant q' \geqslant q'' > 0,$$

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

1. Если σ_{n-1} было префиксным, то σ_n тоже будет префиксным по построению.

2. $$\begin{aligned} l_{\sigma_n}(P_n) &= p_1 l_1 + p_2 l_2 + \cdots + p_{j-1} l_{j-1} + p_{j+1} l_{j+1} + \cdots + \\ &+ p_{n-1} l_{n-1} + q' l_{j+1} + q'' l_{j+1} = \\ &= p_1 l_1 + \cdots + p_{n-1} l_{n-1} + l_j (q' + q'') + (q' + q'') = \\ &= p_1 l_1 + \cdots + p_{n-1} l_{n-1} + l_j p_j + p_j = l_{\sigma_{n-1}}(P_{n-1}) + p_j \end{aligned}$$

3. Пусть схема $\sigma'_n := \{a_i \to \beta_i\}_{i=1}^n$ оптимальна для P_n. Тогда по предшествующей лемме $\sigma'_n = \{a_1 \to \beta'_1, \ldots, a_{n-2} \to \beta'_{n-2}, a_{n-1} \to \beta 0, a_n \to \beta 1\}$. Положим $l' = |\beta|$ и рассмотрим схему $\sigma'_{n-1} := \{a_1 \to \beta'_1, \ldots, a_j \to \beta, \ldots, a_{n-2} \to \beta'_{n-2}\}$, где j определено так, чтобы $p_{j-1} \geqslant q' + q'' \geqslant p_{j+1}$.

4. $$\begin{aligned} l_{\sigma'_n}(P_n) &= l_1 p_1 + \cdots + l_{n-2} p_{n-2} + (l' + 1) q' + (l' + 1) q'' = \\ &= l_1 p_1 + \cdots + l_{n-2} p_{n-2} + l' (q' + q'') + (q' + q'') = \\ &= l_{\sigma'_{n-1}}(P_{n-1}) + p_j. \end{aligned}$$

5. σ'_n — префиксное, значит, σ'_{n-1} тоже префиксное.

6. σ_{n-1} — оптимально, значит, $l_{\sigma'_{n-1}}(P_{n-1}) \geqslant l_{\sigma_{n-1}}(P_{n-1})$.

7. $l_{\sigma_n}(P_n) = l_{\sigma_{n-1}}(P_{n-1}) + p_j \leqslant l_{\sigma'_{n-1}}(P_{n-1}) + p_j = l_{\sigma'_n}(P_n)$, но σ'_n — оптимально, значит, σ_n оптимально. □

6.2.5. Алгоритм Хаффмена

Следующий рекурсивный алгоритм 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. предыдущий алгоритм) и вставляет это число, сдвигая вниз остальные элементы.

Вход: n — длина обрабатываемой части массива P, q — вставляемая сумма.
Выход: измененный массив P.

```
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 − 1{ определение места вставляемого элемента }
    exit for i { все 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 трех, четырех 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.

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

Надежность электронных устройств по мере их совершенствования все время возрастает, но, тем не менее, в их работе возможны ошибки, как систематические, так и случайные. Сигнал в канале связи может быть искажен помехой, поверхность магнитного носителя может быть повреждена, в разъеме может быть потерян контакт. Ошибки аппаратуры ведут к искажению или потере передаваемых или хранимых данных. При определенных 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 прием данных из канала.

6.3.1. Кодирование с исправлением ошибок

Пусть имеется канал связи C, содержащий источник помех:

$$S \xrightarrow{C} S' \quad S \in A^*, S' \in B^*,$$

где S — множество переданных, а S' — соответствующее множество принятых по каналу сообщений. Кодирование F, обладающее таким свойством, что

$$S \xrightarrow{F} K \xrightarrow{C} K' \xrightarrow{F^{-1}} S, \ \forall s \in S\ ,\ F^{-1}(C(F(s))) = s;$$

называется *помехоустойчивым*, или *самокорректирующимся*, или кодированием *с исправлением ошибок*.

Без ограничения общности можно считать, что $A = B = \{0, 1\}$, и что содержательное кодирование выполняется на устройстве, свободном от помех.

6.3.2. Классификация ошибок

Ошибки в канале могут быть следующих типов:

- $0 \to 1$, $1 \to 0$ — ошибка типа замещения разряда;
- $0 \to \Lambda$, $1 \to \Lambda$ — ошибка типа выпадения разряда;
- $\Lambda \to 1$, $\Lambda \to 0$ — ошибка типа вставки разряда.

Каиал характеризуется верхними оценками количества ошибок каждого типа, которые возможны при передаче через канал сообщения определенной длины. Общая характеристика ошибок канала (то есть их количество и типы) обозначается δ.

Пример

Допустим, что имеется каиал с характеристикой $\delta = (1, 0, 0)$, то есть в канале возможна одна ошибка типа замещения разряда при передаче сообщения 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 при передаче сообщения длины $3n$ возможно не более 3 ошибок типа эамещения разряда, но места этих ошибок совершенно не обязательно распределены равномерно по всему сообщению. Ошибки замещения могут произойти в соседних разрядах, и метод голосоваиия восстановит разряд неверно.

6.3.3. 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 δ, то есть $s \in S \subset A^*$, $E_s^\delta \subset B^*$. Если $s' \in E_s^\delta$, то та конкретиая последовательность ошибок, которая позволяет получить из слова s слово s', обозначается $E^\delta(s, s')$. Если тип возможных ошибок в канале подразумевается, то индекс δ не указывается.

ТЕОРЕМА *Чтобы существовало помехоустойчивое кодирование с исправлением всех ошибок, необходимо и достаточно, чтобы* $\forall s1, s2 \in S\ E_{s_1} \cap E_{s_2} = \varnothing$, *то есть необходимо и достаточно, чтобы существовало разбиение множества* B^* *на множества* B_s ($\bigcup B_s = B^*$, $\bigcap B_s = \varnothing$), *такое что* $\forall s \in S\ E_s \subset B_s$.

Доказательство

Если кодирование помехоустойчивое, то очевидно, что $E_{s_1} \cap E_{s_2} = \varnothing$. Обратно: по разбиению $\bigcup B_s$ строится функция F^{-1}: $\forall s' \in B_s\ F^{-1}(s') := s$. □

Пример

Рассмотрим канал, в котором в любом передаваемом разряде происходит ошибка типа замещения с вероятностью p $(0 < p < 1/2)$, причем 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, $\forall s\ E_s = E_2^n$, и исправить все ошибки в двоичном симметричном канале невозможно.

6.3.4. 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. $d(x, y) \leqslant d(x, z) + d(y, z)$.

Пусть

$$d_\delta(\beta', \beta'') := \begin{cases} \min_{\{E^\delta\langle\beta',\beta''\rangle\}} |E^\delta\langle\beta', \beta''\rangle|, & \beta'' \in E^\delta_{\beta'}, \\ +\infty, & \beta'' \notin E^\delta_{\beta'}. \end{cases}$$

Эта функция называется *расстоянием Хэмминга*.

ЗАМЕЧАНИЕ

Мы рассматриваем симметричные ошибки, то есть если в канале допустима ошибка $0 \to 1$, то допустима и ошибка $1 \to 0$.

Введенная функция d_δ является расстоянием. Действительно:

1. $d_\delta(\beta', \beta') = 0 \iff \beta' = \beta''$, поскольку ошибки симметричны, и из последовательности $E\langle\beta', \beta''\rangle$ можно получить последовательность $E\langle\beta'', \beta'\rangle$, применяя обратные ошибки в обратном порядке.
2. $d_\delta(\beta', \beta'') = d_\delta(\beta'', \beta')$ по той же причине.
3. $d_\delta(\beta', \beta'') \leqslant d_\delta(\beta', \beta''') + d_\delta(\beta'', \beta''')$, поскольку $E\langle\beta', \beta'''\rangle \cup E\langle\beta''', \beta''\rangle$ является некоторой последовательностью, преобразующей β' в β'', а $d_\delta(\beta', \beta'')$ является кратчайшей из таких последовательностей.

Пусть $\sigma = \langle a_i \to \beta_i\rangle_{i=1}^n$ — схема некоторого алфавитного кодирования, а d — некоторая метрика на B^*. Тогда минимальное расстояние между элементарными кодами

$$d(\sigma) := \min_{1 \leqslant i < j \leqslant n} d(\beta_i, \beta_j)$$

называется *кодовым расстоянием* схемы σ.

ТЕОРЕМА *Алфавитное кодирование со схемой* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ *и с кодовым расстоянием*

$$d_\delta(\sigma) = \min_{\beta', \beta'' \in V} d_\delta(\beta', \beta'')$$

является кодированием с исправлением p *ошибок типа* δ *тогда и только тогда, когда* $d_\delta(\sigma) > 2p$.

Доказательство

$E(x)$ — это шар радиуса p с центром в x для канала, который допускает не более p ошибок типа δ. Таким образом,

$$E(\beta') \cap E(\beta'') = \varnothing \Longleftrightarrow d_\delta(\sigma) > 2p$$

по теореме 6.3.3.

Пример

Расстояние Хэмминга в E_2^n: $d(\beta', \beta'') := \sum_{i=1}^{n} (\beta_i' \neq \beta_i'')$.

6.3.5. Код Хэмминга для исправления одного замещения

Рассмотрим построение *кода Хэмминга*, который позволяет исправлять одиночные ошибки типа замещения.

Очевидно, что для исправления ошибки вместе с основным сообщением нужно передавать какую-то дополнительную информацию.

Пусть сообщение $\alpha = a_1 \dots a_m$ кодируется словом $\beta = b_1 \dots b_n$, $n > m$. 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 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 $m = 32$ потребуется $k = 6$ 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}.

Рассмотрим в слове $b_1 \ldots b_n$ k разрядов с номерами $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, ... 2^{k-1}. Эти разряды назовем *контрольными*. Остальные разряды, а их ровно m, назовем *информационными*. Поместим в информационные разряды все разряды слова $a_1 \ldots a_n$ как они есть. Контрольные разряды определим следующим образом:

- $b_1 := b_3 \oplus b_5 \oplus b_7 \oplus \ldots$, — то есть все разряды с номерами из V_1, кроме первого;
- $b_2 := b_3 \oplus b_6 \oplus b_7 \oplus \ldots$, — то есть все разряды с номерами из V_2, кроме первого;
- $b_4 := b_5 \oplus b_6 \oplus b_7 \oplus \ldots$, — то есть все разряды с номерами из V_3, кроме первого;

и вообще,

$$b_{2^{j-1}} := \bigoplus_{i \in V_j \setminus \{2^{j-1}\}} b_i.$$

Пусть после прохождения через канал получен код $c_1 \ldots c_n$,
то есть $c_1 \ldots c_n := C(b_1 \ldots b_n)$, причем

$$\exists I \; c_I = \begin{cases} b_I \\ \overline{b_I} \end{cases} \qquad \forall i \neq I \; c_i = b_i.$$

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:

- $j_1 := c_1 \oplus c_3 \oplus c_5 \oplus c_7 \oplus \ldots$, — то есть все разряды с номерами из V_1;
- $j_2 := c_2 \oplus c_3 \oplus c_6 \oplus c_7 \oplus \ldots$, — то есть все разряды с номерами из V_2;
- $j_3 := c_4 \oplus c_5 \oplus c_6 \oplus b_7 \oplus \ldots$, — то есть все разряды с номерами из V_3;

и вообще,

$$j_p := \bigoplus_{q \in V_p} c_q.$$

ТЕОРЕМА $I = J$.

Доказательство

Эти числа равны, потому что поразрядно равны их двоичные представления. Действительно, пусть $i_1 = 0$. Тогда $I \notin V_1$, и значит,

$$j_1 = c_1 \oplus c_3 \oplus c_5 \oplus \cdots = b_1 \oplus b_3 \oplus b_5 \oplus \cdots = 0$$

по определению разряда b_1. Пусть теперь $i_1 = 1$. Тогда $I \in V_1$, и значит,

$$j_1 = c_1 \oplus c_3 \oplus c_5 \oplus \cdots = b_1 \oplus b_3 \oplus b_5 \oplus \cdots \oplus \overline{b_x} \oplus \cdots = 1,$$

так как если в сумме по модулю два 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.

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 определенных 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 оптимальным для текстов. Действительно, в текстах обычно используется существенно меньше, чем 256 символов (в зависимости от языка — примерно 60–80 с учетом 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 расчеты 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, которые позволяют построить (без потери информации!) коды сообщений, имеющие меньшую длину 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 (arj, zip и другие) имеют гораздо лучшие показатели: 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 символов, 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 дает существенное сжатие текста (около 75% для обычных текстов на русском языке). Если текст достаточно велик (сотни тысяч или миллионы букв), 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, причем 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 дает наименьшую

цену кодирования на шаге 2. Такое кодирование будет «абсолютно» оптимальным. К сожалению, указанная экстремальная задача очень трудоемка, поэтому на практике не используется — время на предварительную обработку большого текста оказывается чрезмерно велико.

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 словарь не хранится — за счет того, что при декодировании используется тот же самый алгоритм построения словаря, словарь динамически восстанавливается.

Здесь приведена простейшая реализация этой идеи, известная как *алгоритм Лемпела–Зива*. 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 еще один символ. В сжатое представление записывается найденный код слова и расширяющая буква, а словарь пополняется расширенной комбинацией.

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] := ""; d := 0 { начальное состояние словаря }
k := 1 { номер текущей буквы в исходном тексте }
while k ⩽ n do
  p := FD(k) { p — индекс найденного слова в словаре }
  l := Length(D[p]) { l — длина найденного слова в словаре }
  yield ⟨p, f[k + l]⟩ { код найденного слова и еще одна буква }
  d := d + 1; D[d] := D[p] ∪ f[k + l] { пополнение словаря, здесь ∪ — это конкатенация }
  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
  if D[i] = f[k..k + Length(D[i]) − 1] & Length(D[i]) > l then
    p := i; l := Length(D[i]) { 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 endrecord**, где p — номер слова в словаре, q — код дополняющей буквы.

Выход: исходный текст, заданный последовательностью строк и символов.

```
D[0] := ""; d := 0 { начальное состояние словаря }
for k ⩽ n do
  p := g[k].p { p — индекс слова в словаре }
  q := g[k].q { q — дополнительная буква }
  yield p ∪ q { вывод слова и еще одной буквы }
  d := d + 1; D[d] := D[p] ∪ q { пополнение словаря, здесь ∪ — это конкатенация }
end for
```

ЗАМЕЧАНИЕ

На практике применяют различные усовершенствования этой схемы.

1. Словарь можно сразу инициализировать, например, кодами символов (то есть считать, что одноб уквенные 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 прием, например, закодировать последовательность пробелов парой $\langle k, s\rangle$, где k — количество пробелов, а s — код пробела.

6.5. Шифрование

Защита компьютерных данных от несанкционированного доступа, искажения и уничтожения в настоящее время является серьезной социальной проблемой. Применяются различные подходы к решению этой проблемы.

- Поставить между злоумышленником и данными в компьютере непреодолимый барьер, то есть исключить саму возможность доступа к данным путем физической изоляции компьютера с данными, применения аппаратных ключей защиты и т. п. Такой подход надежен, но он затрудняет доступ к данным и легальным пользователям, 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, разграничения прав доступа и т. п. Практика показывает, что борьба между «хакерами» и модулями защиты операционных систем идет с переменным успехом.
- Хранить данные таким образом, чтобы они могли «сами за себя постоять». Другими словами, так закодировать данные, чтобы даже получив их, злоумышленник не смог бы нанести ущерба.

Этот раздел посвящен 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 шифрах, методах их создания и раскрытия называется *криптографией* (или *тайнописью*).

Свойство шифра противостоять раскрытию называется *криптостойкостью* (или *надежностью*) и обычно измеряется 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 и т. п.) больше, чем сама зашифрованная информация, то шифр считается достаточно надежным.

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 определенному 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 — нечетное (см. [9]).

В этом случае последовательность псевдослучайных чисел имеет период c (см. [9]).

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

$$C_i := S_i \oplus T_i.$$

ЗАМЕЧАНИЕ

Последовательность $T_0, T_1, \ldots$ называется *гаммой шифра*.

Процесс расшифровывания заключается в том, чтобы еще раз сложить шифрованную последовательность с той же самой гаммой шифра:

$$S_i := C_i \oplus T_i.$$

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

Это 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 надежной 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 все сообщение может быть легко дешифровано. Действительно, пусть известно одно исходное слово S_i. Тогда

$$T_i := C_i \oplus S_i,$$

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 приемы:

1. вычисление гаммы шифра по ключу более сложным (или секретным) способом;
2. применение вместо $\oplus$ более сложной (но обратимой) операции для вычисления шифровки;
3. предварительное перемешивание битов исходного сообщения по фиксированному алгоритму.

Наиболее надежным симметричным шифром считается DES (Data Encryption Standard), в котором используется сразу несколько методов повышения криптостойкости.

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).

Для описания метода шифрования с открытым ключом нужны некоторые факты из теории чисел, изложенные (без доказательств) в следующем подразделе.

6.5.4. Модулярная арифметика

В этом подразделе все числа целые. Говорят, что число a *сравнимо по модулю* n с числом b (обозначение: $a \equiv b \pmod{n}$), если a и b при делении на n дают один и тот же остаток:

$$a \equiv b \pmod{n} :\Leftrightarrow a \bmod n = b \bmod n.$$

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:

$$a +_n b :=(a + b) \mod n, \qquad a \cdot_n b :=(a \cdot b) \mod n.$$

ЗАМЕЧАНИЕ

Если из контекста ясно, что подразумеваются операции по модулю n, то индекс n опускается.

Легко видеть, что $\langle \mathbb{Z}_n; +_n \rangle$ образует абелеву группу, а $\langle \mathbb{Z}_n; +_n, \cdot_n \rangle$ — коммутативное кольцо с единицей.

Рассмотрим $\mathbb{Z}_n^*$ — подмножество $\mathbb{Z}_n$ чисел, 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 чисел из множества $\mathbb{Z}_n^*$ существуют обратные по умножению по модулю n.

ЗАМЕЧАНИЕ

Если n — простое число, то $\langle \mathbb{Z}_n^*; +_n, \cdot_n \rangle$ является полем.

Функция $\varphi(n) := |\mathbb{Z}_n^*|$ называется *функцией Эйлера*.

ЗАМЕЧАНИЕ

Если p — простое число, то $\varphi(p) = p - 1$, и вообще, $\varphi(n) < n$.

Можно показать, что

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right)\ldots\left(1 - \frac{1}{p_k}\right),$$

где $p_1, \ldots, p_k$ — все простые делители n.

Имеет место следующая теорема.

ТЕОРЕМА (Эйлера) *Если $n > 1$, то*

$$\forall a \in \mathbb{Z}_n^*\ a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Отсюда непосредственно выводима

ТЕОРЕМА (малая теорема Ферма) *Если $p > 1$ — простое число, то*

$$\forall a \in \mathbb{Z}_p^*\ a^{p-1} \equiv 1 \pmod{p}.$$

Имеет место следующее утверждение.

ТЕОРЕМА *Если числа $n_1, \ldots, n_k$ попарно взаимно простые, число $n = n_1 n_2 \ldots n_k$ — их произведение, x и a — целые числа, то*

$$x \equiv a \pmod{n} \Longleftrightarrow \forall i \in 1..k\ x \equiv a \pmod{n}_i.$$

ЗАМЕЧАНИЕ

Последнее утверждение является следствием теоремы, которая известна как «китайская теорема об остатках».

6.5.5. Шифрование с открытым ключом

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;
   - определяется первая часть открытого ключа $n := pq$;
   - определяется вторая часть открытого ключа — выбирается небольшое нечетное 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: $d := e^{-1} \mod ((p-1)(q-1))$.

   После чего открытый ключ (числа n и e) сообщается всем отправителям сообщений.
2. Отправитель шифрует сообщение (разбивая его, если нужно, на слова S_i длиной менее $\log_2 n$ разрядов):
$$C_i := (S_i)^e \mod n$$
   и отправляет получателю.
3. Получатель расшифровывает сообщение с помощью закрытого ключа d:
$$P_i := (C_i)^d \mod n.$$

ТЕОРЕМА *Шифрование с открытым ключом корректно, то есть в предыдущих обозначениях* $P_i = S_i$.

Доказательство

Легко видеть, что $P_i = (S_i)^{ed} \mod n$. Покажем, что $\forall M < n\ M^{ed} \equiv M \mod n$. 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 $M \not\equiv 0 \pmod p$, то по малой теореме Ферма имеем:
$$M^{ed} \equiv M\left(M^{(p-1)}\right)^{k(q-1)} \equiv M \cdot 1^{k(q-1)} \equiv M \pmod p.$$
Если $M \equiv 0 \pmod p$, то сравнение $M^{ed} \equiv M \pmod p$, очевидно, выполняется. Таким образом,
$$\forall 0 \leqslant M < n\ M^{ed} \equiv M \pmod p.$$
Совершенно аналогично имеем
$$\forall 0 \leqslant M < n\ M^{ed} \equiv M \pmod q,$$

и по следствию к китайской теореме об остатках

$$\forall M < n \; M^{ed} \equiv M \pmod{n}.$$

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. $d := 7^{-1} \mod 20 = 3$, $(7 * 3 \mod 20 = 1)$.

Пусть $S_1 := 3$, $S_2 := 1$, $S_3 := 2$ $(S_1, S_2, S_3 < n = 33)$. Тогда код определяется следующим образом.

1. $C_1 := 3^7 \mod 33 = 2187 \mod 33 = 9$;
2. $C_2 := 1^7 \mod 33 = 1 \mod 33 = 1$;
3. $C_3 := 2^7 \mod 33 = 128 \mod 33 = 29$.

При расшифровке имеем:

1. $P_1 := 9^3 \mod 33 = 729 \mod 33 = 3$;
2. $P_2 := 1^3 \mod 33 = 1 \mod 33 = 1$;
3. $P_3 := 29^3 \mod 33 = 24389 \mod 33 = 2$.

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 партнерам свой открытый ключ.

Заметим, что операции зашифровки и расшифровки по существу одинаковы, и различаются только показателем степени, а потому коммутируют:

$$M = (M^e)^d \bmod n = M^{ed} \bmod n = M^{de} \bmod n = (M^e)^d \bmod n = M.$$

Это обстоятельство позволяет применять различные приемы, известные как *цифровая* (или *электронная*) подпись.

Рассмотрим следующую схему взаимодействия корреспондентов X и Y. Отправитель X кодирует сообщение S своим закрытым ключом ($C := M^d \bmod n$) и посылает получателю Y пару $\langle S, C\rangle$, то есть подписанное сообщение. Получатель Y, получив такое сообщение, кодирует подпись сообщения открытым ключом 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 Y незаметно для последнего. Тогда, перехватив сообщение X, в котором сообщался открытый ключ e, злоумышленник Z может подменить открытый ключ X своим собственным открытым ключом. После этого злоумышленник сможет фальсифицировать все сообщения X подписывая их своей цифровой подписью, и, таким образом, действовать от имени 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, Y сначала расшифровывает его своим закрытым ключом, а потом убеждается в подлинности полученного сообщения, сравнив его с результатом применения открытого ключа X к подписи C.

ЗАМЕЧАНИЕ

К сожалению, даже эти меры не смогут защитить от злоумышленника Z, сумевшего подменить открытый ключ X. Конечно, в этом случае Z не сможет дешифровать исходное сообщение, но он сможет подменить исходное сообщение фальсифицированным.

Комментарии

Вопросы, затронутые в этой главе, очень существенны для практических информационных технологий, которые невозможны без кодирования, сжатия данных и шифрования. Разумеется, в реальных современных программах применяются более изощренные, по сравнению с описанными здесь простейшими вариантами, методы. Общие вопросы кодирования достаточно подробно описаны в [17] и [25]. По вопросам сжатия данных, помимо специальной литературы, см. [18]. Шифрованию посвящено множество специальных монографий. Лаконичное изложение основных идей можно найти в [16]. Алгоритм 5.1 общеизвестен. Прочие алгоритмы этой главы заимствованы (в переработанном виде) из [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:

$$p_a = 1/2,\ p_b = 1/4,\ p_c = 1/8,\ p_d = 1/8.$$

6.3. Показать, что для несимметричных ошибок функция

$$d_\delta(\beta', \beta'') := 2 \min_{\{\beta''' \in B^*\}} \max\left(\min_{\{E^\delta\langle\beta', \beta'''\rangle\}} |E^\delta\langle\beta', \beta'''\rangle|, \min_{\{E^\delta\langle\beta''', \beta''\rangle\}} |E^\delta\langle\beta''', \beta''\rangle| \right)$$

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 вторую 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 отношения, рассмотренным в главе 1, имеется глубокая связь — в сущности это равнообъемные понятия. Возникает естественный вопрос, почему же тогда графам оказывается столь явное предпочтение? 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 1.5.1 и далее). Однако независимое определение понятия отношения *удобнее* — введение специальных терминов и обозначений упрощает изложение теории и делает ее более понятной. То же относится и к теории графов. Стройная система специальных терминов и обозначений теории графов позволяют просто и доступно описывать сложные и тонкие вещи. Особенно важно наличие наглядной графической интерпретации понятия графа (подраздел 7.1.4). Само название «граф» подразумевает наличие графической интерпретации. Картинки позволяют сразу «усмотреть» суть дела на интуитивном уровне, дополняя и украшая утомительные рациональные 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 «графом» очень похожие, но все-таки различные объекты. Здесь используется терминология [23], 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 Кенигсбергских мостах.* Обойти все четыре части суши, пройдя по каждому мосту один раз, и вернуться в исходную точку (рис. 7.1). Эта задача была решена Эйлером[1] в 1736 году.

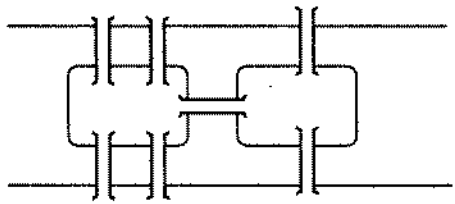


Рис. 7.1. Иллюстрация к задаче о Кенигсбергских мостах

2. *Задача о трех домах и трех колодцах.* Имеется три дома и три колодца. Провести от каждого дома к каждому колодцу тропинку так, чтобы тропинки не пересекались (рис. 7.2). Эта задача была решена Куратовским[2] в 1930 году.

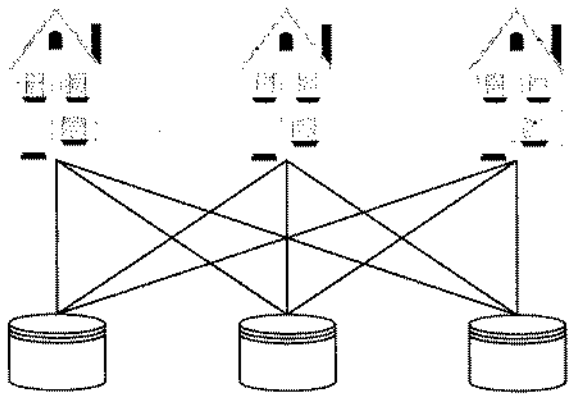


Рис. 7.2. Иллюстрация к задаче о трех домах и трех колодцах

3. *Задача о четырех красках.* Любую карту на плоскости раскрасить четырьмя красками так, чтобы никакие две соседние области не были закрашены одним цветом (рис. 7.3).

[1] Леонард Эйлер (1707–1783)
[2] Куратовский (1896–1979)

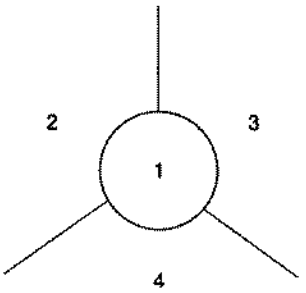


Рис. 7.3. Иллюстрация к задаче о четырех красках

7.1.2. Основное определение

Графом $G(V, E)$ называется совокупность двух множеств — непустого множества V (множества *вершин*) и множества E неупорядоченных пар различных элементов множества V (E — множество *ребер*):

$$G(V,E) = \langle V; E\rangle, \quad V \neq \varnothing, \quad E \subset V \times V, \quad E = E^{-1}.$$

Число вершин графа G обозначим p, а число ребер — q:

$$p := p(G) := |V|, \quad q := q(G) := |E|.$$

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*, вершина v_2 и ребро e 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* вершины v и обозначается $\Gamma^+(v)$:

$$\Gamma^+(v) := \{u \in V \mid (u,v) \in E\}, \quad \Gamma(v) := \Gamma^*(v) := \Gamma^+(v) \cup \{v\},$$
$$u \in \Gamma(v) \Longleftrightarrow v \in \Gamma(u).$$

ЗАМЕЧАНИЕ

Если не оговорено противное, то подразумевается Γ^+ и обозначается просто Γ.

Если $A \subset V$ — множество вершин, то $\Gamma(A)$ — множество всех вершин, смежных с вершинами из A:

$$\Gamma(A) := \{u \in V \mid \exists v \in A \; u \in \Gamma(v)\} = \bigcup_{v \in A} \Gamma(v).$$

7.1.4. Диаграммы

Обычно граф изображают *диаграммой*: вершины — точками (или кружками), ребра — линиями.

Пример

На рис. 7.4 приведен пример диаграммы графа, имеющего четыре вершины и пять ребер. 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 ребра: 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 ребра: e_1 и e_3, e_2 и e_4.

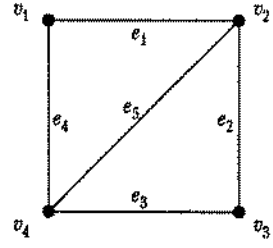


Рис. 7.4. Диаграмма графа

7.1.5. Другие определения

Часто рассматриваются следующие родственные графам объекты.

1. Если элементами множества E являются *упорядоченные* пары, 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 *набором*, содержащим несколько одинаковых элементов, то эти элементы называются *кратными ребрами*, а граф называется *мультиграфом*.
4. Если элементами множества E являются не обязательно двухэлементные, а *любые* 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.

Далее выражение «граф $G(V, E)$» означает неориентированный непомеченный граф без петель и кратных ребер.

7.1.6. Изоморфизм графов

Говорят, что два графа $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ *изоморфны* (обозначается $G_1 \sim G_2$), если существует биекция $h\colon V_1 \to V_2$, сохраняющая смежность:

$$e_1 = (u, v) \in E_1 \Longrightarrow e_2 = \big(h(u), h(v)\big) \in E_2,$$
$$e_2 = (u, v) \in E_2 \Longrightarrow e_1 = \big(h^{-1}(u), h^{-1}(v)\big) \in E_1.$$

Изоморфизм графов есть отношение эквивалентности. Действительно, изоморфизм обладает всеми необходимыми свойствами:

1. рефлексивность: $G \sim G$, где требуемая биекция суть тождественная функция;
2. симметричность: если $G_1 \sim G_2$ с биекцией h, то $G_2 \sim G_1$ с биекцией h^{-1};
3. 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. раздел 2.2).

Пример

Три внешне различные диаграммы, приведенные на рис. 7.5, являются диаграммами одного и того же графа $K_{3,3}$.

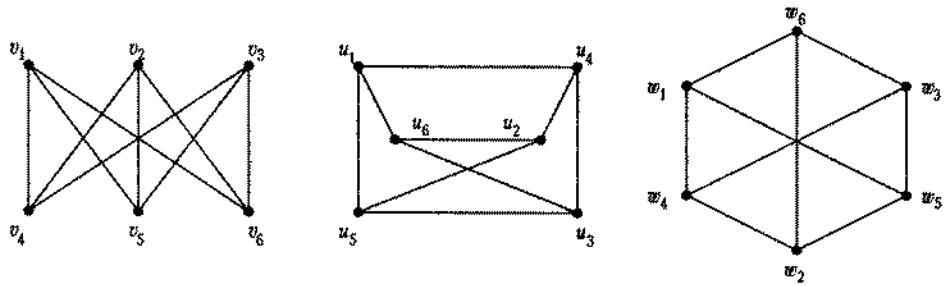


Рис. 7.5. Диаграммы изоморфных графов

Числовая характеристика, одинаковая для всех изоморфных графов, называется *инвариантом* графа. Так, $p(G)$ и $q(G)$ — инварианты графа G.

Не известно никакого набора инвариантов, определяющих граф с точностью до изоморфизма.

Пример

Количество вершин, ребер и количество смежных вершин для каждой вершины не определяют граф. На рис. 7.6 представлены диаграммы графов, у которых указанные инварианты совпадают, но графы при этом не изоморфны.

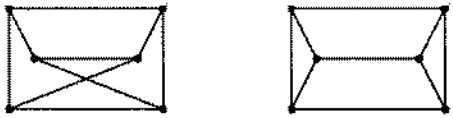


Рис. 7.6. Диаграммы неизоморфных графов с совпадающими инвариантами

7.2. Элементы графов

После рассмотрения определений, относящихся к графам как к цельным объектам, естественно дать определения различным составным элементам графов.

7.2.1. Подграфы

Граф $G'(V', E')$ называется *подграфом* графа $G(V, E)$ (обозначается $G' \subset G$), если $V' \subset V$ и/или $E' \subset E$.

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 ребра 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'.

7.2.2. Валентность

Количество ребер, инцидентных вершине v, называется *степенью* (или *валентностью*) вершины v и обозначается $d(v)$:

$$\forall v \in V \ 0 \leqslant d(v) \leqslant p - 1, \qquad d(v) = |\Gamma(v)|.$$

Обозначим *минимальную* степень вершины графа G через $\delta(G)$, а *максимальную* — через $\Delta(G)$:

$$\delta\left(G(V, E)\right) := \min_{v \in V} d(v), \qquad \Delta\left(G(V, E)\right) := \max_{v \in V} d(v).$$

Если степени всех вершин равны k, то граф называется *регулярным* степени k:

$$\delta(G) = \Delta(G) = k.$$

Степень регулярности является инвариантом графа и обозначается $r(G)$. Для нерегулярных графов $r(G)$ не определено.

Пример

На рис. 7.7 приведена диаграмма регулярного графа степени 3.

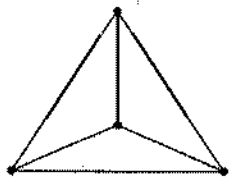


Рис. 7.7. Диаграмма регулярного графа степени 3

Если степень вершины равна 0 (то есть $d(v) = 0$), то вершина называется *изолированной*. Если степень вершины равна 1 (то есть $d(v) = 1$), то вершина называется *концевой*, или *висячей*.

Для орграфа число дуг, исходящих из вершины v, называется *полустепенью исхода*, а входящих — *полустепенью захода*. Обозначаются эти числа, соответственно, $d^-(v)$ и $d^+(v)$.

ТЕОРЕМА (Эйлера) *Сумма степеней вершин графа равна удвоенному количеству ребер:*

$$\sum_{v \in V} d(v) = 2q, \qquad \sum_{v \in V} d^-(v) + \sum_{v \in V} d^+(v) = 2q$$

Доказательство

При подсчете суммы степеней вершин каждое ребро учитывается два раза: для одного конца ребра и для другого. □

7.2.3. Маршруты, цепи, циклы

Маршрутом в графе называется чередующаяся последовательность вершин и ребер

$$v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k,$$

в которой любые два соседних элемента инцидентны.

ЗАМЕЧАНИЕ

Это определение подходит также для псевдо-, мульти- и орграфов. Для «обычного» графа достаточно указать только последовательность вершин или только последовательность ребер.

Если $v_0 = v_k$, то маршрут *замкнут*, иначе *открыт*.

Если все ребра различны, то маршрут называется *цепью*. Если все вершины (а значит, и ребра) 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 $\langle u, v \rangle$. Очевидно, что если есть 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*.

Для орграфов цепь называется *путем*, а цикл — *контуром*.

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 не простой цикл;
5. v_1, v_3, v_4 — простой цикл.

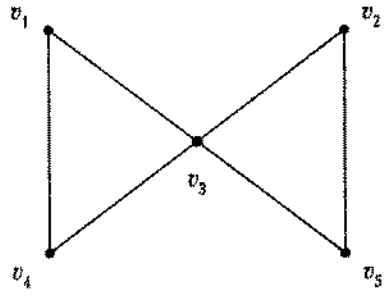


Рис. 7.8. Маршруты, цепи, циклы

7.2.4. Расстояние между вершинами

Длиной маршрута называется количество ребер в нем (с повторениями). 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*.

ЗАМЕЧАНИЕ

Если $\neg\exists\ \langle u, v\rangle$, то по определению $d(u, v) := +\infty$.

Диаметром графа G (обозначается $D(G)$) называется длина длиннейшей геодезической.

Множество вершин, находящихся на одинаковом расстоянии n от вершины v (обозначение $D(v, n)$), называется *ярусом*:

$$D(v, n) := \{u \in V \mid d(v, u) = n\}.$$

7.2.5. Связность

Говорят, что две вершины в графе *связаны*, если существует соединяющая их (простая) цепь. Граф, в котором все вершины связаны, называется *связным*.

Отношение связанности вершин является эквивалентностью. Классы эквивалентности по отношению связанности называются *компонентами связности* графа. Число компонент 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*.

7.3. 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 из используемых обозначений операций над графами являются традиционными и общепринятыми.

7.3.1. Тривиальные и полные графы

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 каждая пара вершин смежна, называется *полным*. Полный граф с p вершинами обозначается K_p, он имеет максимально возможное число ребер:

$$q(K_p) = \frac{p(p-1)}{2}.$$

Полный подграф (некоторого графа) называется *кликой* (этого графа).

7.3.2. Двудольные графы

Двудольный граф (или *биграф*, или *четный граф*) — это граф $G(V, E)$, такой что множество V разбито на два непересекающихся множества V_1 и V_2 ($V_1 \cup V_2 = V \,\&\, V_1 \cap V_2 = \varnothing$), причем 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 ребра, 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. 7.9 приведена диаграмма графа $K_{3,3}$.

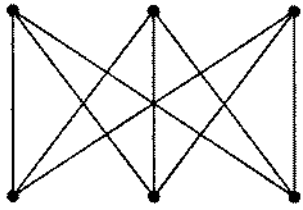


Рис. 7.9. Диаграмма графа $K_{3,3}$

ТЕОРЕМА *Граф является двудольным тогда и только тогда, когда все его простые циклы имеют четную длину.*

Доказательство

Необходимость. От противного.

Пусть $G(V_1, V_2; E)$ — двудольный граф, и $v_1, v_2, \ldots, v_{2k+1}, v_1$ — простой цикл нечетной 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.

Достаточность. Не ограничивая общности, можно считать, что G — связный граф, поскольку каждую компоненту связности можно рассматривать отдельно. Разобьем множество V на 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) — четно then
    V1 := V1 ∪ {u} { в первую долю }
  else
    V2 := V2 ∪ {u} { во вторую долю }
  end if
end for
```

Далее от противного. Пусть есть две вершины в одной доле, соединенные ребром. Пусть для определенности $u, w \in V_2$ 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 $|\langle v, w\rangle|$ нечетны. Эти геодезические имеют общие вершины (по крайней мере, вершину v). Рассмотрим наиболее удаленную от v общую вершину геодезических $\langle v, u\rangle$ и $\langle v, w\rangle$ и обозначим ее v' (может оказаться так, что $v = v'$). Имеем: $|\langle v', u\rangle| + |\langle v', w\rangle| = |\langle v, u\rangle| + |\langle v, w\rangle| - 2|\langle v, v'\rangle|$ — четно, и $v', \ldots, u, w, \ldots, v'$ — простой цикл нечетной длины, что противоречит условию. Если же $u, w \in V_1$, то

длины $|\langle v,u\rangle|$ и $|\langle v,w\rangle|$ четны и аналогично имеем: $v',\ldots,u,w,\ldots,v'$ — простой цикл нечетной длины. □

7.3.3. Направленные орграфы и сети

Если в графе ориентировать все ребра, то получится орграф, который называется *направленным*. Направленный орграф, полученный из полного графа, называется *турниром*.

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 побежденным. В таком случае турнир в смысле теории графов — это как раз результат однокругового турнира в спортивном смысле.

Если в орграфе полустепень захода некоторой вершины равна нулю (то есть $d^+(v)=0$), то такая вершина называется *источником*, если же нулю равна полустепень исхода (то есть $d^-(v)=0$), то вершина называется *стоком*. Направленный орграф с одним источником и одним стоком называется *сетью*.

7.3.4. Операции над графами

Введем следующие операции над графами:

1. *Дополнением графа* $G(V_1,E_1)$ (обозначение — $\overline{G}_1(V_1,E_1)$) называется граф $G(V_2,E_2)$, где
$$V_2:=V_1 \,\&\, E_2:=\overline{E}_1:=\{e\in V_1\times V_1 \mid e\notin E_1\}.$$

2. *Объединением 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 $V_1\cap V_2=\varnothing$, $E_1\cap E_2=\varnothing$) 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 $V_1\cap V_2=\varnothing$, $E_1\cap E_2=\varnothing$) 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

$K_{m,n}=\overline{K}_m+\overline{K}_n$.

4. *Удаление вершины* v из графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) - v$, при условии $v \in V_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := V_1 \setminus \{v\} \,\&\, E_2 := E_1 \setminus \{e = (v_1, v_2) \mid v_1 = v \vee v_2 = v\}.$$

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 $e \in E_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := V_1 \,\&\, E_2 := E_1 \setminus \{e\}.$$

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 $v \notin V_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := V_1 \cup \{v\} \,\&\, E_2 := E_1.$$

Пример

$G + v = G \cup K_1$

7. *Добавление ребра* e в граф $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) + e$, при условии $e \notin E_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := V_1 \,\&\, E_2 := E_1 \cup \{e\}.$$

8. *Стягивание подграфа* A графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1)/A$, при условии $A \subset V_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := (V_1 \setminus A) \cup \{v\} \,\&$$
$$E_2 := E_1 \setminus \{e = (u, w) \mid u \in A \vee w \in A\} \cup \{e = (u, v) \mid u \in \Gamma(A) \setminus A\}.$$

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 $A \subset V_1$, $v \in V_1$, $v' \notin V_1$) дает граф $G_2(V_2, E_2)$, где

$$V_2 := V_1 \cup \{v'\} \,\&\, E_2 := E_1 \cup \{(v, v')\} \cup \{e = (u, v') \mid u \in \Gamma^+(v)\}.$$

Пример

$K_2 \uparrow v = C_3$.

7.4. Представление графов в ЭВМ

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, при решении конкретных задач 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 объемом 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 $n(p, q)$ — объема памяти для каждого представления. Здесь p — число вершин, а q — число ребер.

ЗАМЕЧАНИЕ

Указанные представления пригодны для графов и орграфов, а после некоторой модификации также и для псевдографов, мультиграфов и гиперграфов.

Представления иллюстрируются на конкретных примерах графа G и орграфа D, диаграммы которых представлены на рис. 7.10.

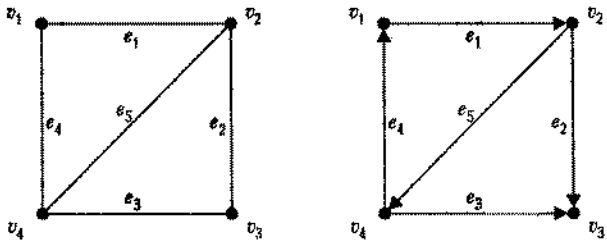


Рис. 7.10. Диаграммы графа (слева) и орграфа (спрааа), используемых в качестве примеров

7.4.2. Матрица смежности

Представление графа с помощью квадратной булевской 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

$$G \quad \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix} \qquad D \quad \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$

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 ребер, 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

$$H[i,j] = \begin{cases} 1, & \text{если вершина } v_i \text{ инцидентна ребру } e_j \text{ и является его концом,} \\ 0, & \text{если вершина } v_i \text{ и ребро } e_j \text{ не инцидентны,} \\ -1, & \text{если вершина } v_i \text{ инцидентна ребру } e_j \text{ и является его началом.} \end{cases}$$

Для матрицы инциденций $n(p,q) = O(pq)$.

Пример

$$G \quad \begin{vmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix} \qquad D \quad \begin{vmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{vmatrix}$$

7.4.4. Списки смежности

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей Γ : **array** $[1..p]$ **of** $\uparrow N$ на списки смежных вершин (элемент списка представлен структурой N : **record** $v : 1..p$; $n : \uparrow N$ **endrecord**), называется *списком смежности*. В случае представления неориентированных графов списками смежности $n(p,q) = O(p+2q)$, а в случае ориентированных графов $n(p,q) = O(p+q)$.

Пример

Списки смежности для графа G и орграфа D представлены на рис. 7.

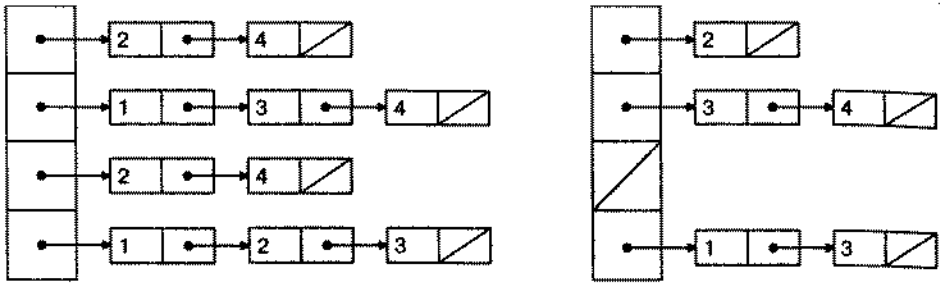


Рис. 7.11. Списки смежности для графа G (слева) и орграфа D (справа)

7.4.5. Массив дуг

Представление графа с помощью массива структур E : **array** $[1..p]$ **of record** $b, e : 1..p$ **endrecord**, отражающего список пар смежных вершин, называется *массивом ребер* (или, для орграфов, *массивом дуг*). Для массива ребер (или дуг) $n(p, q) = O(2q)$.

Пример

Представление с помощью массива ребер (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

7.4.6. Обходы графов

Обход графа — это некоторое систематическое перечисление его вершин (и/или ребер). 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 ее 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 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. Слева в таблице протокол поиска в глубину, а справа — в ширину. Предполагается, что начальной является вершина 1.

u	T	u	T
1	2,4	1	2,4
4	2,3	2	4,3
3	2	4	3
2	∅	3	∅

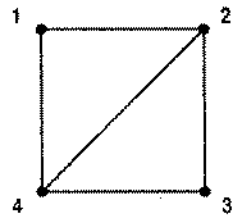


Рис. 7.12. Диаграмма графа к примеру обхода в ширину и в глубину

ТЕОРЕМА *Если граф G связен (и конечен), то поиск в ширину и поиск в глубину обойдут все вершины по одному разу.*

Доказательство

1. 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.

2. 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.
3. 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, смежные с неотмеченными, сами не отмечены (после завершения алгоритма). Но G связен, значит, существует путь (v, w). Следовательно, вершина v не отмечена. Но она была отмечена на первом шаге! □

СЛЕДСТВИЕ *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 монотонной функцией времени поиска в ширину, вершины обходятся в порядке возрастания расстояния от начальной вершины.

СЛЕДСТВИЕ *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, причем в худшем случае время поиска в глубину может быть максимальным, независимо от расстояния до начальной вершины.

СЛЕДСТВИЕ *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*

$$\forall i \geqslant 1 \quad \sum_{j=0}^{d(u_1,u_i)-1} |D(u_1, j)| \leqslant i \leqslant \sum_{j=0}^{d(u_1,u_i)} |D(u_1, j)|.$$

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 *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 этого, 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, задает 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. Гиперграф соответствует многоместному 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 базовыми средствами для построения подавляющего большинства математических моделей, используемых при решении практических задач. С другой стороны, графы допускают наглядное представление в виде диаграмм. Это обстоятельство объясняет 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.

Вершина u в орграфе $G(V, E)$ *достижима* из вершины v, если существует путь из v в u. Путь из v в u обозначим $\langle \overrightarrow{u, v} \rangle$.

Отношение достижимости можно представить матрицей T : **array** $[1..p, 1..p]$ **of** $0..1$, где $T[i, j] = 1$, если вершина v_j достижима из вершины v_i, и $T[i, j] = 0$, если вершина v_j недостижима из вершины v_i.

Рассмотрим отношение строгого частичного порядка $\succ$, которое характеризуется следующими аксиомами:

- антирефлексивность: $\forall v \in V \; \neg(v \succ v)$;
- транзитивность: $\forall u, v, w \; (v \succ w \,\&\, w \succ u \Longrightarrow v \succ u)$;
- антисимметричность: $\forall u, v \; \neg(u \succ v \,\&\, v \succ u)$.

Отношению строгого частичного порядка $\succ \subset V \times V$ можно сопоставить граф $G(V, E)$, в котором $a \succ b \Longleftrightarrow (a, b) \in E$.

ТЕОРЕМА *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 $a \succ b$, но $b \succ a$ по транзитивности, что противоречит строгости упорядочения. □

ТЕОРЕМА *Если орграф $G(V, E)$ не имеет контуров, то достижимость есть строгое частичное упорядочение.*

Доказательство

1. Нет циклов, следовательно, нет петель, следовательно, достижимость антирефлексивна.
2. Если существуют пути из v в w и из w в u, то существует и путь из v в u. Следовательно, достижимость транзитивна.
3. Пусть достижимость — это не строгое упорядочение. Тогда $\exists u, v\ u \succ v \,\&\, v \succ u$, то есть существует путь из v в u и из u в v. Следовательно, существует контур $\langle \vec{u}, \vec{v} \rangle + \langle \vec{v}, \vec{u} \rangle$, что противоречит условию. □

ТЕОРЕМА *Если орграф не имеет контуров, то в нем есть вершина, полустепень захода которой равна 0.*

Доказательство

От противного. Пусть такой вершины нет, тогда для любой вершины найдется вершина, из которой есть дуга в данную вершину. 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 1.7). А именно, минимальный элемент — это источник, то есть вершина, которой в матрице смежности соответствует нулевой столбец.

7.5.3. Транзитивное замыкание

Если E — бинарное отношение на V, то транзитивным замыканием 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 вершины v_i в вершину v_j.*

Доказательство

Индукция по k. База: $k = 1$, $M^1 = M$ — пути длины Пусть M^{k-1} содержит пути длины $k-1$. Тогда

$$M^k[i,j] = \bigvee_{l=1}^{p} M^{k-1}[i,l] \,\&\, M[l,j],$$

то есть путь длины k из вершины i в вершину j существует тогда и только тогда, когда существуют вершина l, такая что существует путь длины $k-1$ из i в l и дуга (l,j), то есть

$$\exists \langle \vec{i,j} \rangle \; |\langle \vec{i,j} \rangle| = k \Longleftrightarrow \exists l \; (\exists \langle \vec{i,l} \rangle \; |\langle \vec{i,l} \rangle| = k-1 \,\&\, (l,j) \in E). \qquad \square$$

Если T — матрица достижимости, то очевидно, что

$$T = \bigvee_{k=1}^{p} M^k.$$

Трудоемкость прямого вычисления по этой формуле составит $O(p^4)$. Матрица достижимости T может быть вычислена по матрице смежности M алгоритмом Уоршалла (алгоритм 1.8) за $O(p^3)$.

Комментарии

Эта глава является вводной главой второй части, поэтому здесь уместно привести беглый обзор учебной литературы по теории графов.

Классическими учебниками по теории графов являются книги [2] и [23]. Первая является библографической 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 [23] взяты за основу в данном учебнике. Книга [23], хотя и имеет сравнительно небольшой объем, 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: [6, 21]. Последняя книга вполне доступна даже неподготовленному читателю, имеет небольшой объем, но в то же время вполне достаточна для первоначального знакомства. Разделы, посвященные теории графов, как правило, присутствуют во всех учебниках по дискретной математике, хотя такие разделы, по естественным причинам, не отличаются полнотой охвата материала.

Особого упоминания заслуживают книги [11] и [5]. Содержание этих книг абсолютно точно соответствует их названиям и является необходимым для программиста дополнением к классическим монографиям типа [23]. Эти источники, особенно [11], в значительной мере повлияли на отбор материала для данного учебника.

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, эта идея 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, приведенные 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 нетривиальном графе существуют вершины одинаковой степени.

7.3. *Задача Рамсея.* Доказать, что среди любых 6 человек есть либо 3 попарно знакомых, либо 3 попарно незнакомых.

7.4. Рассмотрим матрицу смежности ребер 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 ребер Q представлением в ЭВМ графа $G(V,E)$?

7.5. Описать в терминах теории графов отношение эквивалентности на конечном множестве.

ГЛАВА 8 Связность

Данная глава является центральной во второй части. Здесь обсуждается важное для приложений понятие связности, доказывается наиболее глубокая теорема — теорема Менгера и рассматриваются самые распространенные 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. Современная языковая практика склоняется к использованию мужского рода, и мы ей следуем в остальной части книги. Однако исторически сложилось так, что «компонента связности» имеет женский род, и в этом случае мы подчиняемся традиции.

8.1.1. Объединение графов и компоненты связности

Напомним, что если граф G состоит из одной компоненты связности (то есть $k(G) = 1$), то он называется *связным*. В связном графе любые две вершины соединены (простой) цепью.

ТЕОРЕМА *Граф связен тогда и только тогда, когда его нельзя представить в виде объединения двух графов.*

Доказательство

Необходимость. 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 $V_1 \cap V_2 = \varnothing$, $E_1 \cap E_2 = \varnothing$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$. Возьмем $v_1 \in V_1$, $v_2 \in V_2$. 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, $e \notin E$.

Достаточность. От противного. Пусть $k(G) > 1$. Тогда $\exists u, v\ \neg\exists \langle u, v\rangle$. Следовательно, вершины u, v принадлежат разным компонентам связности. Положим G_1 — компонента связности для u, а G_2 — все остальное. Имеем $G = G_1 \cup G_2$. □

ЗАМЕЧАНИЕ

Таким образом, несвязный граф можно всегда 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

Вершина графа называется *точкой сочленения*, если ее удаление увеличивает число компонент связности.

ЛЕММА *В любом нетривиальном графе есть (по крайней мере) две вершины, которые не являются точками сочленения.*

Доказательство

Например, это концы диаметра. □

8.1.3. Оценка числа ребер через число вершин и число компонент связности

Инварианты $p(G)$, $q(G)$ и $k(G)$ не являются независимыми.

ТЕОРЕМА *Пусть p — число вершин, q — число ребер, k — число компонент связности графа. Тогда*

$$p - k \leqslant q \leqslant \frac{(p-k)(p-k+1)}{2}.$$

Доказательство

1. $p - k \leqslant q$. 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 $p - k = p' - k' \leqslant q' = q$. Если v — не изолированная вершина, то $p' = p - 1$, $k' = k$, $q' < q$. Имеем: $p - k = 1 + p' - k' \leqslant 1 + q' \leqslant q$.
2. $q \leqslant (p-k)(p-k+1)/2$. Метод выделения «критических» графов. Число ребер q графа с p вершинами и k компонентами связности не превосходит числа ребер 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 ребер имеет граф

$$K_{p-k+1} \cup \bigcup_{i=1}^{k-1} K_1.$$

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 ребер изменится на $\Delta q = p_2 - (p_1 - 1) = p_2 - p_1 + 1 > 0$. Следовательно, число ребер возросло. Таким образом, достаточно рассмотреть случай

$$K_{p-k+1} \cup \bigcup_{i=1}^{k-1} K_1.$$

Но при этом

$$q(K_{p-k+1} \cup \bigcup_{i=1}^{k-1} K_1) = (p-k+1)(p-k+1-1)/2 + 0 = (p-k)(p-k+1)/2. \quad \square$$

СЛЕДСТВИЕ *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. $\square$

8.2. Вершинная и реберная связность

При взгляде на диаграммы связных графов (сравните, например, рис. 7.9 и 8.1) возникает естественное впечатление, что связный граф может быть «более» или «менее» связен. В этом разделе рассматриваются некоторые количественные меры связности.

8.2.1. Мосты и блоки

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 $\{a, b, u\}$, $\{a, u, w\}$, $\{a, b, u, w\}$, $\{c, d, v\}$, $\{e, f, v\}$ — блоки, и других блоков нет.

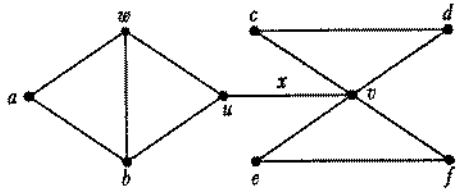


Рис. 8.1. Точки сочленения, мосты и блоки

Если в графе, отличном от K_2, есть мост, то есть и точка сочленения. Концы всякого моста (кроме висячих вершин) являются точками сочленения, но не всякая точка сочленения является концом моста.

ТЕОРЕМА *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. $\exists U, W \; U \cap W = \varnothing \,\&\, U \cup W = V \setminus \{v\} \,\&\, \forall u \in U \; \forall w \in W \; \forall \langle u, w\rangle_G \; v \in \langle u, w\rangle_G$.

Доказательство

1⟹3: Рассмотрим $G - v$. Этот граф не связен, $k(G - v) > 1$, следовательно, $G - v$ имеет (по крайней мере) две компоненты связности, то есть

$$V \setminus \{v\} = U \cup W,$$

где U — множество вершин одной из компонент связности, а W — все остальные вершины

Пусть $u \in U$, $w \in W$. Тогда $\neg\exists \langle u, w\rangle_{G-v}$. 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 $G - v$. Имеем: $\neg\exists \langle u, w\rangle_{G-v}$. Следовательно, $k(G - v) > 1$, откуда v — точка сочленения. □

ТЕОРЕМА *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 8.2.

8.2.2. 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 *k-связным*, если $\varkappa(G) = k$.

Реберной связностью графа G (обозначается $\lambda(G)$) называется наименьшее число ребер, 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 $\lambda - 1$ ребер 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 ребра, инцидентные v. Тогда $\lambda \leqslant \delta$. □

8.3. Теорема Менгера

Интуитивно очевидно, что граф тем более связен, 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.

8.3.1. 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 *реберно-непересекающимися*, если у них нет общих ребер. Если две цепи вершинно не пересекаются, то они и реберно не пересекаются. Множество вершинно-непересекающихся цепей $\langle u, v\rangle$ обозначим через $P(u, v)$.

$$P(u,v) := \{\langle u,v\rangle \mid \langle u,v\rangle_1 \in P \,\&\, \langle u,v\rangle_2 \in P \Longrightarrow \langle u,v\rangle_1 \cap \langle u,v\rangle_2 = \{u,v\}\}.$$

Множество S вершин (и/или ребер) графа G *разделяет* две вершины u и v, если u и v принадлежат разным компонентам связности графа $G - S$. *Разделяющее множество ребер* называется *разрезом*. *Разделяющее множество вершин* для вершин u и v обозначим $S(u, v)$.

$$S(u,v) := \{w \in V \mid G - S = G_1 \cup G_2, v \in G_1, u \in G_2\}.$$

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.

8.3.2. 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. 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 ребер (u, w) и (w, v). В нем $P(u, v) = \langle u, w, v\rangle$ 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 ребер меньше q. Рассмотрим граф G с p вершинами и q ребрами. Пусть $u, v \in V$, u, v — не смежны и S — некоторое наименьшее множество вершин, разделяющее u и v, $n := |S|$. 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. 8.2).

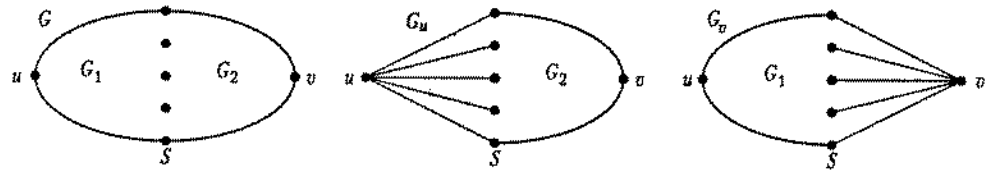


Рис. 8.2. Доказательство теоремы Менгера, случай 1

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 ребер, чем G. Следовательно, по индукционному предположению в G_u и в G_v имеется n вершинно-непересекающихся простых цепей. Комбинируя отрезки этих цепей от S до v и от u до S, получаем n вершинно-непересекающихся простых цепей в G.

2. Пусть все вершины S смежны с u или с v (для определенности пусть с u), и среди вершин S есть вершина w, смежная одновременно с u и с v (рис. 8.3).

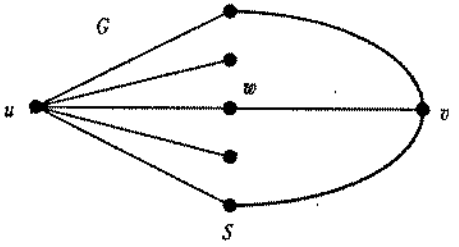


Рис. 8.3. Доказательство теоремы Менгера, случай 2

Рассмотрим граф $G' := G - w$. В нем $S \setminus \{w\}$ — разделяющее множество для u и v, причем наименьшее. По индукционному предположению в G' имеется $|S \setminus \{w\}| = n - 1$ 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 определенности 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. 8.4).

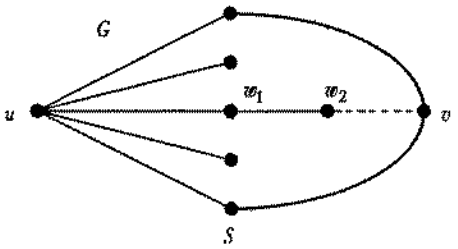


Рис. 8.4. Доказательство теоремы Менгера, случай 3

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 еще 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, имеем n вершинно-непересекающихся простых цепей в G. □

8.3.3. Варианты теоремы Менгера

Теорема Менгера представляет собой весьма общий факт, который в разных формулировках встречается в различных областях математики. Комбинируя вершинно- и реберно-непересекающиеся цепи, разделяя не отдельные вершины, а множества вершин, используя инварианты $\varkappa$ и λ, можно получить множество результатов «типа теоремы Менгера». Например:

ТЕОРЕМА *Для любых двух несмежных вершин u и v графа G наибольшее число реберно-непересекающихся $\langle u, v\rangle$-цепей равно наименьшему числу ребер в $\langle u, v\rangle$-разрезе.*

ТЕОРЕМА *Чтобы граф G был k-связным, необходимо и достаточно, чтобы любые две несмежные вершины были соединены не менее чем k вершинно-непересекающимися простыми цепями.*

Другими словами, в любом графе G любые две несмежные вершины соединены не менее чем $\varkappa(G)$ вершинно-непересекающимися простыми цепями.

8.4. Теорема Холла

Теорема Холла, устанавливаемая в этом разделе, имеет множество различных применений и интерпретаций, с рассмотрения которых мы и начинаем ее изложение.

8.4.1. Задача о свадьбах

Пусть имеется конечное множество юношей, каждый из которых знаком с некоторым подмножеством множества девушек. В каком случае всех юношей можно женить так, чтобы каждый женился на знакомой девушке?

8.4.2. Трансверсаль

Пусть $S = \{S_1, \ldots, S_m\}$ — семейство подмножеств конечного множества E. S_k не обязательно различны и могут пересекаться. *Системой различных представителей* S (или *трансверсалью* S) называется подмножество $C = \{c_1, \ldots, c_m\}$ из m элементов множества E, таких что $c_k \in S_k$. В каком случае существует трансверсаль?

ЗАМЕЧАНИЕ

C является множеством, а потому все элементы C различны, откуда и происходит название «система различных представителей».

8.4.3. Совершенное паросочетание

Паросочетанием (или *независимым множеством ребер*) называется множество ребер, в котором никакие два не смежны. Независимое множество 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.

8.4.4. Теорема Холла — формулировка и доказательство

Вообще говоря, задачи 8.4.1, 8.4.2 и 8.4.3 — это одна и та же задача. Действительно, задача 8.4.1 сводится к задаче 8.4.3 следующим образом. V_1 — множество юношей, V_2 — множество девушек, ребра — знакомства юношей с девушками. В таком случае совершенное паросочетание — искомый набор свадеб. Задача 8.4.2 сводится к задаче 8.4.3 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 8.4.1, 8.4.2 и 8.4.3 имеют общий ответ: в том и только том случае, когда

$$\text{любые } k \left(\begin{array}{c}\text{юношей}\\ \text{подмножеств}\\ \text{вершин из } V_1\end{array}\right) \text{ в совокупности } \left(\begin{array}{c}\text{знакомы с}\\ \text{содержат}\\ \text{смежны с}\end{array}\right)$$

$$\text{не менее чем } k \left(\begin{array}{c}\text{девушками}\\ \text{элементов}\\ \text{вершинами из } V_2\end{array}\right),$$

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

Необходимость. Пусть существует совершенное паросочетание из V_1 в V_2. Тогда в $\Gamma(A)$ входит $|A|$ вершин из V_2 парных к вершинам из A и, возможно, еще что-то. Таким образом, $|A| \leqslant |\Gamma(A)|$.

Достаточность. Добавим в G две новые вершины u и v, так что u смежна со всеми вершинами из V_1, а 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. 8.5). Ясно, что $|P(u, v)| \leqslant |V_1|$ (так как V_1 разделяет u и v).

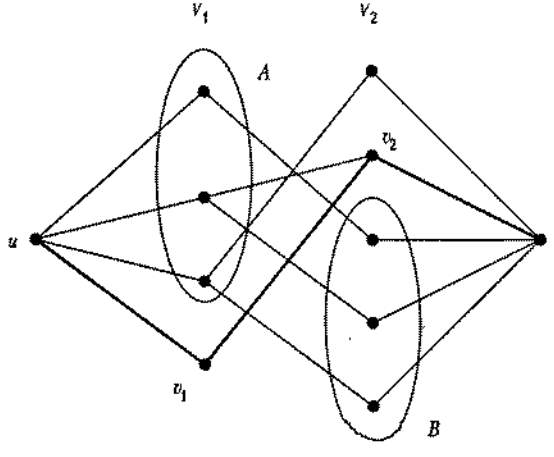


Рис. 8.5. Иллюстрация к доказательству теоремы Холла

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 (u, v_1, v_2, v), 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, $|S| = |A| + |B| \geqslant |A| + |V_1 \setminus A| = |V_1|$. □

8.5. Потоки в сетях

Рассмотрим некоторые примеры практических задач.

Пример

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 в 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 без образования пробок на дорогах (обычно это называют «автомобильным потоком»)? Или же можно поставить 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 перекачено 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 в 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, упомянутой в конце главы.

8.5.1. Определение потока

Пусть $G(V, E)$ — сеть, s и t — соответственно источник и сток сети. Дуги сети нагружены неотрицательными вещественными числами, $c\colon E \to \mathbb{R}^+$. Если u и v — вершины, 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. Элемент $C[u, v] = 0$ соответствует дуге с нулевой пропускной способностью, то есть отсутствию дуги, а элемент $C[u, v] > 0$ соответствует дуге с ненулевой пропускной способностью, то есть дуга присутствует.

Пусть задана функция $f\colon E \to \mathbb{R}$. *Дивергенцией* функции f в вершине v называется число $\mathrm{div}(f, v)$, которое определяется следующим образом:

$$\mathrm{div}(f, u) := \sum_{\{v \mid (u,v) \in E\}} f(u, v) - \sum_{\{v \mid (v,u) \in E\}} f(v, u).$$

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 вершинах, кроме источника и стока.

Число $w(f) := \mathrm{div}(f, s)$ называется *величиной* потока f.

8.5.2. 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 $C(P)$:

$$F(P) := \sum_{e \in P} f(e), \qquad C(P) := \sum_{e \in P} c(e).$$

8.5.3. Теорема Форда и Фалкерсона

ЛЕММА $w(f) = F(P^+) - F(P^-)$.

Доказательство

Рассмотрим сумму $W := \sum_{v \in S} \operatorname{div}(f, v)$. 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 $\operatorname{div}(f, u)$ и $-f(u, v)$ при вычислении $\operatorname{div}(f, v)$, 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, $W = \sum_{v \in S} \operatorname{div}(f, v) = \operatorname{div}(f, s) = w(f)$. □

ЛЕММА $\operatorname{div}(f, s) = -\operatorname{div}(f, t)$.

Доказательство

Рассмотрим разрез $P := (S, T)$, где $S := V \setminus \{t\}$, а $T := \{t\}$. Имеем:

$$\operatorname{div}(f, s) = w(f) = F(P^+) - F(P^-) = F(P^+) = \sum_v f(v, t) = -\operatorname{div}(f, t).$$ □

ЛЕММА $w(f) \leqslant F(P)$.

Доказательство

$w(f) = F(P^+) - F(P^-) \leqslant F(P^+) \leqslant F(P)$. □

ЛЕММА $\max_f w(f) \leqslant \min_P C(P)$.

Доказательство

По предыдущей лемме $w(f) \leqslant F(P)$, следовательно, $\max_f w(f) \leqslant \min_P F(P)$.

По определению $F(P) \leqslant C(P)$, следовательно, $\min_P F(P) \leqslant \min_P C(P)$.

Имеем: $\max_f w(f) \leqslant \min_P C(P)$. □

ТЕОРЕМА (Форда и Фалкерсона) *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 отменой ориентации ребер. Построим множество вершин S следующим образом:

$$S := \{u \in V \mid \exists\, \langle s, u\rangle \in G'\ \forall (u_i, u_{i+1}) \in \langle s, u\rangle \in G'$$
$$(u_i, u_{i+1}) \in E \Longrightarrow f(u_i, u_{i+1}) < C(u_i, u_{i+1})\ \&$$
$$(u_{i+1}, u_i) \in E \Longrightarrow f(u_{i+1}, u_i) > 0\},$$

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 ее R. Но тогда можно найти число δ:

$$\delta := \min_{e \in R} \Delta(e), \quad \Delta(e) := \begin{cases} c(e) - f(e) > 0, & \text{если } e \text{ ориентировано вдоль } R, \\ f(e) > 0, & \text{если } e \text{ ориентировано против } R. \end{cases}$$

В этом случае можно увеличить величину потока на δ, изменив поток для всех дуг аргументальной 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: $0 \leqslant f(e) \leqslant C(e)$, $\operatorname{div}(v) = 0$.

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. □

8.5.4. Алгоритм нахождения максимального потока

Следующий алгоритм определяет максимальный поток в сети, заданной матрицей пропускных способностей дуг. Этот алгоритм использует ту же идею доказательства теоремы Форда и Фалкерсона, а именно, задавшись начальным приближением потока, определяется 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
```

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] :=(, , ) { инициализация }
end for
S[s] := 1; P[s] :=(+, s, ∞){ 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
    end if
  end for
  if S[t] then
    x := t; δ := P[t].δ
    while x ≠ s do
      if P[x].s = + then
        F[P[x].n, x] := F[P[x].n, x] + δ
      else
        F[x, P[x].n] := F[x, P[x].n] − δ
      end if
      x := P[x].n
    end while
    goto M
  end if
until a = 0
```

Обоснование

В качестве первого приближения берется 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 вершин, достижимых из вершины s по аугментальным

цепям. При этом, если в множество S попадает вершина 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 вершин конечно, а отмеченные в массиве N вершины повторно не рассматриваются. Если процесс расширения множества S заканчивается и при этом вершина t не попадает в множество S, то по теореме Форда и Фалкерсона найденный поток F является максимальным и работа алгоритма завершается. □

ЗАМЕЧАНИЕ

Приведенный алгоритм не является самым эффективным. Более подробное изложение известных методов можно найти, например, в [14] или в [11].

8.5.5. Связь между теоремой Менгера и теоремой Форда и Фалкерсона

Теорема Менгера является фундаментальным результатом, который проявляется в различных формах (см., например, задачи 8.4.1, 8.4.2, 8.4.3). Теорема Форда и Фалкерсона также может быть получена из теоремы Менгера. Далее приведена схема неконструктивного доказательства теоремы Форда и Фалкерсона на основе теоремы Менгера. Сначала нужно получить вариант теоремы Менгера в ориентированной реберной форме: наибольшее число $\langle s,t\rangle$-путей, непересекающихся по дугам, равно наименьшему числу дуг в (s,t)-разрезе. Это теорема Форда и Фалкерсона в том случае, когда $\forall e \in E\ C(e) \equiv 1$. Действительно, пусть Q — множество дуг из максимального набора непересекающихся $\langle s,t\rangle$-путей. Назначим $f(e) := 1$, если $e \in Q$, и $f(e) := 0$, если $e \notin Q$. Это поток, так как дивергенция в любой вершине 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 $e \notin Q$, то

1. если дуга e входит в вершину, входящую в пути из Q, или выходит из такой вершины, 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 8.5.3 доказательство теоремы Форда и Фалкерсона конструктивно, из него не только следует заключение о величине максимального потока, но и извлекается способ нахождения максимального потока.

8.6. 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.

8.6.1. 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 вершин несимметрично, а потому определение связности отличается.

Пусть $G(V, E)$ — орграф, v_1 и v_2 — его вершины. Говорят, что две вершины v_1 и v_2 *сильно связаны* в орграфе G, если существует путь (ориентированная цепь) из v_1 в v_2 и из v_2 в v_1. Говорят, что две вершины 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 две вершины v_1 и v_2 *слабо связаны* в орграфе G, если они связаны в графе G', полученном из G отменой ориентации ребер. 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 неверно.

Пример

На рис. 8.6 показаны диаграммы сильно, односторонне и слабо связных орграфов.

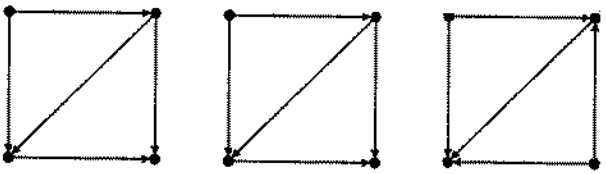


Рис. 8.6. Сильная (слева), односторонняя (в центре) и слабая (справа) связность

8.6.2. Компоненты сильной связности

Компоненты сильной связности (КСС) орграфа G — это его максимальные сильно связные подграфы.

Каждая вершина орграфа принадлежит только одной КСС. Если вершина не связана с другими, то считаем, что она сама образует КСС. *Конденсацией* G^* орграфа G (или *графом Герца*, или *фактор-графом*) называется орграф, который получается стягиванием в одну вершину каждой КСС орграфа G.

Пример

На рис. 8.7 показаны диаграммы орграфа и его конденсации.

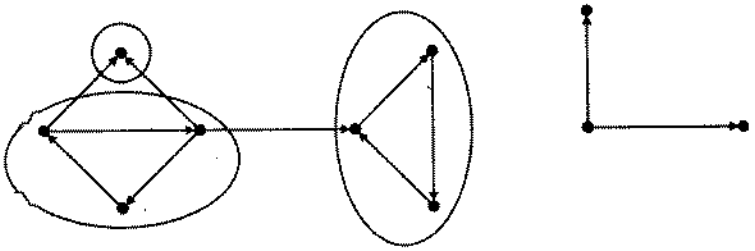


Рис. 8.7. Орграф (слева) и его фактор-граф (справа)

8.6.3. 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 вершин, входящих в компоненту сильной связности.

```
C := ∅
for v ∈ V do
  M[v] := {v} { M[v] список вершин, входящих в ту же КСС, что и v }
  e[v] := 0 { все вершины не рассмотрены }
end for
while V ≠ ∅ do
  select v ∈ V { взять v из V }
  T ← v { 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 вершин. Процедура КСС выделяет все КСС, достижимые из вершины, выбранной в основном алгоритме.

```
if T = ∅ then
  return{ негде выделять }
```

```
end if
v ← T; v → T { стоим в вершине v}
if Γ[v] ∩ V = ∅ then
  C := C ∪ M[v] { это КСС }
  V := V \ {v} { удалить вершину }
  v ← T { снять v со стека }
  КСС { возврат из тупика }
else
  for u ∈ Γ[v] do
    if e[u] = 0 then
      u → T { положить u в стек }
      e[u] := 1 { отметить u }
    else
      repeat
        w ← T { w — склеиваемая вершина }
        V := V \ {w} { удалить ее }
        Γ[u] := Γ[u] ∪ Γ[w] { запомнить смежность }
        M[u] := M[u] ∪ M[w]{ склеивание вершин }
      until u = w
      w → T{ чтобы не убрать ту вершину, }
      V := V ∪ {w}{ к которой слили }
    end if
    КСС { поиск в глубину }
  end for
end if
```

8.7. 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 два классических алгоритма, которые обязан знать каждый программист.

8.7.1. Длина дуг

Алгоритм Уоршалла позволяет ответить на вопрос, 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 помечены числами (эти числа обычно называют *весами*, или *длинами*, дуг), то этот орграф можно представить в виде матрицы весов (длин) C:

$$C[i,j] = \begin{cases} 0, & \text{для } i = j \\ c_{ij}, & \text{конечная величина, если есть дуга из } i \text{ в } j, \\ \infty, & \text{если нет дуги из } i \text{ в } j. \end{cases}$$

Длиной пути называется сумма длин дуг, входящих в этот путь. Наиболее часто на практике встречается задача отыскания *кратчайшего* пути.

8.7.2. Алгоритм Флойда

Алгоритм Флойда находит кратчайшие пути между всеми парами вершин в (ор)графе. В этом алгоритме для хранения информации о путях используется матрица $H[1..p, 1..p]$, где

$$H[i,j] = \begin{cases} k, \text{ если } k \text{ — первая вершина, достигаемая на кратчайшем пути из } i \text{ в } j; \\ 0, \text{ если из i в j нет пути.} \end{cases}$$

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 ее хранения в памяти. В данном случае любой конкретный путь $\langle u, v \rangle$ легко извлекается из матрицы с помощью следующего алгоритма.

```
w := u; yield w{ первая вершина }
while w ≠ v do
  w := H[w, v]; yield w{ следующая вершина }
end while
```

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
```

ЗАМЕЧАНИЕ

Если в G есть цикл с отрицательным весом, то решения поставленной задачи не существует, так как можно «накручивать» на этом цикле сколь угодно короткий путь.

Обоснование

Алгоритм Флойда имеет много общего с алгоритмом Уоршалла (алгоритм 1.8). 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 выдает решение, поскольку оно не всегда существует. Дополнительный цикл по j служит для прекращения работы в случае обнаружения в графе цикла с отрицательным весом. □

8.7.3. Алгоритм Дейкстры

Алгоритм Дейкстры находит кратчайший путь между двумя данными вершинами 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
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 ∈ Γ(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
t := ∞; v := 0
{ поиск конца кратчайшего пути }
for u from 1 to p do
  if X[u] = 0 & T[u] < t then
    v := u; t := 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 более слабого условия, чем положительность длин дуг. А именно, 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.

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

$$\forall u \; T[u] = 1 \Longrightarrow T[u] = d(s,u) \,\&\, T[H[u]] = 1.$$

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 $T[w] = 0$. Имеем: $T[w] = d(s,w) \leqslant d(s,v) < T[v]$, что противоречит выбору вершины v. □

ЗАМЕЧАНИЕ

Известно, что алгоритм Флойда примерно на 50% менее трудоемок, чем применение алгоритма Дейкстры для всех пар вершин.

Комментарии

Изложение центрального результата этой главы — теоремы Менгера 8.3.2 и сопутствующего материала — в основном следует в [23]. Алгоритм нахождения максимального потока в сети заимствован из [11] с небольшими модификациями. Алгоритм выделения компонент сильной связности приведен в [5], 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 [14, 11].

Упражнения

8.1. Доказать, что если $\delta(G) > (p-1)/2$, то граф G связен.

8.2. Доказать вторую теорему подраздела 8.2.1.

8.3. Доказать, что наибольшее число непересекающихся множеств вершин, разделяющих вершины u и v, равно $d(u,v) - 1$.

8.4. «Задача о гароме». 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 условия решения этой задачи.

8.5. 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

$$\forall v \in V \quad \sum_{\{u|(u,v)\in E\}} f(u,v) \leqslant D(v).$$

Найти максимальный поток в такой сети.

8.6. Как может измениться количество компонент сильной связности орграфа при добавлении к нему одной дуги?

8.7. Пусть в графе $G(V, E)$ заданы вероятности успешного прохождения дуг, $0 \leqslant P[v] \leqslant 1$. Вероятность успешного прохождения пути определяется как произведение вероятностей составляющих его дуг. Построить алгоритм нахождения наиболее надежного (то есть имеющего наибольшую вероятность) пути 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 свойства, 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 распространенным классом графов, применяемых в программировании, причем в самых разных ситуациях. Более половины объема 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 $q(G) \geqslant p(G) - 1$. Граф G, в котором $q(G) = p(G) - 1$, называется *древовидным*.

В ациклическом графе G $z(G) = 0$. Пусть u, v — несмежные вершины графа G, $x = (u, v) \notin E$. Если граф $G + x$ имеет только один простой цикл, $z(G + x) = 1$, то граф G называется *субциклическим*.

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.

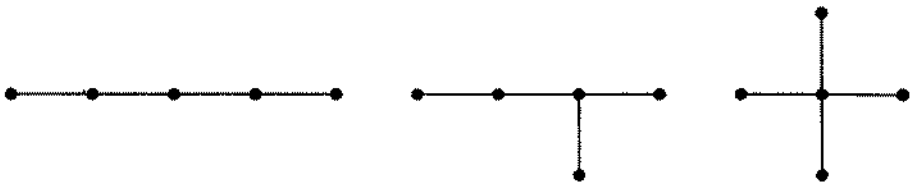


Рис. 9.1. Свободные деревья с 5 вершинами

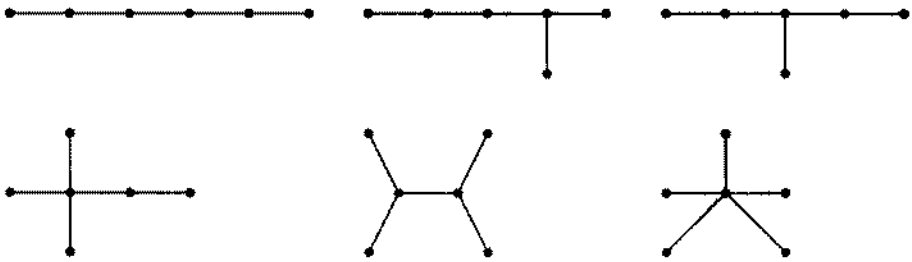


Рис. 9.2. Свободные деревья с 6 вершинами

9.1.2. Основные свойства деревьев

Следующая теорема устанавливает, что два из четырех свойств — связность, ацикличность, древовидность 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,
   $\forall u, v \; \exists! \; \langle u, v \rangle$;
3. G — связный граф, и любое ребро есть мост,
   $k(G) = 1 \,\&\, \forall e \in E \; k(G - e) > 1$;

4. G — связный и древовидный,
   $k(G) = 1 \,\&\, q(G) = p(G) - 1$;
5. G — ациклический и древовидный,
   $z(G) = 0 \,\&\, q(G) = p(G) - 1$;
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 — древовидный 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

$1 \Longrightarrow 2$: От противного. Пусть существуют две цепи $\langle u, v\rangle$ (рис. 9.3 слева). Тогда w_1, w_2 — простой цикл.

$2 \Longrightarrow 3$: Имеем: $\forall u, v\ \exists!\ \langle u, v\rangle$, 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.

$3 \Longrightarrow 4$: Индукция по p. База: $p = 1 \Longrightarrow q = 0$. Пусть $q(G) = p(G) - 1$ для всех графов G с числом вершин меньше p. Тогда удалим из G ребро x (которое является мостом). Получим две компоненты G' и G'', удовлетворяющие индукционному предположению. Имеем:

$$q' = p' - 1,\ q'' = p'' - 1,\ q = q' + q'' + 1 = p' - 1 + p'' - 1 + 1 = p - 1.$$

$4 \Longrightarrow 5$: От противного. Пусть есть цикл с n вершинами и n ребрами. Остальные $p - n$ вершин имеют инцидентные им ребра, которые связывают их с циклом. Следовательно, $q \geqslant p$, что противоречит условию $q = p - 1$.

$5 \Longrightarrow 1$: 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, $k = 1$.

$5 \Longrightarrow 6$: По ранее доказанному $5 \Longrightarrow 1 \Longrightarrow 2$. Имеем: $\forall u, v\ \exists!\ \langle u, v\rangle$. Соединив две несмежные вершины, получим единственный простой цикл.

$6 \Longrightarrow 7$: 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.

$7 \Longrightarrow 2$: Имеем $k(G) = 1$, следовательно, $\forall u, v\ \exists\ \langle u, v\rangle$. Пусть цепь не единственная. Тогда существует цикл Z, причем $Z = K_3 = C_3$. 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 с более чем одной вершиной Z, то имеем больше одного цикла. Если w смежна только с одной вершиной Z, то соединив ее с другой вершиной, получим два цикла.

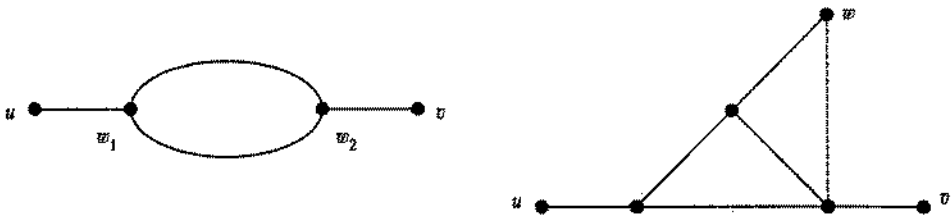


Рис. 9.3. Иллюстрации к доказательству теоремы о свойствах деревьев

$7 \Longrightarrow 8$: 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 $q = p - 1$.

$8 \Longrightarrow 5$: 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 ее 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:

$$q' = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} (p_i - 1) = \sum_{i=1}^{k} p_i - k = p' - k,$$

но $q' = p' - 1$, 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. □

СЛЕДСТВИЕ *В любом нетривиальном дереве имеются по крайней мере две висячие вершины.*

Доказательство

Рассмотрим дерево $G(V, E)$. Дерево — связный граф, следовательно,

$$\forall v_i \in V \; d(v_i) \geqslant 1.$$

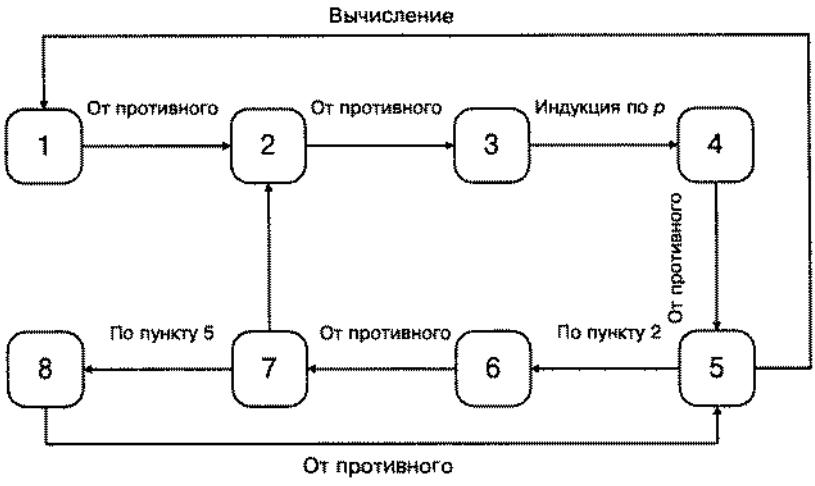


Рис. 9.4. Схема доказательства теоремы о свойствах деревьев

Далее от противного. Пусть $\forall v_i \in 1..p-1\ d(v) > 1$. 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: $2p - 2 > 2p - 1$. □

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 в программировании. Дерево (ориентированное) и иерархия — это равнообъемные понятия.

9.2.1. Ориентированные деревья

Ориентированным деревом (или *ордеревом*, или *корневым* деревом) называется орграф со следующими свойствами:

1. существует единственный узел, полустепень захода которого равна 0. Он называется *корнем* ордерева;
2. полустепень захода всех остальных узлов равна 1;
3. каждый узел достижим из корня.

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.

Рис. 9.5. Ориентированные деревья с 3 узлами

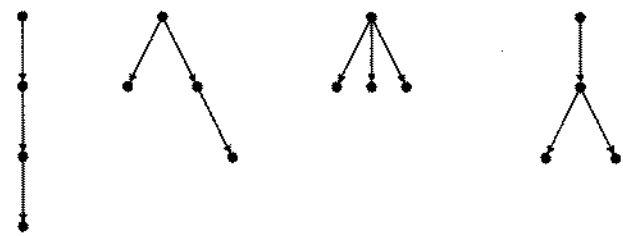


Рис. 9.6. Ориентированные деревья с 4 узлами

ТЕОРЕМА *Ордерево обладает следующими свойствами:*

1. $q = p - 1$;
2. *если в ордереве отменить ориентацию ребер, 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.*

Доказательство

1. Каждое ребро входит в какой-то узел. Из п. 3 определения 9.2.1 имеем:

$$\forall v \in V \backslash \{u\}\ d^+(v) = 1,$$

следовательно, $q = p - 1$.

2. Пусть G — ордерево, граф G' получен из G отменой ориентации ребер, u — корень.

Тогда $\forall v_1 v_2 \in V\ \exists \langle v_1, u\rangle \in G'\ \&\exists \langle u, v_2\rangle \in G'$, следовательно, $\forall v_1, v_2\ \exists (v_1, v_2)$ и граф G' связен. Таким образом, учитывая п. 4. теоремы 9.1.2, G' — дерево.

3. Следует из 2.
4. От противного. Если бы в G существовали два пути из u 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 3. Далее имеем: $\forall v' \in G_v \backslash \{v\}\ d^+(v') = 1$, так как $G_v \subset G$. Все вершины G_v достижимы из v по построению. По определению 9.2.1 получаем, что G_v — ордерево.
6. Пусть вершина u назначена корнем и дуги последовательно ориентированы «от корня» обходом в глубину.

   Тогда $d^+(u) = 0$ по построению; $\forall v \in V \backslash \{u\}\ d^+(v) = 1$, 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. □

ЗАМЕЧАНИЕ

Каждое свободное дерево определяет не более p 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*. Путь из корня в лист называется *ветвью*. Длина наибольшей ветви ордерева называется *высотой*. *Уровень* узла ордерева — это расстояние от корня до узла. Сам корень имеет уровень 0. Узлы одного уровня образуют *ярус* дерева.

ЗАМЕЧАНИЕ

Наряду с «растительной» применяется еще и «генеалогическая» терминология. Узлы, достижимые из узла u, называются *потомками* узла u (потомки образуют поддерево). 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 узла называются *братьями*.

9.2.2. Эквивалентное определенне ордерева

Ордерево T — это конечное множество узлов, таких что:

1. имеется один узел r, называемый корнем данного дерева;
2. остальные узлы (исключая корень) содержатся в k попарно непересекающихся множествах $T_1, \ldots, T_k$, каждое из которых является ордеревом ($k \geqslant 0$).

$$T = \{\{r\}, T_1, \ldots, T_k\}.$$

9.2.3. Упорядоченные деревья

Множества $T_1, \ldots, T_k$ в экивалентном определении ордерева являются поддеревьями. Если относительный порядок поддеревьев $T_1, \ldots, T_k$ фиксирован, то ордерево называется *упорядоченным*.

Пример

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 слева.
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 в центре показана структура областей определения идентификаторов a, b, c, d, e, причем для отображения структуры дерева использована альтернативная техника.
3. Для представленния иерархической структуры вложенности элементов данных и/или операторов управления часто используется техника отступов, показанная на рис. 9.7 справа.

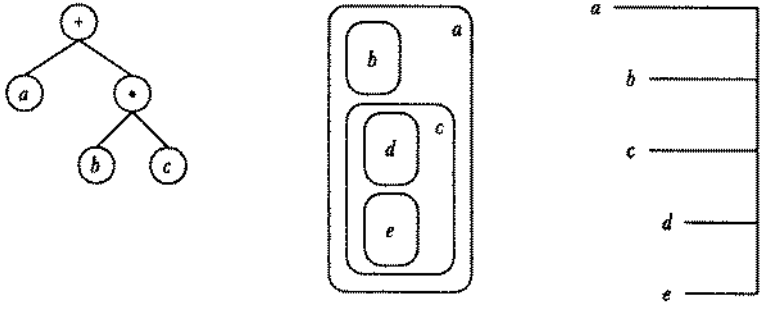


Рис. 9.7. Примеры изображения деревьев в программировании

4. Структура вложенности каталогов и файлов в современных операционных системах является упорядоченным ориентированным деревом.

ОТСТУПЛЕНИЕ

Это отражается даже в терминологии — «корневой каталог диска».

5. Различные «уравновешенные скобочные структуры» (например $(a(b)(c(d)(e)))$) являются ориентированными упорядоченными деревьями.

ЗАМЕЧАНИЕ

Общепринятой практикой при изображении деревьев является соглашение о том, что корень находится наверху и все стрелки дуг ориентированы сверху вниз, поэтому стрелки можно не изображать. Таким образом, диаграммы свободных, ориентированных и упорядоченных деревьев оказываются графически неотличимыми, и требуется дополнительное указание, 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. Обозначим дерево слева — (1), в центре — (2) и справа — (3). Как упорядоченные деревья, они 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).

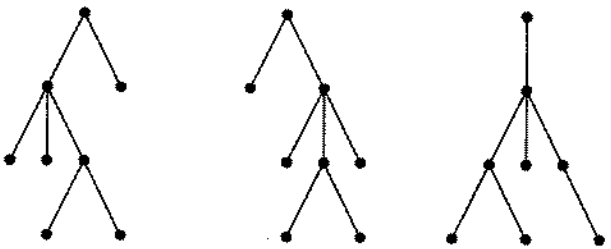


Рис. 9.8. Диаграммы деревьев

9.2.4. Бинарные деревья

Бинарное дерево — это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев — левого и правого.

Бинарное дерево *не является* упорядоченным ордеревом.

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

9.3. Представление деревьев в ЭВМ

Обсуждению представлений деревьев можно предпослать в точности те же рассуждения, что были предпосланы обсуждению представлений графов (см. раздел 7.4). Кроме того, следует подчеркнуть, что задача представления деревьев в программе встречается гораздо чаще, чем задача представления графов общего вида, а потому методы ее решения оказывают еще большее влияние на практику программирования.

9.3.1. Представление свободных, ориентированных и упорядоченных деревьев

Всякое свободное дерево можно ориентировать, назначив один из узлов корнем. Всякое ордерево можно произвольно упорядочить. Всякое упорядоченное дерево можно представить бинарным деревом, проведя правую связь к старшему брату, а левую — к младшему сыну.

Пример

На рис. 9.10 приведены диаграммы упорядоченного и соответствующего ему бинарного деревьев.

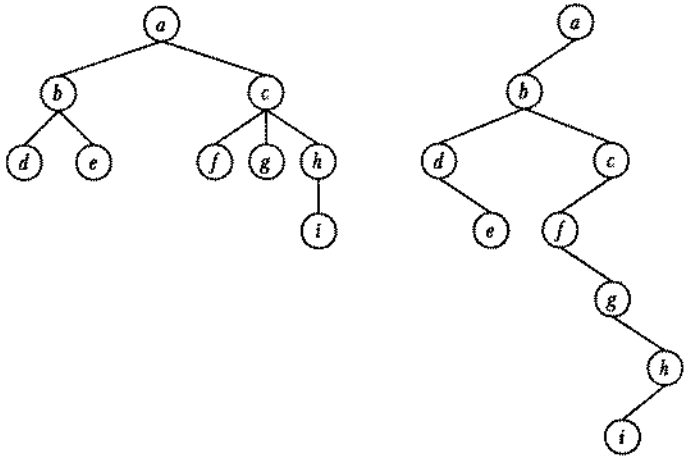


Рис. 9.10. Упорядоченное и бинарное деревья

Таким образом, достаточно рассмотреть представление в ЭВМ бинарных деревьев.

ЗАМЕЧАНИЕ

Из данного представления следует, что множество бинарных деревьев взаимнооднозначно соответствует множеству упорядоченных лесов упорядоченных деревьев.

9.3.2. Представление бинарных деревьев

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**. 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 последним узлом поддерева данного узла. Дерево T обычно определяется следующим образом: T : **array** $[1..p]$ **of record** $i : info$, $k : 1..p$ **end record**. 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**. 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 $n(p) = p$.

9.3.3. Обходы бинарных деревьев

Большинство алгоритмов работы с деревьями основаны на обходах. Возможны следующие основные обходы бинарных деревьев:

Прямой (левый) обход:	попасть в корень, обойти левое поддерево, обойти правое поддерево.
Обратный (симметричный) обход:	обойти левое поддерево, попасть в корень, обойти правое поддерево.
Концевой (правый) обход:	обойти левое поддерево, обойти правое поддерево, попасть в корень.

Кроме трех основных, возможны еще три соответствующих обхода, отличающихся порядком рассмотрения левых и правых поддеревьев. Этим исчерпываются обходы, если в представлении фиксированы только связи «отец-сын».

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.

9.3.4. Алгоритм симметричного обхода бинарного дерева

Реализация обходов бинарного дерева с помощью рекурсивной процедуры не вызывает затруднений. В некоторых случаях из соображений эффективности применение явной рекурсии оказывается нежелательным. Следующий очевидный алгоритм реализует наиболее популярный симметричный обход без рекурсии, но с использованием стека.

Алгоритм 9.1. 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 : { анализирует 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. 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 вполне 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.

Пример

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 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 всех последующих записей на одну позицию вперед.

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 приемом программирования, подробное описание которого выходит за рамки этого учебника. 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, имея ключ, 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 приемами 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 9.2. 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

Для обоснования этого алгоритма 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 трудоемкость 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 9.3. Поиск узла в дереве сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ a.

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
```

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 9.4. Вставка узла в дерево сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ a.
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 }
M : { анализ текущего узла }
if a < p.i then
  if p.l = nil then
    q := NewNode(a) { создаем новый узел }
    p.l := q { и подцепляем его к p слева }
    return T
  else
    p := p.l { продолжаем поиск места для вставки слева }
    goto M
  end if
end if
if a > p.i then
  if p.l = nil then
    q := NewNode(a) { создаем новый узел }
    p.r := q { и подцепляем его к p справа }
    return T
  else
    p := p.r { продолжаем поиск места для вставки справа }
    goto M
  end if
end if
return T { сюда попали, если уже есть такой ключ! }
```

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. □

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 9.5. Удаление узла из дерева сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ a.
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, см. рис. 9.11 слева }
else
  u := p.r
  if u.l = nil then
    u.l := p.l
    Delete(p, q, u, s) { случай 2, см. рис. 9.11 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, см. рис. 9.11 справа }
  end if
end if
return T
```

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. 9.11 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 ведет в узел u, левая связь которого пуста (см. рис. 9.11 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.

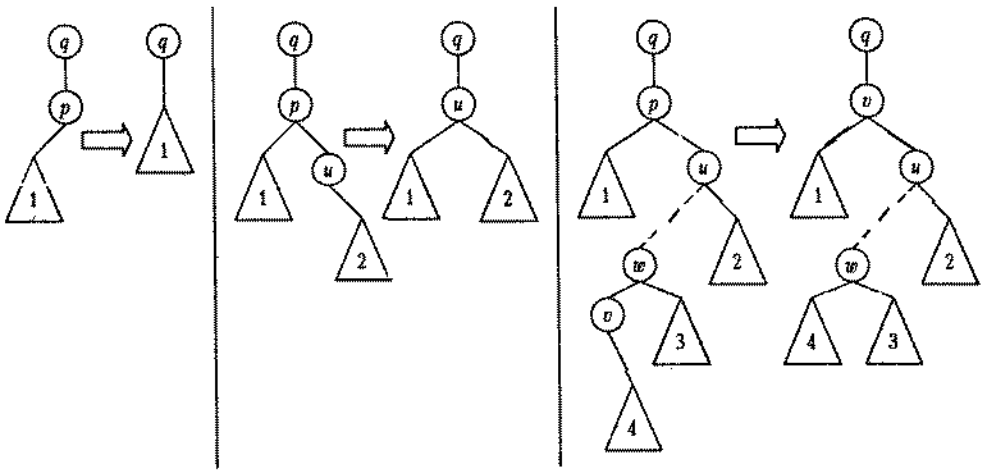


Рис. 9.11. Иллюстрация к алгоритму удаления узла из дерева сортировки

3. Правая связь удаляемого узла p не пуста и ведет 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. 9.11 справа). В этом случае выполняются два преобразования дерева. Сначала информация в узле p заменяется на информацию 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 [8], 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.
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
```

ОТСТУПЛЕНИЕ

В этой простой функции стоит обратить внимание на использование указателя q, который отслеживает значение указателя p «с запаздыванием», то есть указатель q «помнит» предыдущее значение указателя p. Такой прием 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 будет даже менее эффективным, чем неупорядоченный массив.

9.4.9. Выровненные деревья

Ордерево называется *выровненным*, если все узлы, степень которых меньше 2, располагаются на одном или двух последних уровнях. Выровненное дерево имеет наименьшую возможную для данного p высоту h.

Пример

На рис. 9.12 приведены диаграммы выровненного (слева) и невыровненного (справа) деревьев.

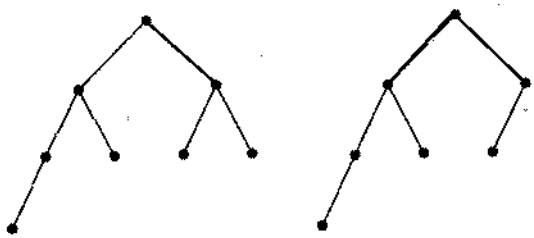


Рис. 9.12. Выровненное (слева) и невыровненное деревья

ЛЕММА $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$.

Доказательство

Индукция по k. База: $k = 0 \Longrightarrow 2^0 = 1, 2^1 - 1 = 1, 1 = 1$.

Пусть $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$, тогда

$$\sum_{i=0}^{k+1} 2^i = \sum_{i=0}^{k} 2^i + 2^{k+1} = 2^{k+1} - 1 + 2^{k+1} = 2^{(k+1)+1} - 1.$$

□

ТЕОРЕМА *Для выровненного 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 имеем: $2^h - 1 < p \leqslant 2^{h+1} - 1$ и $2^h < p+1 \leqslant 2^{h+1}$. Логарифмируя, имеем: $h < \log_2(p+1)$ & $h + 1 \geqslant \log_2(p+1)$. Следовательно,

$$\log_2(p+1) - 1 \leqslant h < \log_2(p+1). \qquad \square$$

ЗАМЕЧАНИЕ

Выровненные деревья дают наибольший возможный эффект при поиске. Однако известно, что вставка/удаление может потребовать полной перестройки всего дерева и, таким образом, трудоемкость операции в худшем случае составит $O(p)$.

9.4.10. Сбалансированные деревья

(Бинарное) дерево называется *подровненным деревом*, или *АВЛ-деревом* (Адельсон-Вельский и Ландис, 1962), или *сбалансированным деревом*, если для любого узла высота левого и правого поддеревьев отличается не более чем на 1.

Пример

На рис. 9.13 приведена диаграмма максимально несимметричного сбалансированного дерева, в котором для всех узлов высота левого поддерева ровно на 1 больше высоты правого поддерева.

ТЕОРЕМА *Для подровненного бинарного дерева* $h < 2\log_2 p$.

Доказательство

Рассмотрим наиболее несимметричные АВЛ-деревья, скажем, такие, у которых всякое левое поддерево на 1 выше правого. Такие АВЛ-деревья имеют максимальную возможную высоту среди всех АВЛ-деревьев с заданным числом вершин. Пусть P_h — число вершин в наиболее несимметричном АВЛ-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: $P_0 = 1$, $(\sqrt{2})^0 = 1$, $1 \geqslant 1$; $P_1 = 2$ $(\sqrt{2})^1 = \sqrt{2}$, $2 \geqslant \sqrt{2}$.

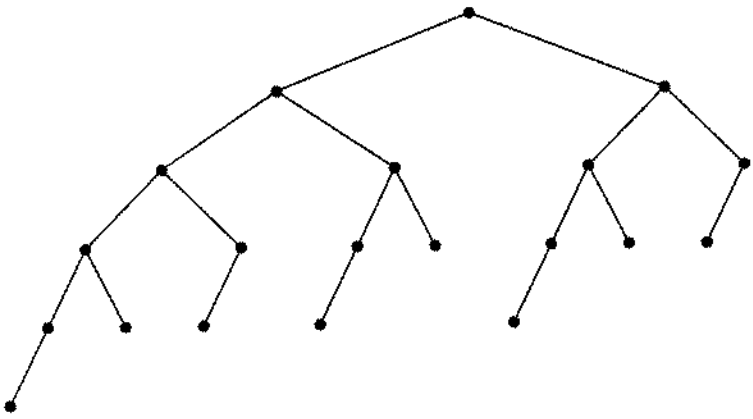


Рис. 9.13. Сбалансированное дерево

Пусть $P_h \geqslant (\sqrt{2})^h$. Тогда

$$P_{h+1} = P_h + P_{h-1} + 1 \geqslant (\sqrt{2})^h + (\sqrt{2})^{h-1} + 1 =$$
$$= (\sqrt{2})^h(1 + \frac{1}{(\sqrt{2})} + \frac{1}{(\sqrt{2})^h}) > (\sqrt{2})^h(1 + \frac{1}{(\sqrt{2})}) > (\sqrt{2})^h\sqrt{2} = (\sqrt{2})^{h+1}.$$

Имеем: $(\sqrt{2})^h = 2^{h/2} \leqslant P_h = p$, следовательно, $\frac{h}{2} \leqslant \log_2 p$ и $h \leqslant 2\log_2 p$. □

ЗАМЕЧАНИЕ

Известна более точная оценка высоты АВЛ-дерева: $h < \log_{\sqrt{5}+1/2} 2\log_2 p$.

Сбалансированные деревья уступают выровненным деревьям по скорости поиска (менее чем в два раза), однако их преимущество состоит в том, что известны алгоритмы вставки и удаления узлов в сбалансированное дерево, которые сохраняют сбалансированность и в то же время при перестройке дерева затрагивают только конечное число узлов (см., например, [13]). Поэтому в подавляющем большинстве случаев АВЛ-дерево оказывается наилучшим вариантом представления дерева сортировки.

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, описанного в подразделе 9.5.3, привели в конечном счете к теории жадных алгоритмов, изложенной в разделе 2.7.5.

9.5.1. Определения

Пусть $G(V, E)$ — граф. *Остовной подграф* графа $G(V, E)$ — это подграф, содержащий все вершины. Остовный подграф, являющийся деревом, называется *остовом*.

ЗАМЕЧАНИЕ

Остов определяется множеством ребер, поскольку вершины остова суть вершины графа.

Несвязный граф не имеет остова. Связный граф может иметь много остовов.

ОТСТУПЛЕНИЕ

Если задать длины ребер, 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 ребрам возврата). Множество кратчайших путей из заданной вершины ко всем остальным также образует остов. Однако этот остов может не быть кратчайшим.

Пример

На рис. 9.14 показаны диаграммы (слева направо) графа, дерева кратчайших путей из вершины 1 с суммарным весом 5 и два кратчайших остова этого графа.

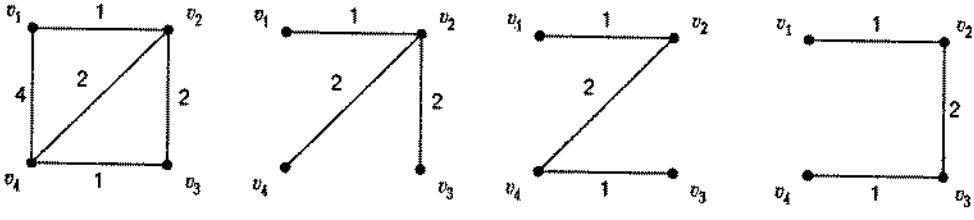


Рис. 9.14. Граф, дерево кратчайших путей и два кратчайших остова

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 9.6. Построение кратчайшего остова
Вход: граф $G(V, E)$, заданный матрицей длин ребер 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
```

ТЕОРЕМА *Алгоритм 9.6 строит кратчайший остов.*

Доказательство

Пусть T_i и T_j — два произвольных поддерева G, $T_i \cap T_j \neq 0$. 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, то $\Delta_{i,j}$ всегда реализуется на некотором ребре с длиной $c_{i,j}$. Далее индукцией по шагам алгоритма 9.6 покажем, что все ребра, включенные в T, принадлежат кратчайшему остову — SST [1]. Вначале выбранных ребер нет, поэтому их множество включается в кратчайший остов. Пусть теперь все ребра, 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, причем более коротким, чем SST, что противоречит выбору SST. □

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 жадный алгоритм, известный как *алгоритм Краскала*, находит кратчайший остов в связном графе.

Алгоритм 9.7. Алгоритм Краскала
Вход: список E ребер графа G с длинами.
Выход: множество T ребер кратчайшего остова.

```
T := ∅
упорядочить E в порядке возрастания длин
k := 1 { номер рассматриваемого ребра }
```

[1] SST — Shortest Sceleton Tree — стандартное обозначение для кратчайшего остова.

```
for i from 1 to p − 1 do
  while добавление ребра E(k) образует цикл в T do
    k := k + 1 { пропустить это ребро }
  end while
  T := T ∪ {E[k]} { добавить это ребро в SST }
end for
```

Обоснование

Заметим, что множество подмножеств множества ребер, не содержащих циклов, образует матроид (см. раздел 2.7). Действительно, если множество ребер не содержит цикла, то любое подмножество также не содержит цикла. Пусть теперь $E' \subset E$ — произвольное множество ребер, а G' — правильный подграф графа G, определяемый этими ребрами. Очевидно, что любое максимальное не содержащее циклов подмножество множества E' является объединением остовов компонент связности G' и по теореме об основных свойствах деревьев (см. подраздел 9.1.2) содержит $p(G') - k(G')$ элементов. Таким образом, по теореме подраздела 2.7.2 множество ациклических подмножеств ребер образует матроид. Далее, рассматриваемый алгоритм, как жадный алгоритм, находит кратчайшее ациклическое подмножество множества ребер. По построению оно содержит $p - 1$ элемент, а значит, является искомым остовом. □

ОТСТУПЛЕНИЕ

Задача о нахождении кратчайшего остова принадлежит к числу немногих задач теории графов, которые можно считать полностью решенными. 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*. На рис. 9.15 приведены, соответственно, диаграммы кратчайшего остова, наивного «решения» задачи Штейнера и правильного решения для случая, когда города расположены в вершинах квадрата.

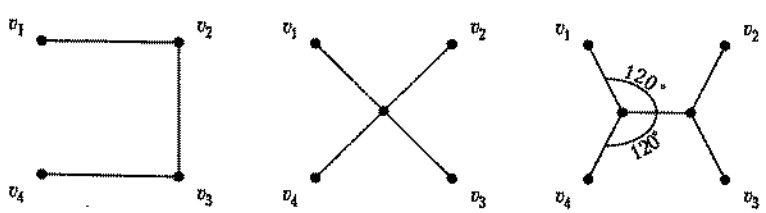


Рис. 9.15. Кратчайший остов, приближенное и точное решение задачи Штейнера

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 программированию и конструированию эффективных алгоритмов. В качестве рекомендуемых источников назовем [1, 8, 13]. В частности, алгоритмы операций с деревом сортировки описаны в [8], откуда они заимствованы с некоторыми дополнительными уточнениями способов реализации. В книге [13] можно найти краткие и доступные описания алгоритмов работы с АВЛ-деревьями, которые здесь опущены за недостатком места.

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*).

ГЛАВА 10 Циклы

После рассмотрения ациклических связных графов, то есть деревьев, естественно перейти к рассмотрению графов с циклами.

10.1. Фундаментальные циклы и разрезы

Первый раздел главы посвящен установлению связи векторных пространств со структурой циклов и разрезов в графе.

10.1.1. Циклы и коциклы

Цикл может входить только в одну компоненту связности графа, поэтому далее без ограничения общности граф считается связным. Цикл (простой) рассматривается как множество ребер.

Разрезом связного графа называется множество ребер, удаление которых делает граф несвязным. *Простым* разрезом называется минимальный разрез, то есть такой, никакое собственное подмножество которого разрезом не является.

В этом параграфе рассматриваются только простые циклы и разрезы, далее слово «простые» опускается.

Между циклами и разрезами существует определенная двойственность, поэтому разрезы иногда называют *коциклами*.

ЗАМЕЧАНИЕ

Поскольку рассматриваются только простые циклы и коциклы, они могут считаться именно множествами, а не последовательностями ребер.

ОТСТУПЛЕНИЕ

Чем больше в графе циклов, тем труднее его разрезать. В дереве, напротив, каждое ребро само по себе является разрезом.

10.1.2. Независимые множества циклов и коциклов

Рассмотрим операцию $\oplus$ сложения по модулю 2 или симметрической разности над множествами ребер:

$$M_1 \oplus M_2 := \{e \mid (e \in M_1 \,\&\, e \notin M_2) \vee (e \notin M_1 \,\&\, e \in M_2)\}.$$

Множество M называется *зависимым* или *линейной комбинацией* множеств $\{M_i\}_{i=1}^n$, если

$$M = \bigoplus_{i=1}^{n} M_i.$$

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.

ЗАМЕЧАНИЕ

Множество подмножеств ребер данного графа образует векторное пространство над двоичной арифметикой. Действительно, $\oplus$ — ассоциативная и коммутативная операция. Далее $M \oplus \varnothing = \varnothing \oplus M = M$ и каждый элемент является своим обратным: $M \oplus M = \varnothing$. Таким образом, подмножества ребер образуют абелеву группу относительно симметричной разности. Далее определим операцию умножения вектора на скаляр: $0M := \varnothing$, $1M := M$. Легко видеть, что аксиомы векторного пространства выполнены. Введенные здесь понятия линейной комбинации, зависимости и независимости являются частными случаями одноименных понятий из разделов 2.5 и 2.7.

10.1.3. Циклический и коциклический ранг

Максимальное независимое множество циклов (или минимальное множество циклов, от которых зависят все остальные) 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 $m(G)$. Максимальное независимое множество коциклов (разрезов) (или минимальное множество коциклов, от которых зависят все остальные) называется *фундаментальной системой коциклов (разрезов)*. Коциклы (разрезы) фундаментальной системы называются *фундаментальными*, а количество коциклов в (данной) фундаментальной системе называется *коциклическим рангом*, или *коцикломатическим числом*, графа G и обозначается $m^*(G)$.

Пусть $T(V, E_T)$ — остов графа $G(V, T)$. *Кодеревом* $T^*(V, E_T^*)$ остова T называется остовный подграф, такой что $E_T^* = E \setminus E_T$. (Кодерево не является деревом!) Ребра кодерева называются *хордами* остова.

ТЕОРЕМА *Если G — связный граф, то*

$$m(G) = q - p + 1, \qquad m^*(G) = p - 1.$$

Доказательство

Рассмотрим некоторый остов T графа G. По теореме 9.1.2 об основных свойствах деревьев каждая хорда $e \in T^*$ остова T порождает ровно один простой цикл z_e. Эти циклы независимы, так как каждый из них содержит свое индивидуальное ребро (хорду e).

Покажем, что любой цикл графа G зависит от $\{Z_e\}_{e\in T^*}$. Заметим, что любой элемент фундаментальной системы зависит от фундаментальной системы, поэтому далее элементы фундаментальной системы не рассматриваем, то есть $Z \neq Z_e$.

Пусть теперь некоторый цикл Z содержит ребра $e_1, \ldots, e_k \in T^*$. Такие ребра в Z обязательно есть, в противном случае $Z \subset T$, что невозможно, так как T — дерево. Докажем индукцией по k, что $Z = Z_{e_1} \oplus \cdots \oplus Z_{e_k}$.

База: пусть $k = 1$, тогда $e_1 \notin T$, $Z\backslash\{e_1\} \subset T$ 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) $Z = Z_{e_1} \oplus \cdots \oplus Z_{e_m}$ для всех циклов Z с числом хорд $m < k$. Рассмотрим цикл Z с k хордами $e_1, \ldots, e_k \in T^*$. Рассмотрим Z_{e_k}. Имеем $Z' := (Z\backslash\{e_k\}) \cup (Z_{e_k}\backslash\{e_k\})$ — тоже цикл (возможно, не простой). Но Z' содержит только $k-1$ хорду $e_1, \ldots, e_{k-1}$. По индукционному предположению $Z' = Z_{e_1} \oplus \cdots \oplus Z_{e_{k-1}}$. Добавим к этому циклу Z_{e_k}. Имеем:

$$Z' \oplus Z_{e_k} = (Z\backslash\{e_k\}) \cup ((Z_{e_k}\backslash\{e_k\}) \oplus Z_{e_k}) = (Z\backslash\{e_k\}) \cup \{e_k\} = Z.$$

Таким образом, $\{Z_e\}_{e\in T^*}$ является фундаментальной системой циклов. Поскольку все фундаментальные системы содержат одинаковое количество элементов (как базисы векторного пространства), достаточно ограничиться рассмотрением любой, например, той, которая определяется остовом T. Пусть теперь $e \in T$. Определим разрез S_e следующим образом. Ребро e — мост в дереве T. Следовательно, удаление ребра e разбивает множество вершин V на два подмножества V_1 и V_2, так что

$$V_1 \subset V, \quad V_2 \subset V, \quad V_1 \cup V_2 = V, \quad V_1 \cap V_2 = \varnothing.$$

Включим в разрез S_e ребро e и все ребра графа G, которые соединяют вершины множества V_1 с вершинами множества V_2. Тогда S_e — это разрез, потому что правильные подграфы, определяемые V_1 и V_2, являются компонентами связности $G - S_e$. Разрез S_e — простой, потому что, если есть ребро, соединяющее V_1 и V_2, то граф связен.

Аналогично можно показать, что $\{S_e\}_{e\in T}$ является фундаментальной системой разрезов. Действительно, любой разрез S содержит хотя бы одно ребро из T, так как T — связный остовной подграф. Разрезы $\{S_e\}_{e\in T}$ независимы, так как каждый содержит уникальное ребро e. Любой разрез S зависит от $\{S_e\}_{e\in T}$, $S = \bigoplus_{e\in T} S_e$. Далее имеем $m^*(G) = |\{S_e\}_{e\in T}| = |E_T| = p - 1$ и

$$m(G) = |\{S_e\}_{e\in T^*}| = |E_T^*| = |E| - |E_T| = q - (p-1) = q - p + 1. \qquad \square$$

10.2. Эйлеровы циклы

Здесь приведено исчерпывающее решение задачи о Кенигсбергских 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 ребра графа по одному разу, то такой цикл называется *эйлеровым* циклом, а граф называется *эйлеровым* графом. Если граф имеет цепь (не обязательно простую), содержащую все вершины по одному разу, то такая цепь называется *эйлеровой* цепью, а граф называется *полуэйлеровым* графом.

Эйлеров цикл содержит не только все ребра (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 четную степень;*
3. *множество ребер G можно разбить на простые циклы.*

Доказательство

$1 \Longrightarrow 2$ Пусть Z — эйлеров цикл. 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 ребра, то когда обход Z будет закончен, будут учтены все ребра, а степени всех вершин — четные.

$2 \Longrightarrow 3$ G — связный и нетривиальный граф, следовательно, $\forall v\ d(v) > 0$. Степени вершин четные, следовательно, $\forall v\ d(v) \geqslant 2$. Имеем:

$$2q = \sum d(v) \geqslant 2p \Longrightarrow q \geqslant p \Longrightarrow q > p - 1.$$

Следовательно, граф G — не дерево, а значит, граф G содержит (хотя бы один) простой цикл Z_1. (Z_1 — множество ребер.) Тогда $G - Z_1$ — остовный подграф, в котором опять все степени вершин четные. 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, 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 ребра 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. □

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 необходимо и достаточно убедиться, что степени всех вершин четные, что нетрудно сделать при любом представлении графа. Следующий алгоритм находит эйлеров цикл в графе, если известно, что граф эйлеров.

Алгоритм 10.1. Алгоритм построения 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 ← S; v → S { v — верхний элемент стека }
  if Γ[v] = ∅ then
    v ← S; yield v
  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. Начиная с произвольной вершины, строим путь, удаляя ребра 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 придем в ту вершину, с которой начали. В противном случае это означало бы, что вершина v имеет нечетную степень, что невозможно по условию. Таким образом, из графа были удалены ребра цикла, а вершины цикла были сохранены в стеке S. Заметим, что при этом степени всех вершин остались четными. Далее вершина v выводится в качестве первой вершины эйлерова цикла, а процесс продолжается, начиная с вершины, стоящей на вершине стека. □

10.2.3. Оценка числа эйлеровых графов

ЛЕММА *В любом графе число вершин нечетной степени четно.*

Доказательство

По теореме Эйлера $\sum d(v) = 2q$, то есть сумма степеней всех вершин — четное число. Сумма степеней вершин четной степени четна, значит, сумма степеней вершин нечетной степени также четна, значит, их четное число. □

Пусть $\mathcal{G}(p)$ — множество всех графов с p вершинами, а $\mathcal{E}(p)$ — множество эйлеровых графов с p вершинами.

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 четными 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 нечетной степени четно, следовательно, любой граф из $\mathcal{E}'(p)$ можно получить из некоторого графа $\mathcal{G}(p-1)$, если добавить новую вершину и соединить ее со всеми старыми вершинами нечетной степени. Следовательно, $|\mathcal{E}'(p)| \leqslant |\mathcal{G}(p-1)|$. Но $|\mathcal{G}(p)| = 2^{C(p,2)}$. Заметим, что

$$C(k,2) - C(k-1,2) = \frac{k!}{2!(k-2)!} - \frac{(k-1)!}{2!(k-3)!} =$$
$$= \frac{k(k-1)}{2} - \frac{(k-1)(k-2)}{2} = \frac{(k-1)(k-k+2)}{2} = k-1.$$

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 прошлом веке: нужно обойти все вершины графа, диаграмма которого показана на рис. 10.1 (в исходной формулировке это были названия столиц различных стран), по одному разу и вернуться в исходную точку. Этот граф представляет собой укладку додекаэдра.

[1] Вильям Гамильтон (1805–1856)

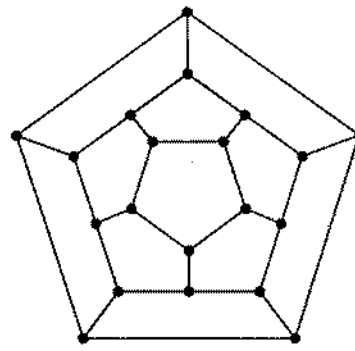


Рис. 10.1. Задача «Кругосветное путешествие»

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 ребра графа. Ясно, что гамильтоновым может быть только связный граф.

ЗАМЕЧАНИЕ

Любой граф G можно превратить в гамильтонов, добавив достаточное количество вершин. Для этого, например, достаточно к вершинам $v_1, \ldots, v_p$ графа G добавить вершины $u_1, \ldots, u_p$ и множество ребер $\{(v_i, u_i)\} \cup \{(u_i, v_{i+1})\}$.

ТЕОРЕМА (Дирак) *Если в графе* $G(V, E)$ $\forall v \in V\ d(v) \geqslant p/2$, *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 $G' := G + u_1 + \cdots + u_n$ был гамильтоновым.

Пусть $v, u_1, w, \ldots, v$ — гамильтонов цикл в графе G', причем $v \in G$, $u_1 \in G'$, $u_1 \notin G$. 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 несмежна 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' несмежна 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) 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 коммивояжера

Рассмотрим следующую задачу, известную как *задача коммивояжера*. Имеется p городов, расстояния между которыми известны. Коммивояжер 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 коммивояжера — это задача отыскания кратчайшего гамильтонова цикла в полном графе.

Можно предложить следующую простую схему решения задачи коммивояжера: сгенерировать все $p!$ возможных перестановок вершин полного графа, подсчитать для каждой перестановки длину маршрута и выбрать из них кратчайший. Очевидно, такое вычисление потребует $O(p!)$ шагов.

Как известно, $p!$ с ростом p растет быстрее, чем любой полином от p, и даже быстрее, чем 2^p (см. раздел 5.2). Таким образом, решение задачи коммивояжера описанным методом *полного перебора* оказывается практически неосуществимым даже для сравнительно небольших p. Более того, известно, что задача коммивояжера принадлежит к числу так называемых *NP-полных* задач, подробное обсуждение которых выходит за рамки этого учебника. Вкратце суть проблемы NP-полноты сводится 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 трудоемкость. Более того, если бы удалось отыскать эффективный 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 не известно необходимых и достаточных условий. 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 (также как и задача коммивояжера). 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

$$\lim_{p\to\infty} \frac{|H(p)|}{|G(p)|} = 1,$$

где $H(p)$ — множество гамильтоновых графов с p вершинами, а $G(p)$ — множество всех графов с p вершинами. Таким образом, задача отыскания гамильтонова цикла или эквивалентная задача коммивояжера являются практически востребованными, но для них неизвестен (и, скорее всего, не существует) эффективный алгоритм решения.

Комментарии

Из вопросов, рассмотренных в этой главе, наибольшее внимание в литературе уделено задаче коммивояжера. Анализ различных подходов к ее решению см., например, в [11]. Алгоритм 10.1 описан в [14], в других источниках (например в [6]) можно найти другие варианты решения этой задачи.

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 $G(V, E)$ $\forall u, v \in V$ $d(u) + d(v) \geqslant p$, то граф G гамильтонов.

ГЛАВА 11 Независимость и покрытия

В этой главе рассматриваются некоторые известные задачи теории графов и указываются связи между ними. Особое внимание уделяется обсуждению методов решения переборных задач на примере задачи отыскания максимального независимого множества вершин.

11.1. Независимые и покрывающие множества

Прежде всего введем определения и рассмотрим основные свойства независимых и покрывающих множеств вершин и ребер. Эти определения и свойства используются в последующих разделах.

11.1.1. Покрывающие множества вершин и ребер

Говорят, что вершина *покрывает* инцидентные ей ребра, а ребро *покрывает* инцидентные ему вершины.

Множество таких вершин, которые покрывают все ребра, называется *вершинным покрытием*. Наименьшее число вершин во всех вершинных покрытиях называется *числом вершинного покрытия* и обозначается α_0.

Пример

1. Для полного графа $\alpha_0(K_p) = p - 1$.
2. Для полного двудольного графа $\alpha_0(K_{m,n}) = \min(m, n)$.
3. Для вполне несвязного графа $\alpha_0(\overline{K}_p) = 0$.
4. Для несвязного графа $\alpha_0(G_1 \cup G_2) = \alpha_0(G_1) + \alpha_0(G_2)$.

Множество таких ребер, которые покрывают все вершины, называется *реберным покрытием*. Наименьшее число ребер во всех реберных покрытиях называется *числом реберного покрытия* и обозначается α_1.

ЗАМЕЧАНИЕ

Для графа с изолированными вершинами α_1 не определено.

Пример

1. Для четного цикла $\alpha_1(K_{2n}) = n$, для нечетного цикла $\alpha_1(K_{2n+1}) = n + 1$.
2. Для полного графа с четным числом вершин $\alpha_1(K_{2n}) = n$, для полного графа с нечетным числом вершин $\alpha_1(K_{2n+1}) = n + 1$.
3. Для полного двудольного графа $\alpha_1(K_{m,n}) = \max(m, n)$.

11.1.2. Независимые множества вершин и ребер

Множество вершин называется *независимым*, если никакие две из них не смежны. Наибольшее число вершин в независимом множестве вершин называется *вершинным числом независимости* и обозначается β_0.

Пример

1. Для полного графа $\beta_0(K_p) = 1$.
2. Для полного двудольного графа $\beta_0(K_{m,n}) = \max(m, n)$.
3. Для вполне несвязного графа $\beta_0(\overline{K}_p) = p$.
4. Для несвязного графа $\beta_0(G_1 \cup G_2) = \beta_0(G_1) + \beta_0(G_2)$.

Множество ребер называется *независимым*, если никакие два из них не смежны. Наибольшее число ребер в независимом множестве ребер называется *реберным числом независимости* и обозначается β_1.

ЗАМЕЧАНИЕ

Независимое множество ребер называется также *паросочетанием*.

Пример

1. Для четного цикла $\beta_1(K_{2n}) = n$, для нечетного цикла $\beta_1(K_{2n+1}) = n - 1$.
2. Для полного графа с четным числом вершин $\beta_1(K_{2n}) = n$, для полного графа с нечетным числом вершин $\beta_1(K_{2n+1}) = n - 1$.
3. Для полного двудольного графа $\beta_1(K_{m,n}) = \min(m, n)$.

11.1.3. Связь чисел независимости и покрытий

Приведенные 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

Покажем, что имеют место 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 ребер, инцидентных только вершинам из N_0, стало быть, любое ребро инцидентно вершине (или вершинам) из $V \setminus M_0$. 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 реберное 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 звезд. (*Звездой* называется граф, диаметр которого не превосходит двух, $D(G) \leqslant 2$.) Пусть этих звезд m. Имеем $|M_1| = \sum_{i=1}^{m} n_i$, где n_i — число ребер в i-й звезде. Заметим, что звезда из n_i ребер покрывает $n_i + 1$ вершину. Имеем: $p = \sum_{i=1}^{m}(n_i + 1) = m + \sum_{i=1}^{m} n_i = m + |M_1|$. Возьмем 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 ребер, то есть $|N_1| = \beta_1$. Построим реберное покрытие Y следующим образом. Множество N_1 покрывает $2|N_1|$ вершин. Добавим по одному ребру, инцидентному непокрытым $p - 2|N_1|$ вершинам, таких ребер $p - 2|N_1|$. Тогда множество Y — реберное покрытие, то есть $|Y| \leqslant \alpha_1$ 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. Однако это условие является достаточным, но 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 есть множество процессов, использующих неразделяемые ресурсы. Соединим ребрами вершины, соответствующие процессам, которым требуется один и тот же ресурс. Тогда β_0 будет определять количество возможных параллельных процессов.

11.2. Построение независимых множеств вершин

Данный раздел фактически является вводным обзором методов решения переборных задач. Методы рассматриваются на примере задачи отыскания максимального независимого множества вершин графа. Данный обзор не претендует на полноту, но описание основополагающих идей и терминов в нем присутствует.

11.2.1. 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 трудоемких.

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 $w(X) := |X|$, а $\mathcal{E} := \{Y \subset V \mid \forall u, v \in Y \ (u, v) \notin E\}$ (см. раздел 2.7.5).

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. подраздел 2.7.1) 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.

11.2.2. Поиск с возвратами

Даже если для решения задач, подобных поставленной в предыдущем разделе, не удается найти эффективного алгоритма, всегда остается возможность 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 ∈ Ɛ & |Y| > m then
    m := |Y|; X := Y { наилучшее известное значение X }
  end if
end for
```

ЗАМЕЧАНИЕ

Для выполнения этого алгоритма потребуется $O(2^n)$ шагов.

ОТСТУПЛЕНИЕ

Алгоритм, трудоемкость которого (число шагов) ограничена полиномом от характерного размера задачи, принято называть *эффективным*, в противоположность *неэффективным* алгоритмам, трудоемкость которых ограничена более быстро растущей функцией, 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* (от английского названия этого метода — backtracking).

Идея поиска с возвратами состоит в следующем. Находясь в некоторой ситуации, пробуем изменить ее в надежде найти решение. Если изменение не привело к успеху, то возвращаемся в исходную ситуацию (отсюда название «поиск с возвратами») и пробуем изменить ее другим образом, и так до тех пор, пока не будут исчерпаны все возможности.

Для рассматриваемой задачи метод поиска с возвратами может быть реализован с помощью следующего рекурсивного алгоритма.

Алгоритм 11.1. Поиск с возвратами
Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0 { наилучшее известное значение β0 }
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).

```
e := false { признак расширяемости множества }
for v ∈ V do
  if S ∪ {v} ∈ E then
    e := true { множество можно расширить }
    BT(S ∪ {v}, T \ Γ*(v)) { пробуем добавить v }
  end if
end for
if e then
  if |S| > m then
    X := S; m := |S| { наибольшее независимое множество }
  end if
end if
```

Обоснование

По построению вершина v добавляется в множество S только при сохранении независимости расширенного множества. В алгоритме это обстоятельство указано в форме условия $S \cup \{v\} \in \mathcal{E}$. Проверить сохранение условия независимости нетрудно, например, с помощью следующего цикла.

```
f := true { множество S независимое }
for u ∈ S do
  if (u, v) ∈ E then
    f := false; exit for { множество S ∪ {v} зависимое }
  end if
end for
```

Этот цикл не включен в явном виде в рекурсивную процедуру BT, чтобы не загромождать основной текст и не затуманивать основную идею поиска с возвратами. Таким образом, множество S, а следовательно, и множество X — независимые. В тот момент, когда множество S нельзя расширить (e = **false**), оно максимально по определению. Переменная m глобальна, поэтому среди всех максимальных независимых множеств в конце работы алгоритма X является наибольшим независимым множеством вершин. □

11.2.3. Улучшенный перебор

Применение метода поиска с возвратами не гарантирует эффективности — трудоемкость 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 трудоемкость выполнения каждого шага перебора или уменьшить количество перебираемых возможностей в среднем (при сохранении оценки количества шагов в худшем случае). Такие приемы 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 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 $x \in Q_k^+$:

$$S_{k+1} = S_k \cup \{x\},$$
$$Q_{k+1}^- = Q_k^- - \Gamma(x),$$
$$Q_{k+1}^+ = Q_k^+ - (\Gamma(x) \cup \{x\}).$$

Шаг назад от $k+1$ к k:

$$S_k = S_{k+1} - \{x\},$$
$$Q_k^+ = Q_k^+ - \{x\},$$
$$Q_k^- = Q_k^- \cup \{x\}.$$

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, $k \leqslant p-1$.

11.2.4. Алгоритм построения максимальных независимых множеств вершин

Приведенный ниже алгоритм, обоснование которого дано в предыдущем разделе, строит все максимальные независимые множества вершин заданного графа.

Алгоритм 11.2. 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, 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 }
$v := \text{last}(S[k])$ { 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 завершен }
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.

11.3. Доминирующие множества

Задача о наименьшем покрытии (сокращенно 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 трудоемких задач.

В этом разделе на основе рассмотрения понятия доминирующего множества ЗНП формулируется в различных вариантах, и приводятся сведения о связи ЗНП с другими задачами.

11.3.1. Определения

Для орграфа $G(V, E)$ множество вершин $S \subset V$ называется *доминирующим множеством*, если $S \cup \Gamma(S) = V$, то есть для любой вершины $v \in V$ либо $v \in S$, либо существуют вершина $s \in S$ и ребро (s, v).

Доминирующее множество называется *минимальным*, если его подмножество не является доминирующим. Доминирующее множество называется *наименьшим*, если число элементов в нем минимально.

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.

11.3.2. Доминирование и независимость

Доминирование тесно связано с вершинной независимостью.

ТЕОРЕМА *Независимое множество вершин является максимальным тогда и только тогда, когда оно является доминирующим.*

Доказательство

Необходимость. Пусть множество вершин S ($S \subset V$) — 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.

Достаточность. 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.

11.3.3. 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* (сокращенно ЗНП).

ЗНП является весьма общей задачей, к которой сводятся многие другие задачи.

Пример

1. *Задача о выборе переводчиков.* Организации нужно нанять переводчиков для перевода с определенного множества языков. Каждый из имеющихся переводчиков владеет некоторыми иностранными языками и требует определенную зарплату. Требуется определить, каких переводчиков следует нанять, чтобы сумма расходов на зарплату была минимальной.
2. *Задача о развозке.* Поставщику нужно доставить товары своим потребителям. Имеется множество возможных маршрутов, каждый из которых позволяет обслужить определенное подмножество потребителей и требует определенных расходов. Требуется определить, какие маршруты следует использовать, чтобы все потребители были обслужены, а сумма транспортных расходов была минимальной.

11.3.4. Эквивалентные формулировки ЗНП

ЗНП может быть сформулирована многими разными способами.

1. 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.

ЗАМЕЧАНИЕ

Из этой формулировки происходит название «задача о наименьшем покрытии».

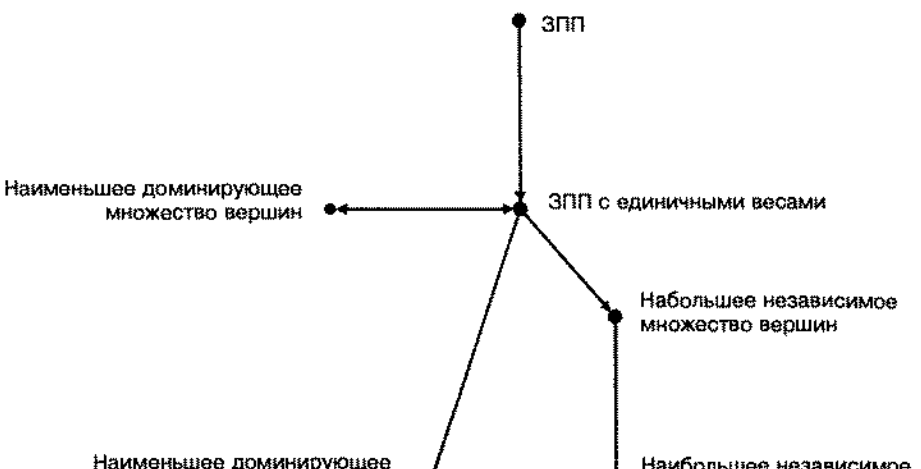


Рис. 11.1. Связь различных задач

2. ЗНП можно сформулировать как задачу линейного программирования:

$$\min \sum_{i=1}^{p} c_i x_i$$

при ограничениях

$$\sum_{j=1}^{p} t_{ij} x_j \geqslant 1 \qquad (i = 1, \ldots, p),$$

где

$$c_i \geqslant 0, \quad x_i = \begin{cases} 1, & \text{если } E_i \in E', \\ 0, & \text{в противном случае,} \end{cases}$$

$$t_{ij} = \begin{cases} 1, & \text{если } v_i \in E_j, \\ 0, & \text{в противном случае.} \end{cases}$$

3. Дана булева матрица T : **array** $[1..p, 1..p]$ **of** 0..1. Каждому столбцу приписан вес c_j. Найти такое множество столбцов минимальной стоимости, чтобы любая строка содержала единицу хотя бы в одном из выбранных столбцов.

11.3.5. Связь ЗНП с другими задачами

Известно, что ЗНП относится к числу трудоемких задач, и для ее решения применяются переборные алгоритмы с теми или иными улучшениями.

На рис. 11.1 приведена схема (см. [11]), 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.

Комментарии

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 [4]. Алгоритм построения максимальных независимых множеств заимствован из [11]. Здесь этот алгоритм использован прежде всего в качестве примера для иллюстрации способов и особенностей решения переборных задач.

Упражнения

11.1. Доказать, что $\alpha_0 \geqslant \beta_1, \alpha_1 \geqslant \beta_0$.

11.2. Написать алгоритм построения всех клик графа.

11.3. Доказать, что наименьшее доминирующее множество является независимым.

ГЛАВА 12 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, [7]).

12.1. Хроматическое число

Способ явного выражения хроматического числа через другие инварианты графа неизвестен. Известны только некоторые оценки, часть из которых приведена в данном разделе.

Раскраской графа называется такое приписывание цветов (натуральных чисел) его вершинам, что никакие две смежные вершины не получают одинаковый цвет. Наименьшее возможное число цветов в раскраске называется *хроматическим числом* и обозначается $\chi(G)$. 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.

Пример

$\chi(\overline{K}_p) = 1,\ \chi(K_p) = p,\ \chi(K_{m,n}) = 2,\ \chi(C_{2n}) = 2,\ \chi(C_{2n+1}) = 3,\ \chi(T) = 2,$
где T — свободное дерево.

ТЕОРЕМА $\chi(G) \leqslant 1 + \Delta(G)$.

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 G: $p(G) = p$. Тогда, если $v \in V$, то $\chi(G - v) \leqslant \Delta(G - v) + 1 \leqslant \Delta(G) + 1$. Но $d(v) \leqslant \Delta(G)$, значит, хотя бы

один цвет в $\Delta(G)+1$ раскраске графа $G-v$ свободен для v. Покрасив v в этот цвет, имеем $\Delta(G)+1$ раскраску G. □

ТЕОРЕМА $p/\beta_0(G) \leqslant \chi(G) \leqslant p-\beta_0(G)+1$.

Доказательство

1. Пусть $\chi(G)=n$ и $V=V_1\cup\cdots\cup V_n$, где V_i — одноцветные классы. V_i — независимое множество вершин, значит, $|V_i|\leqslant\beta_0(G)$. Имеем:

$$p=\sum_{i=1}^{n}|V_i|\leqslant n\beta_0(G)\Longrightarrow p/\beta_0\leqslant\chi.$$

2. 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 $G-S$ можно получить $(n+1)$-раскраску G, так как все вершины из S можно покрасить в один новый цвет. Следовательно,

$$\chi(G)\leqslant f(G-S)+1\leqslant p-\beta_0+1.$$ □

ТЕОРЕМА *Пусть* $\chi:=\chi(G),\overline{\chi}:=\chi(\overline{G})$. *Тогда*

$$2\sqrt{p}\leqslant\chi+\overline{\chi}\leqslant p+1,$$
$$p\leqslant\chi\overline{\chi}\leqslant\left(\frac{p+1}{2}\right)^2.$$

Доказательство

1. Пусть $\chi(G)=n$, $V_1,\ldots,V_n$ — одноцветные классы, $p_i:=|V_i|$. Тогда

$$\sum_{i=1}^{n}p_i=p,$$

следовательно,

$$\max_{i=1}^{n}p_i\geqslant p/n.$$

Но V_i — независимые множества в G, следовательно, V_i — клики в $\overline{G}$. Значит,

$$\overline{\chi}\geqslant\max_{i=1}^{n}p_i\geqslant p/n.$$

Имеем $\chi\overline{\chi}\geqslant n\cdot p/n=p$.

2. Известно, что среднее геометрическое не превосходит среднего арифметического:

$$\frac{a+b}{2}\geqslant\sqrt{ab}.$$

Следовательно, $\chi+\overline{\chi}\geqslant2\sqrt{\chi\overline{\chi}}\geqslant2\sqrt{p}$.

3. Докажем индукцией по p, что $\chi + \overline{\chi} \leqslant p + 1$. 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,

$$\chi(G) \leqslant \chi(G - v) + 1 \,\&\, \chi(\overline{G}) \leqslant \chi(\overline{G} - v) + 1.$$

Если $\chi(G) < \chi(G - v) + 1 \vee \chi(\overline{G}) < \chi(\overline{G} - v) + 1$, 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

$$\chi(G) = \chi(G - v) + 1 \,\&\, \chi(\overline{G}) = \chi(\overline{G} - v) + 1.$$

Положим $d := d(v)$ в графе G, тогда $\overline{d} = p - d - 1$ — степень 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 раскрасить 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,

$$\chi + \overline{\chi} = \chi(G) + \chi(\overline{G}) =$$
$$= \chi(G - v) + 1 + \chi(\overline{G} - v) + 1 \geqslant d + 1 + p - d - 1 + 1 = p + 1.$$

4. Имеем $2\sqrt{\chi\overline{\chi}} \leqslant \chi + \overline{\chi} \leqslant p + 1$. Следовательно, $\left(\frac{p+1}{2}\right)^2 \geqslant \chi\overline{\chi}$. □

12.2. 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.

12.2.1. Укладка графов

Граф *укладывается* на некоторой поверхности, если его можно нарисовать на этой поверхности так, чтобы ребра 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, и не содержащая внутри себя вершин и ребер, называется *гранью*. Число граней плоского графа G обозначается $r(G)$.

ЗАМЕЧАНИЕ

Внешняя часть плоскости также образует грань.

Пример

На рис. 12.1 показаны диаграммы планарного графа K_4 и его укладки на плоскости. Этот граф имеет 4 грани.

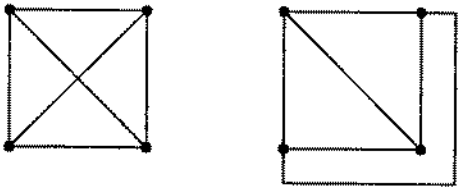


Рис. 12.1. Планарный граф и его укладка

12.2.2. Эйлерова характеристика

Для графов, уложенных на некоторой поверхности, справедливо определенное соотношение между числом вершин, ребер и граней. Это соотношение называется *эйлеровой характеристикой* поверхности.

ТЕОРЕМА (формула Эйлера) *В связном планарном графе справедливо следующее:*

$$p - q + r = 2.$$

Доказательство

Индукция по q. База: $q = 0 \Longrightarrow p = 1 \& r = 1$. Пусть теорема верна для всех графов с q ребрами: $p - q + r = 2$. Добавим еще одно ребро. Если добавляемое ребро соединяет существующие вершины, то $q' = q + 1$, $p' = p$, $r' = r + 1$ и $p' - q' + r' = p - q - 1 + r + 1 = p - q + r = 2$. Если добавляемое ребро соединяет существующую вершину с новой, то $p' = p + 1$, $q' = q + 1$, $r' = r$ и $p' - q' + r' = p + 1 - q - 1 + r = p - q + r = 2$ □

СЛЕДСТВИЕ *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 $3r \leqslant 2q$. Имеем

$$2 = p - q + r \leqslant p - q + \frac{2q}{3} \Longrightarrow 3p - 3q + 2q \geqslant 6 \Longrightarrow q \leqslant 3p - 6. \quad \square$$

СЛЕДСТВИЕ *K_5 и $K_{3,3}$ непланарны.*

Доказательство

1. Рассмотрим K_5. 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.
2. Рассмотрим $K_{3,3}$. 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, $4r \leqslant 2q$ или $2r \leqslant q$. По формуле Эйлера $6 - 9 + r = 2$, откуда $r = 5$. Имеем $2r = 2 \cdot 5 = 10 \leqslant q = 9$ — противоречие. □

ЗАМЕЧАНИЕ

Граф планарен тогда и только тогда, когда он не содержит в качестве подграфов ни K_5, ни $K_{3,3}$. Доказательство достаточности этого утверждения выходит за рамки данного курса.

СЛЕДСТВИЕ *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

Эйлер вывел свою формулу, исследуя многогранники. Действительно, развертка многогранника — это плоский граф, и обратно, связному плоскому графу соответствует многогранник. Таким образом, теория графов имеет связи с самыми разными, на первый взгляд далекими, областями знания.

12.2.3. Теорема о пяти красках

ТЕОРЕМА *Всякий планарный граф можно раскрасить пятью красками.*

Доказательство

Достаточно рассматривать связные графы, потому что

$$\chi(\bigcup_{i=1}^{n} G_i) = \max_{i=1}^{n} \chi(G_i).$$

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 $p+1$ вершиной. 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.

1. Если $d(v) < 5$, то в 5-раскраске $G - v$ существует цвет, свободный для v.
2. Если $d(v) = 5$ и для $\Gamma^+(v)$ использованы не все пять цветов, то в 5-раскраске $G - v$ существует цвет, свободный для v.
3. Остался случай, когда $d(v) = 5$ и все пять цветов использованы (вершина v_i покрашена в цвет i, рис. 12.2). Пусть G_{13} — правильный подграф $G - v$, 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 v_1, произведем перекраску

$1 \leftrightarrow 3$. При этом получится 5-раскраска $G - v$, но цвет 1 будет свободен для v. В противном случае существует простая цепь, соединяющая v_1 и v_3 и состоящая из вершин, покрашенных в цвета 1 и 3. В этом случае 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 v. □

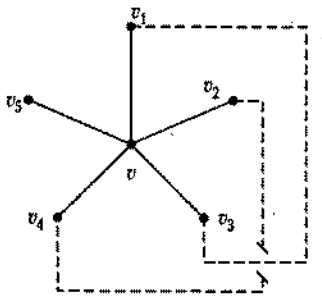


Рис. 12.2. Иллюстрация к доказательству теоремы о пяти красках

12.3. Алгоритмы раскрашивания

В этом разделе на примере алгоритмов раскрашивания графов вводится понятие приближенного алгоритма в более широком смысле по сравнению с тем, который обычно подразумевается в вычислительной математике и при проведении численных расчетов на ЭВМ.

12.3.1. Точный алгоритм раскрашивания

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 $G - S$.

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 11.2.4.

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 ее в 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 возрастет (но может уменьшиться, если $\chi(G) < k$). На каждом шаге алгоритма рассматривается одно из множеств, следовательно, процесс закончится. □

12.3.2. Приближенный алгоритм последовательного раскрашивания

В предыдущем подразделе алгоритм точного раскрашивания был построен на основе алгоритма выделения максимальных независимых множеств вершин, который имеет переборный характер. Таким образом, алгоритм точного раскрашивания также имеет переборный характер. В таких случаях целесообразно рассматривать *приближенные* 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 12.1. 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 не раскрашены }
end for
for v ∈ V do
  A := {1, ..., p} { все цвета }
  for u ∈ Γ+(v) do
    A := A \ {C[u]} { занятые для 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. □

12.3.3. Улучшенный алгоритм последовательного раскрашивания

Следующий алгоритм также строит допустимую раскраску, применяя такую эвристику: начинать раскрашивать следует с вершины наибольшей степени, поскольку, если их раскрашивать в конце процесса, то более вероятно, что для них не найдется свободного цвета и придется задействовать еще один цвет.

Алгоритм 12.2. 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 ее 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 все что можно красим в цвет 1, затем в оставшемся красим все 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. □

Комментарии

Центральный результат этой главы — теорема о пяти красках — изложен по книге [23]. Алгоритмы раскрашивания изложены по книге [11], в которой можно найти их более детальное и подробное обсуждение. Различные применения задачи о раскраске графов в программировании и смежные вопросы освещены в [5].

Упражнения

12.1. Доказать, что

$$\chi(G(V,E)) \leqslant 1 + \max_{V' \subset V} \delta(G'(V',E')).$$

12.2. Доказать, что если в планарном графе каждая грань есть C_n, то

$$q = \frac{n(p-2)}{n-2}.$$

12.3. Построить примеры графов, для которых алгоритмы последовательного раскрашивания строят не минимальную раскраску.

Литература

1. А. Ахо, Дж. Хопкрофт, Дж. Ульман, Построение и анализ вычислительных алгоритмов, Мир, 1979.
2. К. Берж, Теория графов и ее применения, Изд. иностр. лит., 1962.
3. Д. А. Владимиров, Булевы алгебры, Наука, 1969.
4. М. Гэри, Д. Джонсон, Вычислительные машины и труднорешаемые задачи, Мир, 1982.
5. В. А. Евстигнеев, Применение теории графов в программировании, Наука, 1985.
6. В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич, Лекции по теории графов, Наука, 1990.
7. А. П. Ершов, Введение в теоретическое программирование, Наука, 1977.
8. Д. Кнут, Искусство программирования для ЭВМ, т. 1, Мир, 1977.
9. Д. Кнут, Искусство программирования для ЭВМ, т. 2, Мир, 1977.
10. П. Кон, Универсальная алгебра, Мир, 1968.
11. Н. Кристофидес, Теория графов, алгоритмический подход, Мир, 1978.
12. В. Кук, Г. Бейз, Компьютерная математика, Наука, 1990.
13. С. С. Лавров, Л. И. Гончарова, Автоматическая обработка данных, хранение информации в памяти ЭВМ, Наука, 1971.
14. В. Липский, Комбинаторика для программистов, Мир, 1988.
15. Э. Мендельсон, Введение в математическую логику, Наука, 1984.
16. В. И. Нечаев, Элементы криптографии. Основы теории защиты информации, Высшая школа, 1999.
17. Дискретная математика и математические вопросы кибернетики. Под ред. С. В. Яблонского и О. Б. Лупанова, Наука, 1974.
18. И. В. Романовский, Дискретный анализ, Невский диалект, 1999.
19. В. Н. Сачков, Введение в комбинаторные методы дискретной математики, Наука, 1977.
20. В. Н. Сачков, Введение в комбинаторные методы дискретной математики, Наука, 1982.

21. Р. Уилсон, Введение в теорию графов, Мир, 1977.

22. Э. Фрид, Элементарное введение в абстрактную алгебру, Мир, 1979.

23. Ф. Харари, Теория графов, Мир, 1973.

24. Ч. Чень, Р. Ли, Математическая логика и автоматическое доказательство теорем, Наука, 1983.

25. С. В. Яблонский, Введение в дискретную математику, Наука, 1986.

Алфавитный указатель