Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Инженерно-экономический факультет

Кафедра экономической информатики

Ю. В. Поттосин, Т. Г. Пинчук, С. А. Поттосина

ОСНОВЫ ДИСКРЕТНОЙ МАТЕМАТИКИ И ТЕОРИИ АЛГОРИТМОВ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)»

Минск БГУИР 2021

УДК 519.854(075.8)
ББК 22.176я73
П64

Р е ц е н з е н т ы:

кафедра цифровых систем и технологий государственного учреждения образования «Институт бизнеса Белорусского государственного университета» (протокол №8 от 20.02.2020);

заведующий лабораторией идентификации систем государственного научного учреждения «Объединенный институт проблем информатики Национальной академии наук Беларуси» доктор технических наук, профессор А. А. Дудкин

Поттосин, Ю. В.

П64 Основы дискретной математики и теории алгоритмов : учеб.-метод. пособие / Ю. В. Поттосин, Т. Г. Пинчук, С. А. Поттосина. – Минск : БГУИР, 2021. – 122 с. : ил.

ISBN 978-985-543-600-4.

Содержит основные разделы дискретной математики: множества, отношения, основные понятия теории графов, комбинаторные задачи и методы комбинаторного поиска, булевы функции и их графическое представление, нормальные формы, минимизация ДНФ, элементы математической логики, основы теории алгоритмов, конечный автомат и минимизация полных автоматов. Рассматриваются прикладные задачи по некоторым разделам с описанием алгоритма их решения.

УДК 519.854(075.8)
ББК 22.176я73

ISBN 978-985-543-600-4

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

В предлагаемом учебно-методическом пособии описываются задачи дискретной математики, возникающие при моделировании ситуаций, связанных с разработкой и тестированием программных продуктов, исследованиями структуры коммуникационных связей в организации, разработкой систем выбора наилучшего решения из заданного множества, формализацией логических рассуждений.

Излагаются основные понятия теории множеств и отношений. Описывается аппарат булевых векторов и матриц для представления теоретико-множественных объектов и операций над ними. Довольно подробно рассматриваются основы теории графов, абстрактной булевой алгебры с различными интерпретациями, теории булевых функций. Особое внимание уделяется комбинаторным задачам оптимизации: раскраске графов, кратчайшему покрытию множеств и др.

Описаны элементы математической логики и предикаты, а также построение и решение логических уравнений. Дается понятие машины Тьюринга как модели алгоритма.

Описаны классические методы минимизации булевых функций и систем булевых функций в терминах булевых и троичных векторов и матриц. Предлагаются начальные сведения по теории конечных автоматов. Рассматривается задача минимизации числа состояний конечного автомата.

Материал подготовлен на основе курса лекций, который читался в течение ряда лет в Белорусском государственном университете информатики и радиоэлектроники. При этом использовались материалы, изложенные в монографиях, список которых приведен в конце данного учебно-методического пособия.

1. МНОЖЕСТВА

1.1. Основные понятия

Понятие множества является одним из основных понятий математики. Под *множеством* обычно понимается совокупность или набор каких-то объектов, имеющих что-то общее, и при этом каждый из них чем-то отличается от другого. Основным также является понятие *элемента множества*. Это исходные понятия, и поэтому точного определения для них нет. Принадлежность элемента a множеству A обозначается как $a \in A$. Если же некоторый элемент a не принадлежит множеству A, то следует писать $a \notin A$. Элементы множества обозначаются строчными латинскими буквами $a, b, c, \ldots, z$. Множества обозначают прописными латинскими буквами $A, B, C, \ldots, Z$.

Множество A является *подмножеством* множества B, если всякий элемент из A принадлежит множеству B. Этот факт обозначается $A \subseteq B$ ($\subseteq$ – знак включения). При этом говорят, что множество B *содержит*, или *покрывает*, множество A. Множества A и B равны ($A = B$), если $A \subseteq B$ и $B \subseteq A$. Множество, не имеющее ни одного элемента, называется *пустым* и обозначается $\varnothing$. Оно является подмножеством любого множества, т. е. $\varnothing \subseteq A$ для любого A. Пустое множество, а также само A являются *несобственными подмножествами* множества A. Если $A \subseteq B$ и $A \neq B$ для некоторого непустого множества A, то A является *собственным подмножеством* множества B, и это обозначается как $A \subset B$ ($\subset$ – знак строгого включения). Нетрудно доказать, что число подмножеств любого конечного множества, содержащего n элементов, равно 2^n. Действительно, число подмножеств пустого множества равно единице, а добавление к любому конечному множеству еще одного элемента увеличивает это число в два раза.

Множество, число элементов которого конечно, называется *конечным*, в противном случае – *бесконечным*. Количество элементов в конечном множестве A называется мощностью множества и обозначается $|A|$. Мощность бесконечного множества выражается через соответствие. Если конечные множества A и B *равномощны*, т. е. $|A| = |B|$, то между ними можно установить *взаимно однозначное соответствие*. Каждому элементу из A ставится в соответствие элемент из B, и наоборот. Для бесконечных множеств отношение равномощности устанавливается путем нахождения взаимно однозначного соответствия между их элементами. Бесконечное множество натуральных чисел $\mathbb{N} = \{1, 2, \ldots\}$ является *счетным* множеством. Множества, равномощные с $\mathbb{N}$, называются *счетными*, в противном случае – *несчетными*. Для того чтобы выяснить, является ли некоторое множество A счетным, надо найти способ установить взаимно однозначное соответствие между A и множеством натуральных чисел, т. е. способ пронумеровать элементы множества A. Примером несчетного множества является множество всех действительных чисел отрезка $[0, 1]$. Конечные и счетные множества называются дискретными множествами.

Множество всех элементов, которые могут встретиться в данном исследовании, называется универсальным множеством и обозначается U.

Задать множество можно различными способами.

Перечисление элементов. Это простейший способ задания конечного множества. Например, если множество A состоит из элементов $a_1, a_2, \ldots, a_n$, то можно записать $A = \{a_1, a_2, \ldots, a_n\}$.

Указание свойств элементов. При таком способе задается одно или несколько свойств, по которым определяется принадлежность элементов к данному множеству. Если $P(x)$ означает, что x обладает свойством P, то $A = \{x \,/\, P(x)\}$ есть множество всех тех и только тех элементов, которые обладают свойством P. Например, $A = \{x \mid x = 2^k, k \in \mathbb{N}\}$ – множество всех чисел, каждое из которых представляет собой число 2 в натуральной степени.

Индуктивный способ. Задается некоторая порождающая процедура, которая определяет способ получения элементов множества из уже полученных элементов. Например, для бесконечного множества $A = \{1, 2, 4, 8, 16, \ldots\}$ такой определяющей процедурой является следующая: 1) $1 \in A$; 2) если $m \in A$, то $2m \in A$.

Алгебраический способ. При этом способе дается формула, по которой можно получить множество из других множеств с помощью алгебраических операций над ними.

Визуальное представление множеств. Множества изображаются на плоскости в виде фигур, называемых *диаграммами Эйлера – Венна*. Этот способ используется обычно для наглядной демонстрации операций над множествами или отношений между множествами. Пример использования данного способа будет приведен при описании операций над множествами.

Булевы векторы. Как было уже сказано, всякое множество, подлежащее рассмотрению, считается подмножеством множества U. Тогда любое множество M можно представить вектором с $|U|$ компонентами, которые соответствуют элементам множества U. Компонента этого вектора равна 1, если соответствующий элемент принадлежит множеству M, и 0 – в противном случае. Пусть $U = \{a, b, c, d, e\}$ и $M = \{a, b, d\}$. Тогда M представится вектором 11010. Векторы 00000 и 11111 задают соответственно пустое множество $\varnothing$ и универсальное множество U.

1.2. Операции над множествами

Как было сказано в подразд. 1.1, множество можно представить в виде результата операций над другими множествами.

Объединение множеств A и B представляет собой множество, содержащее те и только те элементы, которые принадлежат A или B (хотя бы одному из этих множеств):

$$A \cup B = \{x \mid x \in A \text{ или } x \in B\}.$$

Пересечением множеств A и B является множество, содержащее те и только те элементы, каждый из которых принадлежит как A, так и B:

$$A \cap B = \{x \mid x \in A \text{ и } x \in B\}.$$

Разность множеств A и B состоит из элементов множества A, которые не принадлежат множеству B:

$$A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}.$$

Симметрическая разность множеств A и B содержит все элементы из A, не принадлежащие B, и все элементы из B, не принадлежащие A:

$$A \Delta B = \{x \mid (x \in A \text{ и } x \notin B) \text{ или } (x \in B \text{ и } x \notin A)\}.$$

Дополнение множества A состоит из элементов универсального множества U, не принадлежащих A:

$$\overline{A} = \{x \mid x \in U \text{ и } x \notin A\}.$$

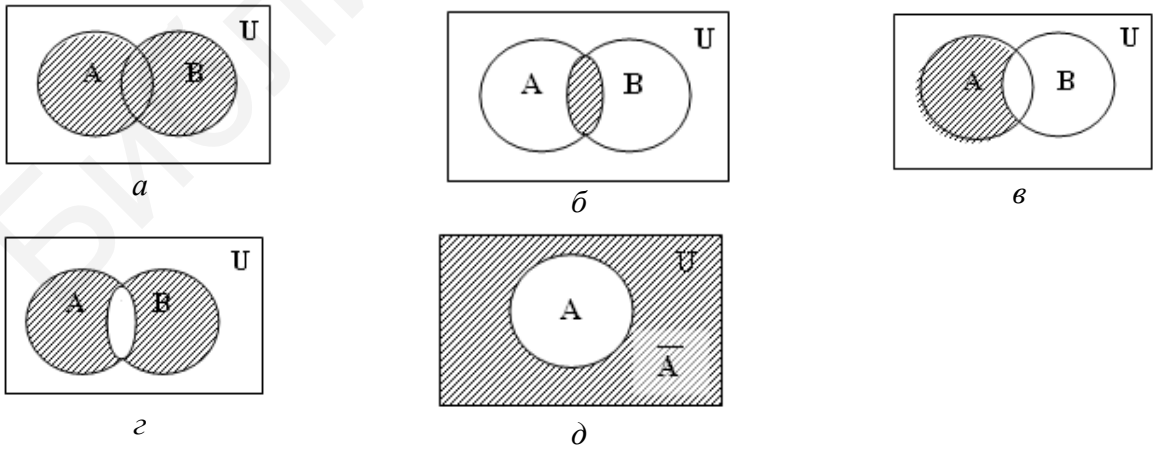
Операцию дополнения обозначают еще символом $\neg$.

На основе перечисленных операций строятся теоретико-множественные формулы. Понятие формулы определим индуктивно следующим образом:

а) любой символ, обозначающий множество, есть формула;

б) если A и B – формулы, то $A \cup B$, $A \cap B$, $A \setminus B$, $A \Delta$ B, $\neg A$ – также формулы.

Операции объединения, пересечения и дополнения называются булевыми. Бинарные операции объединения, пересечения, разности и унарная операция дополнения проиллюстрированы на диаграммах Венна (рис. 1.1). Результирующее множество элементов, соответствующее каждой из этих операций, изображено заштрихованной областью.



а *б* *в* *г* *д*

Рис. 1.1. Операции над множествами:
а – объединение; *б* – пересечение; *в* – разность;
г – симметрическая разность; *д* – дополнение

1.3. Булева алгебра множеств

Абстрактная алгебраическая система, состоящая из множества подмножеств некоторого универсального множества с введенными выше операциями дополнения, пересечения и объединения, составляют *булеву алгебру множеств*. Перечислим основные законы этой алгебры, используя общепринятое правило, что если в формуле отсутствуют скобки, устанавливающие порядок выполнения операций, то сначала выполняется дополнение, потом пересечение и затем объединение. Для повышения наглядности формулы знак пересечения множеств, подобно знаку арифметического умножения, будем опускать.

Коммутативность:

$A \cup B = B \cup A$; $\quad A\,B = B\,A$.

Ассоциативность:

$A \cup (B \cup C) = (A \cup B) \cup C$; $\quad A\,(B\,A) = (A\,B)\,C$.

Дистрибутивность:

$A\,(B \cup C) = A\,B \cup A\,C$; $\quad A \cup B\,C = (A \cup B)\,(A \cup C)$.

Идемпотентность:

$A \cup A = A$; $\quad A\,A = A$.

Законы де Моргана:

$\overline{A \cup B} = \bar{A}\,\bar{B}$; $\quad \overline{A\,B} = \bar{A} \cup \bar{B}$.

Законы операций с константами (пустым и универсальным множествами):

$A \cup \varnothing = A$; $\quad A\,U = A$;

$A\,\varnothing = \varnothing$; $\quad A \cup U = U$;

$A \cup \bar{A} = U$; $\quad A\,\bar{A} = \varnothing$.

Закон двойного дополнения:

$\bar{\bar{A}} = A$.

Заметим, что для каждой пары формул, представляющих тот или иной закон, справедливо следующее: одна из формул получается из другой взаимной заменой всех операций пересечения на операции объединения и всех символов $\varnothing$ на символы U. Этот факт известен под названием *принципа двойственности*. Заметим также, что для операции пересечения пустое множество имеет свойство нуля, универсальное множество – свойство единицы. Для операции объединения универсальное множество имеет свойство нуля, а пустое множество – свойство единицы.

Формула, в которой присутствуют символы операций над множествами, есть способ задания множества. Две формулы *равносильны*, если они представляют одно и то же множество. Любую формулу булевой алгебры множеств можно вывести путем равносильных преобразований, используя формулы из приведенного списка. Данный список является достаточным, но для вывода любой формулы данной алгебры можно воспользоваться меньшим списком, т. е.

некоторые формулы этого списка можно вывести из других. Например, формулу $A \cup B\ C = (A \cup B)\ (A \cup C)$ (дистрибутивность объединения относительно пересечения) можно получить следующим образом. Ее правую часть, используя дистрибутивность пересечения, представим как $(A \cup B)\ (A \cup C) = (A \cup B)\ A \cup (A \cup B)\ C$. Раскрыв скобки (по закону ассоциативности), получим

$$(A \cup B)\ A \cup (A \cup B)\ C = A\ A \cup B\ A \cup A\ C \cup B\ C.$$

Применим закон идемпотентности и введем константу U ($A\ A = A = A\ U$), в результате чего после применения закона коммутативности пересечения правая часть примет вид $A\ U \cup A\ B \cup A\ C \cup B\ C$. После вынесения за скобки A получим $A\ (U \cup B \cup C) \cup B\ C$, что равно левой части исходного выражения согласно свойству константы U.

Подобным образом выведем закон поглощения $A \cup A\ B = A$, которого нет в приведенном списке:

$$A \cup A\ B = A\ U \cup A\ B = A\ (U \cup B) = A.$$

Используя принцип двойственности, получим: $A\ (A \cup B) = A$.

Формулу $A\ B \cup \bar{A}\ C = A\ B \cup \bar{A}\ C \cup B\ C$ выведем следующим образом:

$$A\ B \cup \bar{A}\ C \cup B\ C = A\ B \cup \bar{A}\ C \cup B\ C(A \cup \bar{A}) = A\ B(U \cup C) \cup \bar{A}\ C(U \cup B) = A\ B \cup \bar{A}\ C.$$

Используя только что выведенную формулу и закон поглощения, докажем, что $A \cup \bar{A}\ B = A \cup B$:

$$A \cup \bar{A}\ B = A\ U \cup \bar{A}\ B = A\ U \cup \bar{A}\ B \cup U\ B = A \cup \bar{A}\ B \cup B = A \cup B.$$

1.4. Векторы

Для описания свойств элементов множества удобны векторные представления**.** Пусть нас интересуют свойства (значения, состояния, признаки, атрибуты) элемента v множества V по n фиксированным характеристикам $A_1, A_2, \ldots, A_n$. При этом каждая характеристика A_i ($i = 1, 2, \ldots, n$) представлена множеством из m_i допустимых значений $a_i \in A_i$. В таком случае каждый элемент v множества V может быть задан упорядоченным набором значений $a_1, a_2, \ldots, a_n$ по интересуемым характеристикам $A_1, A_2, \ldots, A_n$, так что $a_i \in A_i$, $i = 1, 2, \ldots, n$. В результате получаем $v = (a_1, a_2, \ldots, a_n)$.

Вектор v – упорядоченный набор элементов $v = (a_1, a_2, \ldots, a_n)$, где $a_1, a_2, \ldots, a_n$ – компоненты (координаты) вектора. Число n компонент называется длиной (размерностью) вектора.

Два вектора $v_1 = (a_1, a_2, \ldots, a_n)$ и $v_2 = (b_1, b_2, \ldots, b_n)$ равны, если они имеют одинаковую длину и соответствующие координаты их равны.

Операции над множеством векторов – объединение, пересечение, разность, дополнение – аналогичны соответствующим операциям над множествами элементов.

Рассмотрим еще некоторые операции над векторами.

Проекцией вектора $v = (a_1, a_2, \ldots, a_n)$ на A_i является его i-я компонента: $\text{пр}_i v = a_i$.

Проекцией вектора $v = (a_1, a_2, \ldots, a_n)$ на оси с номерами $i_1, i_2, ..., i_k$ называется вектор длиной k: $\text{пр}_{i_1, i_2, ..., i_k} v = (a_{i_1}, a_{i_2}, ..., a_{i_k})$.

Проекцией множества векторов V на A_i называется множество проекций всех векторов из V на A_i: $\text{пр}_i V = \{\text{пр}_i v / v \in V\}$.

Проекцией множества векторов V на множества $A_{i_1}, A_{i_2}, ..., A_{i_k}$ называется множество проекций всех векторов $v \in V$ на $A_{i_1}, A_{i_2}, ..., A_{i_k}$: $\text{пр}_{i_1, i_2, ..., i_k} V = \{\text{пр}_{i_1, i_2, ..., i_k} v / v \in V\}$.

Проекцией упорядоченного множества векторов на A_i называется упорядоченное множество проекций векторов на A_i. Проекцией упорядоченного множества векторов V на $A_{i_1}, A_{i_2}, ..., A_{i_k}$ называется упорядоченное множество проекций всех векторов $v \in V$ на $A_{i_1}, A_{i_2}, ..., A_{i_k}$. Кроме того, над векторами одинаковой длины возможно выполнение различных операций сравнения, задаваемых теми или иными правилами сравнения векторов. Например, *правило сравнения векторов по предпочтению.*

Пусть V – множество векторов длиной n, компонентами которых являются числа. Вектор $v_1 = (a_1, a_2, \ldots, a_n)$ предпочтительнее вектора $v_2 = (b_1, b_2, \ldots, b_n)$ (обозначение $v_1 \succ v_2$), если хотя бы одна компонента вектора v_1 больше одноименной компоненты вектора v_2, а остальные – больше или равны одноименным компонентам.

2. ОТНОШЕНИЯ БИНАРНЫЕ И N-АРНЫЕ

2.1. Декартово произведение

Декартовым, или *прямым*, *произведением* двух множеств A и B (обозначается $A \times B$) называется множество всех таких упорядоченных пар (a, b), что $a \in A$ и $b \in B$. Пусть, например, $A = \{a, b, c\}$ и $B = \{l, m\}$. Тогда $A \times B = \{(a, l), (b, l), (c, l), (a, m), (b, m), (c, m)\}$. Это понятие распространяется на случай с более чем одним сомножителем. Декартово произведение множеств $A_1, A_2, \ldots, A_n$

(обозначается $A_1 \times A_2 \times \ldots \times A_n$) есть множество всех векторов $(a_1, a_2, \ldots, a_n)$ размерностью n, таких, что $a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n$.

Декартово произведение n одинаковых сомножителей $A \times A \times \ldots \times A$ обозначается символом A^n и называется n-й *степенью* множества A. При этом $A^1 = A$. Примером декартова произведения является $R \times R = R^2$ – множество точек на плоскости. Здесь элементы $x \in R$ и $y \in R$ служат координатами некоторой точки на плоскости. Другим примером является множество R^3 точек в трехмерном евклидовом пространстве. Обобщением этих понятий является n-мерное пространство.

Любое подмножество $R \subseteq A_1 \times A_2 \times \ldots \times A_n$ декартова произведения n множеств называется *n-арным отношением*. При $n = 1, 2, 3$ имеем *унарное*, *бинарное*, *тернарное* отношения соответственно. Унарное отношение на множестве A представляет собой подмножество множества A.

2.2. Бинарные отношения (соответствия)

Бинарным отношением, или *соответствием* между элементами множеств A и B, называется любое подмножество $R \subseteq A \times B$ декартова произведения этих множеств. Тот факт, что некоторые $a \in A$ и $b \in B$ находятся в отношении R, иногда выражают как $a\,R\,b$. В качестве примера бинарного отношения рассмотрим отношение R между элементами множеств $A = \{1, 2, 3\}$ и $B = \{1, 2, 3, 4, 5, 6\}$, которое можно выразить словами так: элемент $x \in A$ есть делитель элемента $y \in B$. Тогда имеем $R = \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 2), (2, 4), (2, 6), (3, 3), (3, 6)\}$.

Бинарное отношение удобно представлять в виде двоичной (*булевой*) матрицы. При этом элементы множеств A и B должны быть пронумерованы, и если i-й элемент множества A соответствует j-му элементу множества B, то элемент матрицы, расположенный на пересечении i-й строки и j-го столбца, имеет значение 1, в противном случае он имеет значение 0. Например, рассмотренное отношение R будет представлено следующей матрицей:

$$\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}.$$

Проекция элемента (a, b) множества $A \times B$ на множество A есть элемент a. Аналогично элемент b является проекцией элемента (a, b) множества $A \times B$ на множество B.

Проекцией множества $E \subset A \times B$ на A называется множество всех тех элементов из A, которые являются проекциями элементов из E на множество A.

Для множеств A и B проекцией элемента (2, 4) на множество A является элемент 2, а проекцией множества {(1, 2), (2, 2), (2, 4)} – множество {1, 2}.

Сечением множества $R \subset A \times B$ по a, обозначаемым $R(a)$, называется множество всех тех элементов $y \in B$, для которых $(a, y) \in R$.

Сечением $R(X)$ множества R по $X \subset A$ является объединение сечений для всех элементов из X. Пусть $R = \{(1, 1), (1, 3)\ (1, 5), (1, 6), (2, 2), (2, 4), (3, 3), (3, 6)\}$. Тогда $R(2) = \{2, 4\}$, а если $X = \{2, 3\}$, то $R(X) = \{2, 3, 4, 6\}$.

Бинарное отношение можно задавать с помощью сечений. Например, отношение, представленное матрицей

$$\begin{array}{c} \begin{matrix} b_1 & b_2 & b_3 & b_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{matrix},$$

можно задать следующим образом: $R(a_1) = \{b_1, b_3\}$, $R(a_2) = \{b_1, b_3, b_4\}$, $R(a_3) = \{b_1, b_4\}$, $R(a_4) = \varnothing$, $R(a_5) = \{b_4\}$. Множество сечений для всех $a \in A$ называется *фактор-множеством*.

Областью определения отношения $R \subseteq A \times B$ является проекция множества R на A, областью определения рассматриваемого отношения – $\{a_1, a_2, a_3, a_5\}$. *Областью значений* отношения $R \subseteq A \times B$ является сечение множества R по A, областью значений рассматриваемого отношения – $\{b_1, b_3, b_4\}$.

Образом множества $X \subseteq A$ относительно R называется множество $\{b \mid b \in B,\ x \in X,\ (x, b) \in R\}$. *Прообразом* множества $Y \subseteq B$ относительно R называется множество $\{a\ \ a \in A, y \in Y, (a, y) \in R\}$. В нашем последнем примере образом множества $\{a_1, a_3\}$ относительно R является $\{b_1, b_3, b_4\}$, а прообразом множества $\{b_3, b_4\}$ будет $\{a_1, a_2, a_3, a_5\}$.

Обратным отношением R^{-1} для некоторого отношения $R \subseteq A \times B$ является множество, образованное теми парами $(b, a) \in B \times A$, для которых $(a, b) \in R$. Матрица, представляющая отношение R^{-1}, получается транспонированием матрицы, представляющей R, т. е. заменой строк столбцами и наоборот.

Например, рассмотренному отношению R будет соответствовать обратное отношение R^{-1}, представляемое матрицей

$$\begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{matrix}.$$

2.3. Операции над бинарными отношениями

Поскольку всякое отношение есть некоторое множество пар, над отношениями применимы все стандартные операции над множествами, т. е. объединение, пересечение, дополнение. Универсальным множеством для операции дополнения при этом является $A \times B$.

Рассмотрим операцию *композиции* отношений. Заданы множества A, B, C и отношения $R \subseteq A \times B$ и $S \subseteq B \times C$. Композиция отношений S и R, обозначаемая SR, (не путайте с пересечением множеств S и R!) – это такое отношение между элементами множеств A и C, что для всех $a \in A$ сечение множества SR по a совпадает с сечением множества S по подмножеству $R(a) \subseteq B$. Это записывается в виде $(SR)(a) \subseteq S(R(a))$. Например, пусть отношения R и S заданы следующими матрицами соответственно:

$$R = \begin{array}{c} \begin{matrix} b_1 & b_2 & b_3 & b_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{matrix}, \quad S = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{matrix}.$$

Тогда композиция SR этих отношений представится матрицей

$$SR = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{matrix}.$$

2.4. Функциональные отношения

Отношение $R \subseteq A \times B$ называется *функциональным*, если для каждого $a \in A$ сечение множества R по a содержит не более одного элемента, т. е. для каждого a справедливо $|\{a \mid (a, b) \in R, b \in B\}| \le 1$. В функциональном отношении не существует пар с одинаковым левым элементом и различными правыми элементами, т. е. если $(a, b) \in R$ и R – функциональное отношение, то в R не может быть пары вида (a, c), где $b \neq c$. Матрица, представляющая функциональное отношение, в каждой строке имеет не более одной единицы. Например,

$$\begin{array}{c} \begin{matrix} b & d & e \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix}.$$

Если сечение функционального отношения R по любому элементу a из множества A содержит один и только один элемент, то отношение R называется *всюду определенным*.

Если отношение R^{-1}, обратное для функционального отношения R, также является функциональным, то отношение R называется *взаимно однозначным*.

Для всякого функционального отношения $R \subseteq A \times B$ можно определить *функцию*, связанную с этим отношением. Для обозначения функции используется запись $f : A \to B$. Если $(x, y) \in R$, то это можно выразить как $y = f(x)$, где x является *аргументом*, а y *– значением функции* f.

Множество $\{x \mid (x, y) \in R\}$ называется *областью определения* функции f. Если это множество совпадает с A, то функция f является *всюду определенной*. Такая функция называется *отображением* множества A в B. В противном случае функцию называют *частичной*.

Множество $\{y \mid (x, y) \in R\}$ называется *областью значений* функции f. Если область значений функции f совпадает с множеством B, то f называют отображением A на B, *сюръективным* отображением, или *сюръекцией*. Обязательным условием существования отображения A на B является $|A| \geq |B|$.

Если функциональное отношение $R \subseteq A \times B$, определяющее функцию f, является взаимно однозначным, то функцию f называют *инъективным* отображением, или *инъекцией*. В этом случае существует функция f^{-1}, которая является *обратной* к функции f. При этом если $y = f(x)$, то $x = f^{-1}(y)$, а мощность области определения функции f не должна превышать $|B|$.

Функция f называется *биективным* отображением, или *биекцией*, если она является как сюръективным, так и инъективным отображением. Такое отображение называется еще *1-1 соответствием*.

На рис. 2.1 изображены схемы рассмотренных видов отображений.

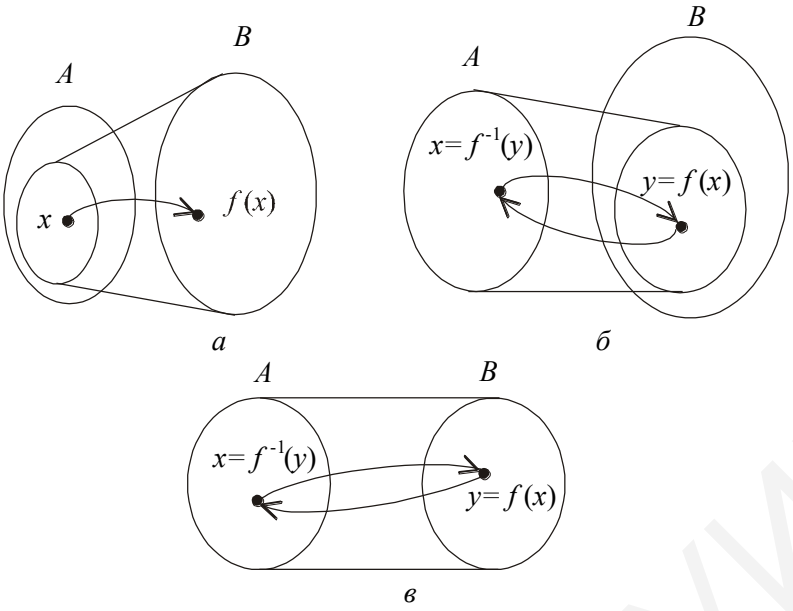


Рис. 2.1. Схемы функциональных отображений:
а – сюръекция; *б* – инъекция; *в* – биекция

Если R – взаимно однозначное отношение между элементами одного и того же множества, т. е. $R \subseteq A \times A = A^2$, и, кроме того, R и R^{-1} всюду определены, то отображение, связанное с R, называется *подстановкой*.

Функция, определенная на множестве натуральных чисел, называется *последовательностью*, а каждое ее значение – *членом* последовательности.

Отображение f произвольного множества в множество действительных чисел называется *функционалом*. Примером функционала может служить определенный интеграл.

Отображение $f : A \to B$, где A и B – некоторые множества функций, называется *оператором*. Оператор преобразует одну функцию в другую. Примером оператора является оператор суперпозиции функций, где аргументами некоторых функций служат другие функции. Это понятие будет использовано далее.

2.5. Бинарные отношения на множестве

Пусть $R \subseteq A \times A$. Определим некоторые свойства, которыми может обладать или не обладать такое отношение:

– *рефлексивность*: если $a = b$, то $a\ R\ b$;

– *иррефлексивность*: если $a\ R\ b$, то $a \neq b$;

– *симметричность*: если $a\ R\ b$, то $b\ R\ a$;

– *антисимметричность*: если $a\ R\ b$ и $b\ R\ a$, то $a = b$;

– *транзитивность*: если $a\ R\ b$ и $b\ R\ c$, то $a\ R\ c$;

– *дихотомия*: если $a \neq b$, то либо $a\ R\ b$, либо $b\ R\ a$.

Следует выделить некоторые типы бинарных отношений, характеризуемые определенным набором свойств.

Отношение *эквивалентности* рефлексивно, симметрично и транзитивно. Примерами отношения эквивалентности являются равносильность формул, подобие геометрических фигур, принадлежность студентов к одной группе, принадлежность населенных пунктов к одному району и т. п.

Отношение эквивалентности делит множество на непересекающиеся подмножества – *классы эквивалентности*. С другой стороны, всякое разбиение множества M на непересекающиеся подмножества задает отношение эквивалентности на множестве M: любые два элемента, принадлежащие одному и тому же классу разбиения, эквивалентны, а элементы, принадлежащие различным классам, не являются эквивалентными. Множество всех классов эквивалентности образует *фактор-множество* множества M по R (обозначается M/R).

Отношение *совместимости* рефлексивно и симметрично. Примерами отношения совместимости являются близость чисел, знакомство людей и т. п.

Отношение *нестрогого порядка* рефлексивно, антисимметрично и транзитивно. Отношения $\leq$ (меньше или равно) и $\geq$ (больше или равно) для действительных чисел так же, как $\subseteq$ и $\supseteq$ для множеств, являются отношениями нестрогого порядка.

Отношение *строгого порядка* иррефлексивно, антисимметрично и транзитивно. Отношениями строгого порядка являются $<$ (меньше) и $>$ (больше) для действительных чисел, а также $\subset$ и $\supset$ для множеств.

Множество M, на котором задано отношение порядка R (строгого или нестрогого), может быть *полностью* упорядоченным, если любые два элемента a и b из M находятся в отношении R, т. е. $a\ R\ b$ или $b\ R\ a$. При этом говорят, что a и b *сравнимы*. Если M содержит хотя бы одну пару элементов c и d, для которых не имеет место ни $c\ R\ d$, ни $d\ R\ c$, то множество M является *частично* упорядоченным, а указанные элементы c и d *несравнимы*. Отношение *полного порядка* обладает свойствами иррефлексивности, антисимметричности и дихотомии. Полный порядок называют еще *линейным* или *совершенным*.

Для множества действительных чисел отношения $\leq$ и $<$ являются отношениями полного порядка. Для семейства подмножеств некоторого множества M отношение $\subseteq$ является отношением частичного порядка, например, $\{a_1, a_3\} \subseteq \{a_1, a_2, a_3\}$, а подмножества $\{a_1, a_3\}$ и $\{a_1, a_2, a_4\}$ несравнимы.

Порядок букв в алфавите и естественный порядок цифр являются полными порядками. На основе порядка букв строится *лексикографический* порядок слов, используемый в словарях и определяемый следующим образом.

Обозначим это отношение порядка символом $\prec$. Пусть имеются слова $w_1 = a_{11}a_{12} \ldots a_{1m}$ и $w_2 = a_{21}a_{22} \ldots a_{2n}$. Тогда $w_1 \prec w_2$, если и только если либо $w_1 = pa_iq$, $w_2 = pa_jr$ и $a_i \prec a_j$, где p, q и r – некоторые слова, возможно, пустые, а a_i и a_j – буквы, либо $w_2 = w_1p$, где p – непустое слово.

Например, учебник $\prec$ ученик и мор $\prec$ море. В первом случае p = уче, a_i = б, a_j = н, q = ник, r = ик, и в алфавите буква «н» стоит дальше буквы «б». Потому в словаре слово «ученик» следует искать после слова «учебник». Во втором случае w_1 = мор и p = е. Согласно лексикографическому порядку слово «море» должно быть помещено в словаре после слова «мор».

3. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ

3.1. Абстрактный граф

Граф можно определить как совокупность двух множеств: $G = (V, E)$, где V – непустое множество, элементы которого называются *вершинами*, и E – произвольное множество пар (v_i, v_j) элементов из множества V, т. е. $v_i \in V$, $v_j \in V$, $E \subseteq V^2$. Элементы множества E называются *ребрами*.

Само понятие графа подразумевает графическое представление данного объекта. Вершины изображаются точками, а ребра – линиями, соединяющими эти точки. Если ребра представляют упорядоченные пары вершин, соответствующие линии изображаются стрелками (рис. 3.1). Такие ребра называют *ориентированными ребрами* или, чаще, *дугами*. В этом случае имеем дело с *ориентированным* графом в отличие от *неориентированного* графа, на ребрах которого порядок вершин не задан.

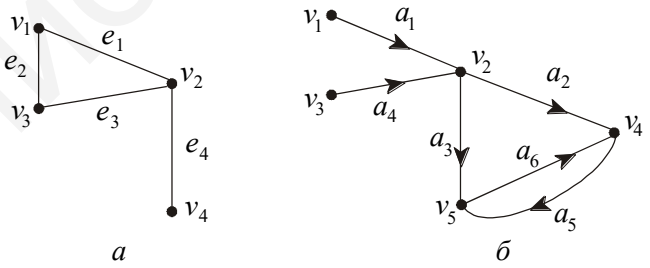


Рис. 3.1. Примеры графов:
а – неориентированный; *б* – ориентированный

Вершины неориентированного графа, связываемые ребром, считаются *концами* этого ребра. Например, концами ребра e_2 графа на рис. 3.1, *а* являются вершины v_1 и v_3. Принято обозначать ребра также парами их концов, например $e_2 = v_1v_3$. Всякая упорядоченная пара вершин (v_i, v_j), представляющая дугу в ориентированном графе, имеет *начало* v_i и *конец* v_j. Говорят, что дуга *выходит* из начала и *входит* в конец. В ориентированном графе на рис. 3.1, *б* началом

дуги a_4 является вершина v_3, концом – вершина v_2. Это можно представить как $a_4 = (v_3, v_2)$.

Между вершинами и ребрами неориентированного графа, так же как между вершинами и дугами ориентированного графа, существует отношение *инцидентности*. При этом в неориентированном графе $G = (V, E)$ вершина $v \in V$ и ребро $e \in E$ *инцидентны*, если v является одним из концов ребра e. В ориентированном графе $G = (V, A)$ вершина $v \in V$ и дуга $a \in A$ *инцидентны*, если v является началом либо концом дуги a. Две вершины неориентированного графа *смежны*, если они инцидентны одному и тому же ребру.

Граф может содержать *петли*, т. е. ребра, концы которых совпадают, или дуги, у которых начало совпадает с концом. Очевидно, ориентация петли несущественна.

Множество всех вершин графа G, смежных с вершиной v, называется *окрестностью* вершины v и обозначается символом $N(v)$. Мощность множества $N(v)$, обозначаемая $d(v)$, называется *степенью* вершины v. В ориентированном графе с некоторой вершиной v подобным образом связаны два множества: *полуокрестность исхода* $N^+(v)$ – множество вершин, в которые входят дуги, исходящие из вершины v, и *полуокрестность захода* $N^-(v)$ – множество вершин, из которых исходят дуги, заходящие в v. Соответственно мощность множества $N^+(v)$ называется *полустепенью исхода* и обозначается $d^+(v)$, а мощность множества $N^-(v)$ – *полустепенью захода* и обозначается $d^-(v)$. Можно говорить об окрестности $N(v)$ и степени $d(v)$ вершины v ориентированного графа. При этом

$$N(v) = N^+(v) \cup N^-(v) \quad \text{и} \quad d(v) = d^+(v) + d^-(v).$$

Для неориентированного графа с множеством ребер E очевидно следующее соотношение:

$$\sum_{v \in V} d(v) = 2|E|,$$

откуда следует, что в любом неориентированном графе число вершин с нечетной степенью всегда четно.

Для ориентированного графа с множеством дуг A имеем

$$\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = |A|.$$

В практических приложениях граф (ориентированный или неориентированный), как правило, является *конечным*, т. е. его множество вершин конечно. Специальный раздел теории графов изучает также *бесконечные графы*, у которых множество вершин бесконечно.

Граф $G = (V, E)$, у которого множество ребер пусто, т. е. $E = \varnothing$, называется *пустым* графом. Неориентированный граф называется *полным*, если любые две его вершины смежны. Полный граф, число вершин которого n, обозначается символом K_n.

Обозначим множество ребер полного графа символом U. *Дополнением* графа $G = (V, E)$ является граф $\bar{G} = (V, \bar{E})$, у которого $\bar{E} = U \setminus E$. Очевидно, что всякий полный граф является дополнением некоторого пустого графа и, наоборот, всякий пустой граф является дополнением некоторого полного графа.

Граф называется *двудольным*, если множество его вершин V разбито на два непересекающихся подмножества V' и V'', а концы любого его ребра находятся в различных подмножествах. Такой граф задается как $G = (V', V'', E)$ или как $G = (V', V'', A)$. В *полном двудольном* графе (V', V'', E) каждая вершина из V' связана ребром с каждой вершиной из V''. Полный двудольный граф, у которого $|V'| = p$ и $|V''| = q$, обозначается символом $K_{p, q}$.

3.2. Графическое представление бинарного отношения

Граф в том виде, как он определен в подразд. 3.1, является, по сути, графическим представлением бинарного отношения. Пусть задано бинарное отношение $R \subseteq A \times B$. Если $A \cap B = \varnothing$, то данное отношение можно представить двудольным ориентированным графом $G = (A, B, R)$, где каждая пара $(a, b) \in R$ представляется дугой, исходящей из вершины a и заходящей в вершину b. На рис. 3.2 представлено отношение R между элементами множеств A и B, где $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3, b_4, b_5\}$, $R = \{(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_1, b_5), (a_2, b_2), (a_2, b_4), (a_3, b_3)\}$.

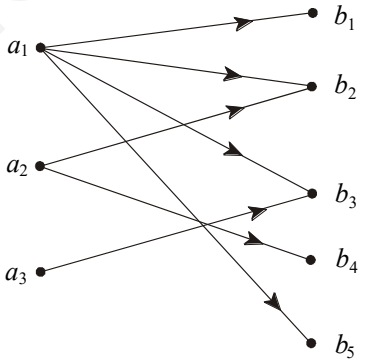


Рис. 3.2. Графическое представление отношения между элементами множеств A и B

Операция композиции отношений, рассмотренная в разд. 2, проиллюстрирована на рис. 3.3, где отношение R между элементами множеств A и B и отношение S между элементами множеств B и C показаны совместно (рис. 3.3, *а*). В представлении отношения SR на рис. 3.3, *б* видно, что вершина $a \in A$ соединена с вершиной $c \in C$ дугой тогда и только тогда, когда существует

вершина $b \in B$, которая в графе на рис. 3.3, *а* является концом дуги, исходящей из a, и началом дуги, заходящей в c.

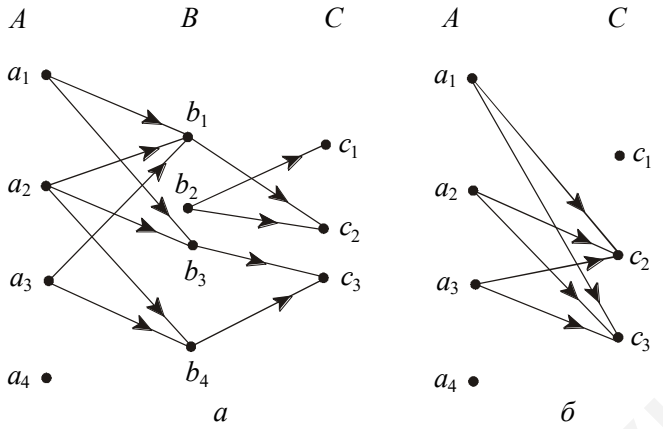


Рис. 3.3. Представление композиции отношений:
а – отношения R и S; *б* – отношение SR

В графическом представлении функционального отношения $R = \{(a, b), (c, b), (b, d), (e, d), (d, d)\}$ между элементами множеств $A = \{a, b, c, d, e\}$ и $B = \{b, d, e\}$, рассмотренных в разд. 2, из каждой вершины выходит только одна дуга, включая петли (рис. 3.4).

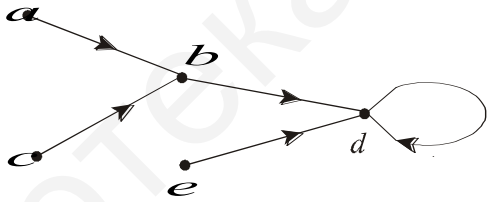


Рис. 3.4. Представление функционального отношения

3.3. Матричные представления графа

Поскольку граф можно рассматривать как графическое представление некоторого бинарного отношения, его можно задать той же булевой матрицей, которая задает данное отношение (описана в разд. 2). Эта матрица называется *матрицей смежности* графа. Строки и столбцы матрицы смежности соответствуют вершинам графа, а элемент ее на пересечении строки v_i и столбца v_j имеет значение 1 тогда и только тогда, когда вершины v_i и v_j смежны. В матрице смежности ориентированного графа этот элемент имеет значение 1, если и только если в данном графе имеется дуга с началом в вершине v_i и концом в вершине v_j. Графы, показанные на рис. 3.1, имеют следующие матрицы смежности:

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}, \quad \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}.$$

Нетрудно видеть, что матрица смежности неориентированного графа обладает симметрией относительно главной диагонали.

Ориентированный граф можно задать также *матрицей инцидентности*, которая определяется следующим образом. Ее строки соответствуют вершинам графа, столбцы – дугам. Элемент на пересечении строки *v* и столбца *a* имеет значение 1, если вершина *v* является началом дуги *a*, и значение –1, если *v* является концом дуги *a*. Если вершина *v* и дуга *a* не инцидентны, то указанный элемент имеет значение 0. Матрица инцидентности неориентированного графа имеет тот же вид, только в ней все значения –1 заменяются на 1. Матрицы инцидентности графов, представленных на рис. 3.1, будут иметь следующий вид:

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}, \quad \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 & -1 & 1 \end{bmatrix} \end{array}.$$

Заметим, что при матричном представлении графа его вершины, а также ребра или дуги оказываются упорядоченными. Любая строка матрицы смежности является векторным представлением окрестности соответствующей вершины. Любой столбец матрицы инцидентности неориентированного графа содержит ровно две единицы. Сумма значений элементов любого столбца матрицы инцидентности ориентированного графа равна нулю.

3.4. Части графа

Граф $H = (W, F)$ называется *подграфом* графа $G = (V, E)$, если $W \subseteq V$, $F \subseteq E$ и обе вершины, инцидентные любому ребру из F, принадлежат W. Подграф H графа G называется его *остовным подграфом*, если $W = V$. Если F является множеством всех ребер графа G, все концы которых содержатся в множестве W, то подграф $H = (W, F)$ называется *подграфом, порожденным множеством* W.

Любая последовательность вида $v_1, e_1, v_2, e_2, \dots, e_k, v_{k+1}$, где $v_1, v_2, \dots, v_{k+1}$ – вершины некоторого графа, а $e_1, e_2, \dots, e_k$ – его ребра, причем $e_i = v_i v_{i+1}$ ($i = 1, 2, \dots, k$), называется *маршрутом*. Маршрут может быть конечным либо бесконечным. Одно и то же ребро может встречаться в маршруте не один раз. *Длиной маршрута* называется количество входящих в него ребер, причем каждое ребро считается столько раз, сколько оно встречается в данном маршруте.

Маршрут, все ребра которого различны, называется *цепью*. Цепь, все вершины которой различны, называется *простой цепью*. С понятием длины цепи связано понятие *расстояния* в графе. Под расстоянием между двумя вершинами понимается длина кратчайшей цепи, связывающей данные вершины.

Маршрут $v_1, e_1, v_2, e_2, \dots, e_k, v_1$ называется *циклическим*. Циклическая цепь называется *циклом*. *Простой цикл* – это циклическая простая цепь.

Любую цепь и любой цикл графа можно рассматривать как его подграф.

Граф является *связным*, если между любыми двумя его вершинами имеется цепь. Связный подграф некоторого графа, не содержащийся ни в каком другом его связном подграфе, называется *компонентой связности* или просто *компонентой* данного графа.

В ориентированном графе *маршрутом* называется последовательность вида $v_1, a_1, v_2, a_2, \dots, a_k, v_{k+1}$, где для всякой дуги a_i вершина v_i является началом, а v_{i+1} – концом. Вершина v_1 является началом маршрута, а вершина v_{k+1} – его концом. Маршрут, в котором все вершины, кроме, возможно, начальной и конечной, различны, называется *путем*. Путь вида $v_1, a_1, v_2, a_2, \dots, a_k, v_1$ называется *контуром*.

Вершина v_j в ориентированном графе является *достижимой* из вершины v_j, если в этом графе имеется путь с началом в v_i и концом в v_j. Ориентированный граф является *сильно связным*, если любая его вершина достижима из любой вершины.

Ориентированный граф называется *транзитивным*, если из существования дуг $a_p = (v_i, v_j)$ и $a_q = (v_j, v_k)$ следует существование дуги $a_r = (v_i, v_k)$. *Транзитивным замыканием* ориентированного графа $G = (V, A)$ называется граф $G^* = (V, A^*)$, где A^* получено из A добавлением минимально возможного количества дуг, необходимого для того, чтобы граф G^* был транзитивным.

3.5. Обобщения графов

Существуют различные обобщения понятия графа. Одним из таких обобщений является *мультиграф*. Это граф, в котором любые две вершины могут быть связаны любым количеством ребер, т. е. мультиграф допускает *кратные ребра*.

В некоторых задачах используются графы, на множествах вершин или ребер которых заданы функции, принимающие значения из множеств действительных, целых или натуральных чисел. Эти значения называются *весами*. То-

гда речь идет о *взвешенных графах*, о графах со *взвешенными вершинами*, со *взвешенными ребрами* или со *взвешенными дугами*. Графы со взвешенными ребрами используются в транспортных задачах и задачах о потоках в сетях. Мультиграф можно рассматривать как граф, ребра которого взвешены натуральными числами, представляющими кратности ребер.

Иногда рассматриваются *смешанные графы*, у которых наряду с элементами ориентированного графа (дугами) имеются элементы неориентированного графа (ребра). Ребром может быть заменена пара противоположно направленных дуг в ориентированном графе, соединяющих одни и те же вершины. Смешанные графы используются при решении задач, связанных с установлением схемы выполнения операций в технологическом процессе.

Еще одним обобщением понятия графа является *гиперграф*, который также представляет собой два множества – множество вершин и множество ребер, однако если ребром графа является пара вершин, то ребром гиперграфа может быть любое непустое подмножество множества вершин.

Гиперграф может служить моделью принципиальной электрической схемы. При этом полюса элементов данной схемы соответствуют вершинам гиперграфа, а электрические цепи – ребрам. Электрическая цепь здесь рассматривается как множество выводов, соединенных между собой проводниками. Многие понятия, связанные с графами, распространяются на случай гиперграфа, однако графически изобразить гиперграф гораздо труднее, чем граф. Вместе с тем от гиперграфа можно перейти к двудольному графу, долями которого являются множество вершин и множество ребер гиперграфа, а ребра показывают принадлежность вершин гиперграфа его ребрам.

3.6. Доминирующие множества графа

Подмножество S множества вершин V графа G называется *доминирующим множеством* графа G, если выполняется условие $S \cup N(S) = V$, где $N(S) = \bigcup_{v \in S} N(v)$, а $N(v)$ – множество вершин, смежных с вершиной v. Другими словами, множество S является доминирующим, если каждая вершина из множества $V \setminus S$ смежна с некоторой вершиной из S. Если S является доминирующим множеством некоторого графа G, то всякое множество вершин $S' \supseteq S$ этого графа также является доминирующим. Поэтому представляет интерес задача нахождения *минимальных* доминирующих множеств, т. е. таких, у которых ни одно собственное подмножество не является доминирующим. Доминирующее множество, имеющее наименьшую мощность, принято называть *наименьшим*. Эта мощность называется *числом доминирования* графа G и обозначается символом $\beta(G)$.

Нетрудно видеть, что наименьшими доминирующими множествами графа G (рис. 3.5) являются $\{v_3, v_7\}$, $\{v_5, v_7\}$ и $\{v_6, v_7\}$.

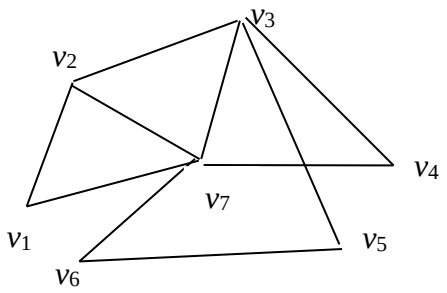


Рис. 3.5. Граф *G*

Его матрица смежности, дополненная единицами на главной диагонали, имеет вид

$$\begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & \\ \left[\begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{matrix}\right] & \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{matrix} \end{matrix}.$$

Каждое из соответствующих множеств строк рассматриваемой матрицы покрывает все ее столбцы.

Наглядным примером задачи о наименьшем доминирующем множестве является одна из задач о ферзях, где надо расставить на шахматной доске наименьшее число ферзей так, чтобы каждая клетка была под ударом хотя бы одного из них. Для этого достаточно найти наименьшее доминирующее множество в графе, вершины которого соответствуют клеткам шахматной доски и две вершины связаны ребром, если и только если они взаимно достижимы ходом ферзя. Найденное множество вершин указывает, куда надо поставить ферзей, а их число, которое в данном случае равно пяти, есть число доминирования данного графа.

Задача о наименьшем доминирующем множестве сводится к известной задаче о *кратчайшем покрытии*, которая подробно будет рассмотрена далее. В данном случае ее можно поставить следующим образом. Матрицу смежности заданного графа *G* дополним единицами на главной диагонали. Тогда требуется найти минимальную совокупность строк полученной матрицы, такую, чтобы каждый столбец имел единицу по крайней мере в одной из строк найденной совокупности. Говорят, что строка матрицы *покрывает* столбец, если данный столбец имеет единицу в этой строке.

3.7. Независимые множества графа

Подмножество S множества вершин V графа G называется *независимым множеством* графа G, если выполняется условие $S \cap N(S) = \varnothing$, т. е. любые две вершины из S не смежны. Если S – независимое множество, то любое его подмножество также является независимым. Поэтому представляет интерес задача нахождения в графе *максимальных независимых множеств*, т. е. таких, которые не являются собственными подмножествами каких-либо других независимых множеств.

Независимое множество, имеющее наибольшую мощность среди всех независимых множеств графа G, называют *наибольшим независимым множеством*, а его мощность называют *числом независимости* графа G и обозначают символом $\alpha(G)$.

Примером задачи нахождения наибольшего независимого множества является другая задача о ферзях, в которой надо расставить на шахматной доске наибольшее число ферзей так, чтобы ни один из них не находился под ударом другого. Наибольшее независимое множество графа, представляющего шахматную доску, как определено ранее, покажет, на какие клетки надо поставить ферзей. Наибольшее число ферзей, расставленных при указанном условии, которое в данном случае равно восьми, есть число независимости данного графа.

Рассмотрим один из способов нахождения в графе всех максимальных независимых множеств [5].

Пусть G – заданный граф с произвольно упорядоченным множеством вершин $V = \{v_1, v_2, \ldots, v_n\}$. Рассмотрим последовательность подграфов $G_1, G_2, \ldots, G_n$, порожденных подмножествами $V_1, V_2, \ldots, V_n$, где $V_i = \{v_1, v_2, \ldots, v_i\}$ $(i = 1, 2, \ldots, n)$. Пусть $S^i = \{S_1^i, S_2^i, \ldots, S_{k_i}^i\}$ – совокупность всех максимальных независимых множеств графа G_i. Преобразуем S^i следующим образом. Для каждого S_j^i $(j = 1, 2, \ldots, k_i)$ получим множество

$$S' = (S_j^i \setminus N(v_{i+1})) \cup \{v_{i+1}\}.$$

Если в S^i найдется такой элемент S_l^i, что $S_l^i \subset S'$, то S_l^i в S^i заменяем на S'. Если найдется такой элемент S_l^i в множестве S^i, что $S' \subseteq S_l^i$, то S^i не изменяем. В остальных случаях S' добавляем в множество S^i в качестве нового элемента.

Нетрудно убедиться, что в результате таких преобразований множество S^i превращается в S^{i+1} – совокупность всех максимальных независимых множеств графа G_{i+1}. Действительно, все $S_1^i, S_2^i, \ldots, S_{k_i}^i$ являются независимыми множествами для G_{i+1}. Множество S' является также независимым. Все поглощаемые множества удаляются, так что остаются только максимальные. То, что S^{i+1} содержит все максимальные независимые множества графа G_{i+1}, легко доказывается от противного. Пусть S'' – максимальное независимое множество графа G_{i+1}, не полученное в результате описанных преобразований. Но тогда

$S''\setminus\{v_{i+1}\}$ является максимальным независимым множеством графа G_i, которого нет в S^i, что противоречит определению множества S^i.

Начиная от $S^1 = \{\{v_1\}\}$, выполним цепочку таких преобразований, в результате чего получим $S^n = S$ – совокупность всех максимальных независимых множеств графа $G_n = G$.

Для графа, изображенного на рис. 3.5, имеем последовательность результатов применения описанного преобразования, где S^7 представляет совокупность всех максимальных независимых множеств:

$S^2 = \{\{v_1\}, \{v_2\}\}$;
$S^3 = \{\{v_1, v_3\}, \{v_2\}\}$;
$S^4 = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}\}$;
$S^5 = \{\{v_1, v_3\}, \{v_1, v_4, v_5\}, \{v_2, v_4, v_5\}\}$;
$S^6 = \{\{v_1, v_3, v_6\}, \{v_1, v_4, v_5\}, \{v_1, v_4, v_6\}, \{v_2, v_4, v_5\}, \{v_2, v_4, v_6\}\}$;
$S^7 = \{\{v_1, v_3, v_6\}, \{v_1, v_4, v_5\}, \{v_1, v_4, v_6\}, \{v_2, v_4, v_5\}, \{v_2, v_4, v_6\}, \{v_5, v_7\}\}$.

Иногда требуется найти одно, но наибольшее независимое множество. Можно для этого получить все максимальные независимые множества и из них выбрать наибольшее. Тогда интересно знать верхнюю границу числа максимальных независимых множеств в графе с n вершинами. Известно, что она выражается следующими формулами, где k – некоторое положительное целое число:

$2 \cdot 3^{k-1}$, если $n = 3k - 1$;
$3 \cdot 3^{k-1}$, если $n = 3k$;
$4 \cdot 3^{k-1}$, если $n = 3k + 1$.

Экстремальным графом, т. е. графом, в котором достигается указанная граница, для случая $n = 3k$ является показанный на рис. 3.6 несвязный граф, состоящий из k компонент. Для случая $n = 3k - 1$ или $n = 3k + 1$ один из треугольников заменяется соответственно на изолированное ребро или полный четырехвершинный граф.

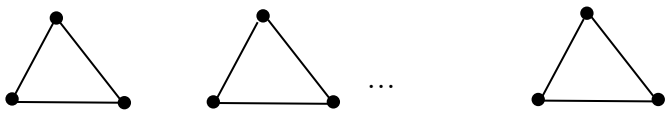


Рис. 3.6. Экстремальный граф для максимальных независимых множеств

Верхняя граница числа максимальных независимых множеств растет во много раз быстрее, чем число вершин графа. Поэтому представляет интерес способ получения максимальных независимых множеств, не требующий их од-

новременного представления. Рассмотрим такой способ, получающий эти множества в лексикографическом порядке.

Пусть $V = \{v_1, v_2, \ldots, v_n\}$ – множество вершин графа G. Весь процесс нахождения максимальных независимых множеств можно разбить на n этапов, каждый из которых связан с определенной вершиной $v_i \in V$. На i-м этапе находятся все максимальные независимые множества, содержащие вершину v_i и не содержащие вершин с меньшими номерами, т. е. таких v_j, для которых $j < i$.

Пусть S_i – одно из независимых множеств графа G, формируемых на i-м этапе. За начальное значение множества S_i принимается множество, состоящее из единственной вершины v_i. Множество S_i расширяется за счет поочередного включения в него элементов $v_j \in V$, удовлетворяющих следующим условиям:

$$i < j \le n \quad \text{и} \quad v_j \in \bigcap_{v \in S_i} \{V \setminus N(v)\}.$$

Каждый раз при соблюдении этих условий выбирается v_j с минимальным j. Это расширение множества S_i продолжается до тех пор, пока множество $\bigcap_{v \in S_i} \{V \setminus N(v)\}$ не станет пустым.

Результат расширения S_i проверяется на максимальность согласно следующему свойству: независимое множество S является максимальным в том и только в том случае, когда $S \cup N(S) = V$.

Действительно, если это равенство не выполняется, то найдется вершина в множестве $V \setminus S$, не смежная ни с одной вершиной из S. Присоединив ее к S, получим независимое множество, содержащее S в качестве собственного подмножества, т. е. S не максимально. Если же это равенство выполняется, то не существует вершины в графе G, которая была бы не смежна ни с одной вершиной из S и ее можно было бы добавить к S, не нарушая независимости S.

Множество, прошедшее такую проверку, включается в решение. Чтобы построить следующее по порядку независимое множество, из полученного S_i (включенного или не включенного в решение) удаляется вершина v_p, присоединенная к S_i последней, и выполняется та же процедура с вершинами v_q, где $q > p$.

За n этапов можно получить все максимальные независимые множества заданного графа. Однако данный процесс можно прекратить на k-м этапе, где $k < n$, если видно, что из оставшихся $n - k$ вершин не получится максимального независимого множества. Для этого надо для каждой вершины v_i сформировать множества $A_i = \{v_i, v_{i+1}, \ldots, v_n\}$ и $B_i = \bigcup_{j=i}^{n} N(v_j)$ и на очередном i-м этапе проверить условие $A_i \cup B_i = V$. Если оно не выполняется, то данный процесс надо прекратить.

Для графа на рис. 3.5 множества S_i ($i = 1, 2, 3, 4, 5, 6$) будут принимать следующие значения (этап, связанный с вершиной v_7, не выполняется, так как $A_7 \cup B_7 = \{v_7\} \cup \{v_1, v_2, v_3, v_4, v_6\} \neq V$):

S_1: $\{v_1\}$, $\{v_1, v_3\}$, $\underline{\{v_1, v_3, v_6\}}$, $\{v_1, v_4\}$, $\underline{\{v_1, v_4, v_5\}}$, $\underline{\{v_1, v_4, v_6\}}$, $\{v_1, v_5\}$, $\{v_1, v_6\}$;
S_2: $\{v_2\}$, $\{v_2, v_4\}$, $\underline{\{v_2, v_4, v_5\}}$, $\underline{\{v_2, v_4, v_6\}}$, $\{v_2, v_5\}$, $\{v_2, v_6\}$;
S_3: $\{v_3\}$, $\{v_3, v_6\}$;
S_4: $\{v_4\}$, $\{v_4, v_5\}$, $\{v_4, v_6\}$;
S_5: $\{v_5\}$, $\underline{\{v_5, v_7\}}$;
S_6: $\{v_6\}$.

Здесь подчеркнуты те значения S_i, которые вошли в решение.

Для получения одного наибольшего независимого множества рассмотренный способ удобнее, чем предыдущий. Хранить все максимальные независимые множества одновременно не надо. Они получаются последовательно, и каждый раз следует оставлять только те из них, которые обладают наибольшей мощностью.

Вершинным покрытием графа $G = (V, E)$ называется такое множество $B \subseteq V$, что каждое ребро из E инцидентно хотя бы одной вершине из B. Очевидно, что если B – вершинное покрытие, то $V \setminus B$ – независимое множество. *Вершинное покрытие* наименьшей мощности в графе G является его *наименьшим вершинным покрытием*. Ясно, что если B – наименьшее вершинное покрытие, то $V \setminus B$ – наибольшее независимое множество.

Чтобы найти наименьшее вершинное покрытие в графе G, достаточно покрыть столбцы его матрицы инцидентности минимальным количеством ее строк. Матрица инцидентности графа на рис. 3.7 имеет следующий вид:

$$\begin{array}{cccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} & \\ \left[\begin{array}{cccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] & \begin{array}{c} v_1 \\ v_1 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \end{array}.$$

Строки v_1, v_3, v_5 и v_7 составляют кратчайшее покрытие этой матрицы. Множество вершин $\{v_1, v_3, v_5, v_7\}$ является наименьшим вершинным покрытием данного графа. Следовательно, $\{v_2, v_4, v_6\}$ – одно из наибольших независимых множеств. Оно присутствует в решении предыдущего примера.

Нахождение наибольшего независимого множества – довольно трудоемкая задача. Поэтому иногда довольствуются получением максимального неза-

висимого множества, по мощности близкого к наибольшему. Одним из способов решения такой задачи является чередование следующих двух операций: выбора вершины с наименьшей степенью в качестве элемента искомого множества и удаления выбранной вершины из графа вместе с окрестностью. В результате может получиться наибольшее независимое множество, однако, как видно на примере графа, приведенного на рис. 3.7, это бывает не всегда.

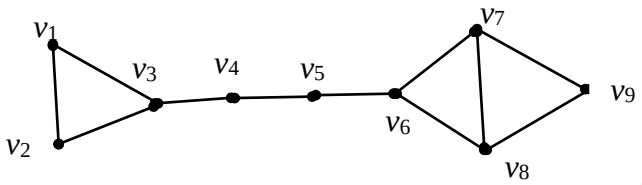


Рис. 3.7. Граф для демонстрации получения независимого множества

Действительно, в качестве первой вершины для включения в решение может быть выбрана вершина v_5. Тогда вслед за ней в решение включаются вершины v_3 и v_7. Мощность полученного таким образом множества на единицу меньше мощности наибольшего независимого множества $\{v_1, v_4, v_6, v_9\}$.

Подграф графа G, порождаемый его независимым множеством, является *пустым подграфом*, т. е. множество его ребер пусто. В графе $\bar{G}$, который является дополнением графа G, это же множество порождает *полный подграф*, т. е. подграф графа $\bar{G}$, в котором все вершины попарно смежны в графе $\bar{G}$. Полный подграф называют еще *кликой*. Таким образом, задача нахождения полных подграфов, или клик, и задача нахождения независимых множеств в некотором графе являются двойственными по отношению друг к другу.

3.8. Постановка задачи о раскраске графа

Раскраской некоторого графа $G = (V, E)$ называется такое разбиение множества вершин V на непересекающиеся подмножества $V_1, V_2, \ldots, V_k$, что никакие две вершины из одного, любого, из этих подмножеств не смежны. Считается, что вершины, принадлежащие одному и тому же подмножеству V_i, выкрашены при этом в один и тот же цвет i. Задача состоит в том, чтобы раскрасить вершины графа G в минимальное число цветов. Оно называется *хроматическим числом* графа и обозначается $\chi(G)$.

Задача раскраски графа имеет много приложений в различных областях человеческой деятельности. К задаче раскраски сводятся составление расписания занятий в учебном заведении, распределение оборудования на предприятии, выбор расцветки проводов в сложных электрических схемах и многие другие практические задачи. Например, при минимизации числа ячеек памяти вычислительной машины для хранения промежуточных результатов в процессе выполнения программы можно построить граф, вершины которого соответствуют промежуточным результатам вычисления и две вершины связаны ребром, если

и только если соответствующие величины используются одновременно и не могут храниться в одной и той же ячейке. Хроматическое число данного графа представляет минимум упомянутых ячеек.

Иногда ставится задача раскраски ребер графа $G = (V, E)$, где требуется получить такое разбиение множества ребер E на непересекающиеся подмножества $E_1, E_2, \dots, E_p$, что ни одна пара ребер из одного и того же E_i ($i = 1, 2, \dots, p$) не имеет общей инцидентной вершины. Данная задача сводится к раскраске вершин. Для этого надо построить *реберный граф* $L(G)$ графа G. Вершины графа $L(G)$ соответствуют ребрам графа G, и две вершины графа $L(G)$ связаны ребром, если и только если соответствующие ребра графа G имеют общую инцидентную вершину в G. Раскраска ребер графа G соответствует раскраске вершин графа $L(G)$.

3.9. Метод раскраски графа

Очевидно, всякое множество одноцветных вершин графа является независимым множеством. Поэтому получить минимальную раскраску можно следующим образом: найти все максимальные независимые множества; получить кратчайшее покрытие множества вершин графа максимальными независимыми множествами; удалить некоторые вершины из элементов полученного покрытия, добившись того, чтобы каждая вершина входила в одно и только в одно из выделенных независимых множеств. Для графов, число независимых множеств которых невелико, этот способ является приемлемым. Однако, как показывает оценка, приведенная в разд. 2, это число для некоторых графов может оказаться настолько большим, что данный способ вообще не сможет быть реализован. Существует немало методов раскраски, не использующих задачу покрытия и получающих точно минимальное число цветов, но и их применение существенно ограничено размерностью задачи.

Рассмотрим один из методов раскраски графа, который не гарантирует получения минимума цветов, но дает раскраску, близкую к минимальной, а во многих случаях совпадающую с ней [5].

Процесс раскраски графа $G = (V, E)$ представляет собой последовательность шагов, на каждом из которых выбирается вершина и окрашивается в определенный цвет. Текущая ситуация характеризуется следующими объектами: k – число задействованных цветов; A – множество еще не раскрашенных вершин; $B_1, B_2, \dots, B_k$ – совокупность подмножеств множества вершин V, такая, что B_i ($i = 1, 2, \dots, k$) содержит те и только те вершины из множества A, которые нельзя раскрасить в i-й цвет. Обратим внимание на следующие два случая:

1. Имеется вершина $v \in A$, такая, что $v \in B_i$ для всех $i = 1, 2, \dots, k$.

2. Имеется вершина $v \in A$ и цвет i такие, что $v \notin B_i$ и $N(v) \cap A \subseteq B_i$.

В первом случае вершину v надо красить в $(k + 1)$-й цвет, удалить ее из множества A и из всех множеств B_i, где она была, сформировать множество

B_{k+1} и увеличить k на единицу. Если таких вершин несколько, из них выбирается та, для которой множество B_{k+1} имеет максимальную мощность.

Во втором случае вершину v надо красить в i-й цвет, удалить ее из множества A и из всех множеств B_j, где она была.

Во всех остальных случаях из множества A выбираются вершина v и цвет i такие, что $v \notin B_i$ и приращение $\Delta|B_i|$ мощности множества B_i минимально среди всех пар v, B_i ($v \in A$, $i = 1, 2, \ldots, k$). Вершина v удаляется из A и из всех B_j, где она была, и красится в i-й цвет.

Выполнение описанных действий повторяется до тех пор, пока множество A не станет пустым.

В начальной ситуации $A = V$, $k = 0$ и рекомендуется выбирать вершину с максимальной степенью и красить ее в цвет 1, а множество B_1 будет представлять ее окрестность.

Продемонстрируем применение данного метода на примере графа, изображенного на рис. 3.8, представив далее результаты выполнения последовательности шагов.

Шаг 1: $\{v_1\}$; $B_1 = \{v_2, v_6, v_8, v_{10}\}$; $A = \{v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$.

Шаг 2 (первый случай: $v_2 \in B_1$): $\{v_1\}$, $\{v_2\}$; $B_1 = \{v_6, v_8, v_{10}\}$, $B_2 = \{v_4, v_5\}$; $A = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$.

Шаг 3 (второй случай: $N(v_8) = \varnothing$): $\{v_1\}$, $\{v_2, v_8\}$; $B_1 = \{v_6, v_{10}\}$, $B_2 = \{v_4, v_5\}$; $A = \{v_3, v_4, v_5, v_6, v_7, v_9, v_{10}\}$.

Шаг 4 (выбор v_7, цвет 1, $\Delta|B_1| = 1$): $\{v_1, v_7\}$, $\{v_2, v_8\}$; $B_1 = \{v_3, v_6, v_{10}\}$, $B_2 = \{v_4, v_5\}$; $A = \{v_3, v_4, v_5, v_6, v_9, v_{10}\}$.

Шаг 5 (выбор v_3, цвет 2, $\Delta|B_2| = 1$): $\{v_1, v_7\}$, $\{v_2, v_3, v_8\}$; $B_1 = \{v_6, v_{10}\}$, $B_2 = \{v_4, v_5, v_6\}$; $A = \{v_4, v_5, v_6, v_9, v_{10}\}$.

Шаг 6 (первый случай: $v_6 \in B_i$, $i = 1, 2$): $\{v_1, v_7\}$, $\{v_2, v_3, v_8\}$, $\{v_6\}$; $B_1 = \{v_{10}\}$, $B_2 = \{v_4, v_5\}$, $B_3 = \{v_9\}$; $A = \{v_4, v_5, v_9, v_{10}\}$.

Шаг 7 (выбор v_5, цвет 3, $\Delta|B_3| = 1$): $\{v_1, v_7\}$, $\{v_2, v_3, v_8\}$, $\{v_5, v_6\}$; $B_1 = \{v_{10}\}$, $B_2 = \{v_4\}$, $B_3 = \{v_4, v_9\}$; $A = \{v_4, v_9, v_{10}\}$.

Шаг 8 (второй случай: $N(v_{10}) \cap A \subseteq B_3$): $\{v_1, v_7\}$, $\{v_2, v_3, v_8\}$, $\{v_5, v_6, v_{10}\}$; $B_1 = \varnothing$, $B_2 = \{v_4\}$, $B_3 = \{v_4, v_9\}$; $A = \{v_4, v_9\}$.

Шаг 9 (выбор v_4, цвет 1, $\Delta|B_1| = 1$): $\{v_1, v_4, v_7\}$, $\{v_2, v_3, v_8\}$, $\{v_5, v_6, v_{10}\}$; $B_1 = \{v_9\}$, $B_2 = \varnothing$, $B_3 = \{v_9\}$; $A = \{v_9\}$.

Шаг 10: завершение работы с результатом $\{v_1, v_4, v_7\}$, $\{v_2, v_3, v_8, v_9\}$, $\{v_5, v_6, v_{10}\}$.

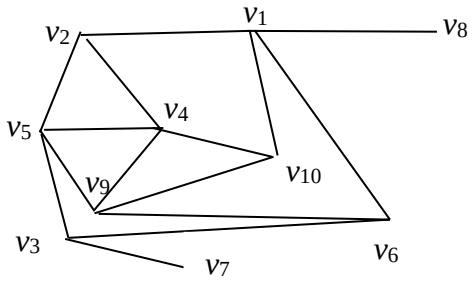


Рис. 3.8. Раскрашиваемый граф

Полученная раскраска для данного графа является минимальной, так как хроматическое число графа $\gamma(G)$ не может быть меньше числа вершин его наибольшей клики, а на рис. 3.8 видны клики, состоящие из трех вершин.

Описанный способ дает возможность иногда делать заключение о том, что полученная раскраска является минимальной. Назовем шаги, выполненные в первом и втором случаях, «хорошими», а в остальных случаях – «сомнительными». Если в процессе раскраски выполнялись только хорошие шаги, то, очевидно, полученная раскраска является минимальной. Если приходилось выполнять сомнительные шаги, то полученная раскраска может оказаться не минимальной, но по соотношению между количествами хороших и сомнительных шагов можно судить о близости полученной раскраски к минимальной.

Иногда можно получить раскраску графа, минимальную или близкую к минимальной, с помощью так называемого «*жадного*» алгоритма, где на каждом шаге в текущий цвет раскрашивается как можно больше вершин. Желательно для этого брать наибольшее независимое множество. Раскрашенные вершины удаляются из графа, вводится новый цвет, в него раскрашивается опять как можно больше вершин и так далее до тех пор, пока множество вершин графа не станет пустым. Однако есть пример графа, для которого число цветов, полученное при такой раскраске, может отличаться от минимального на сколь угодно большую величину [8].

Рассмотрим неограниченную последовательность деревьев $T_1, T_2, \ldots, T_i, \ldots$, начало которой изображено на рис. 3.9. Дерево T_1 состоит из трех вершин и двух ребер. Дерево T_i получается из T_{i-1} присоединением к каждой вершине T_{i-1} двух смежных с ней вершин. Наибольшее независимое множество дерева T_i составляют все вершины, не принадлежащие T_{i-1}. Число цветов, получаемое при раскраске дерева T_i данным способом, равно $i + 1$, хотя ясно, что всякое дерево можно раскрасить в два цвета.

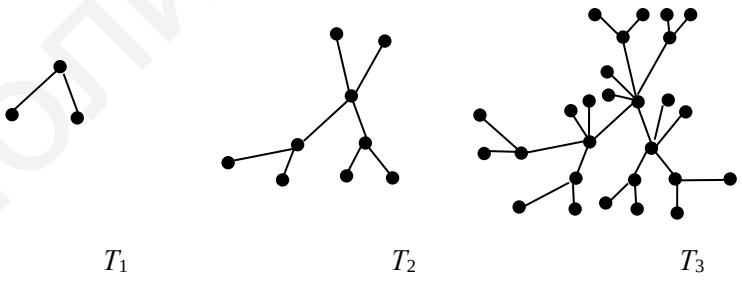


Рис. 3.9. Деревья, раскрашиваемые «жадным» алгоритмом

3.10. Бихроматические графы

Граф G называется *k-хроматическим*, если $\gamma(G) = k$. Очевидно, пустые и только пустые графы являются 1-хроматическими. Особый класс составляют *бихроматические графы*, т. е. такие, у которых $\gamma(G) = 2$.

Т е о р е м а К ё н и г а. Непустой граф является бихроматическим тогда и только тогда, когда он не содержит циклов нечетной длины.

Допустим, что $G = (V, E)$ – связный бихроматический граф. Это не нарушает общности, так как в случае несвязного графа последующие рассуждения можно провести для каждой его компоненты в отдельности. Пусть V^1 и V^2 – множества вершин графа G, раскрашенных соответственно в цвета 1 и 2. Всякое ребро соединяет вершину из V^1 с вершиной из V^2. Следовательно, всякая цепь, начинающаяся и оканчивающаяся в одном и том же множестве V^i ($i = 1, 2$), имеет четную длину. Пусть теперь G – связный граф, не имеющий циклов нечетной длины. Возьмем любую вершину v из графа G. Сформируем множество V^1 из вершин, отстоящих от v на четном расстоянии в графе G, и множество V^2 из всех остальных вершин. Ни одна пара вершин из V^2 не связана ребром. Действительно, если имелось бы ребро $v_iv_j \in E$, у которого $v_i \in V^2$ и $v_j \in V^2$, то цикл, составленный из цепи, соединяющей v с v_i, ребра v_iv_j и цепи, соединяющей v_j с v, имел бы нечетную длину, что противоречит условию.

Очевидно, всякий двудольный граф, имеющий хотя бы одно ребро, является бихроматическим, так как любая доля двудольного графа представляет собой независимое множество.

Бихроматичность графа легко установить, используя способ последовательной раскраски. Для этого произвольно выбирается вершина v и красится в цвет 1. Вершины ее окрестности $N(v)$ красятся в цвет 2, неокрашенные вершины из окрестностей вершин, принадлежащих $N(v)$, красятся в цвет 1 и т. д. В результате либо граф раскрашивается в два цвета, либо на каком-то шаге смежные вершины оказываются окрашенными в один и тот же цвет. Это говорит о том, что граф не является бихроматическим.

3.11. Методы поиска в графе

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина просматривается (посещается) в точности один раз. Поэтому важной задачей является нахождение хороших методов поиска в графе.

Под *обходом графов* (*поиском на графах*) понимается процесс систематического просмотра всех ребер или вершин графа с целью отыскания ребер или вершин, удовлетворяющих некоторому условию.

При решении многих задач, использующих графы, необходимы эффективные методы регулярного обхода вершин и ребер графов. К стандартным и наиболее распространенным методам относятся:

– поиск в глубину (Depth First Search, DFS);

– поиск в ширину (Breadth First Search, BFS).

Эти методы чаще всего рассматриваются на ориентированных графах, но они применимы и для неориентированных, ребра которых считаются двуна-

правленными. Алгоритмы обхода в глубину и в ширину лежат в основе решения различных задач обработки графов, например, построения остовного леса, проверки связности, ацикличности, вычисления расстояний между вершинами и других.

При поиске в глубину посещается первая вершина, затем необходимо идти вдоль ребер графа, до попадания в тупик. Вершина графа является *тупиком*, если все смежные с ней вершины уже посещены. После попадания в тупик нужно возвращаться назад вдоль пройденного пути, пока не будет обнаружена вершина, у которой есть еще не посещенная вершина, а затем необходимо двигаться в этом новом направлении. Процесс оказывается завершенным при возвращении в начальную вершину, причем все смежные с ней вершины уже должны быть посещены. Таким образом, основная идея поиска в глубину формулируется следующим образом: когда возможные пути по ребрам, выходящим из вершин, разветвляются, необходимо сначала полностью исследовать одну ветку и только потом переходить к другим (если они останутся нерассмотренными).

Алгоритм поиска в глубину

Шаг 1. Всем вершинам графа присваивается название «непосещенная». Выбирается первая вершина и называется «посещенной».

Шаг 2. Для последней посещенной вершины выбирается смежная не посещенная вершина, и ей присваивается название посещенной. Если таких вершин нет, то берется предыдущая посещенная вершина.

Шаг 3. Повторять шаг 2 до тех пор, пока все вершины не окажутся посещенными.

Пример исследования графа в глубину представлен на рис. 3.10.

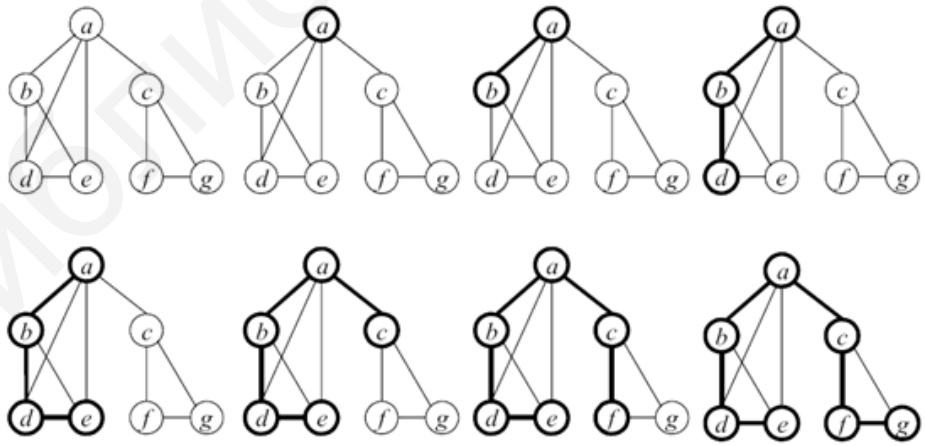


Рис. 3.10. Пример поиска в глубину

При поиске в ширину после посещения первой вершины посещаются все соседние с ней вершины. Потом посещаются все вершины, находящиеся на расстоянии двух ребер от начальной. При каждом новом шаге посещаются верши-

ны, расстояние от которых до начальной на единицу больше предыдущего. Чтобы предотвратить повторное посещение вершин, следует вести список посещенных вершин. Для хранения временных данных, необходимых для работы алгоритма, используется очередь – упорядоченная последовательность элементов, в которой новые элементы добавляются в конец, а старые удаляются из начала.

Таким образом, основная идея поиска в ширину заключается в том, что сначала исследуются все вершины, смежные с начальной (вершина, с которой начинается обход). Эти вершины находятся на расстоянии 1 от начальной. Затем исследуются все вершины на расстоянии 2 от начальной, затем все на расстоянии 3 и т. д. Обратим внимание, что при этом для каждой вершины сразу находится длина кратчайшего маршрута от начальной вершины.

Алгоритм поиска в ширину

Шаг 1. Всем вершинам графа дается название «непосещенная». Выбирается первая вершина и называется «посещенной» (и заносится в очередь).

Шаг 2. Посещается первая вершина из очереди (если она непосещенная). Все ее соседние вершины заносятся в очередь. После этого она удаляется из очереди.

Шаг 3. Повторяется шаг 2 до тех пор, пока очередь не пуста.

Пример исследования графа в ширину представлен на рис. 3.11.

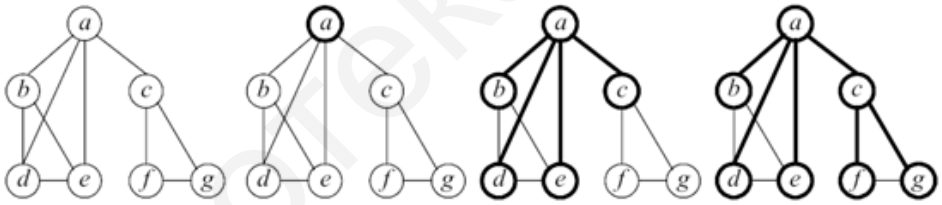


Рис. 3.11. Пример поиска в ширину

Поиск в ширину можно использовать при определении бихроматичности графа подобно тому, как это было сделано в подразд. 3.10. Произвольно выбранной вершине графа припишем индекс 1, а смежным с ней вершинам – индекс 2. Вершинам из окрестностей вершин с индексом 2 пытаемся приписать индекс 1. Затем вершинам из окрестностей вершин с индексом 1 пытаемся приписать индекс 2. Если в результате такого процесса каждая вершина примет либо индекс 1, либо индекс 2, то граф бихроматический. Если же при этом в окрестности вершины с одним индексом окажется вершина с тем же индексом, то граф не является бихроматическим. Вершине *a* в графе на рис. 3.11 припишем индекс 1, а вершинам *b, c, d* и *e* – индекс 2. Вершины *d* и *e* из окрестности вершины *b* имеют тот же индекс, что и вершина *b*. Значит, этот граф не является бихроматическим.

4. КОМБИНАТОРНЫЕ ЗАДАЧИ И МЕТОДЫ КОМБИНАТОРНОГО ПОИСКА

Область математики, в которой изучаются вопросы о том, сколько различных комбинаций, подчиненных тем или иным условиям, можно составить из заданных объектов, называется *комбинаторикой*. Объектом исследования комбинаторики являются дискретные множества. Под комбинаторной конфигурацией понимают подмножество заданного дискретного множества, удовлетворяющее некоторым свойствам.

4.1. Перечислительные задачи

В комбинаторном анализе рассматриваются задачи о расположении элементов в соответствии с точно определенными правилами, выясняется, могут ли быть осуществлены такие расположения и сколькими способами. Особенностью комбинаторных исследований является то, что в них используются преимущественно три операции: отбор подмножеств некоторого множества, упорядочение и размещение элементов этих подмножеств.

Набор элементов $a_{i_1}, a_{i_1}, ..., a_{i_r}$ множества $A = \{a_1, a_2, \ldots, a_n\}$ называется выборкой объема r из n-множества или (n,r)-выборкой. Выборки бывают упорядоченными и неупорядоченными, с повторениями и без повторений элементов. Любое допустимое расположение элементов в выборке называют решением комбинаторной задачи.

Можно выделить три типа комбинаторных задач: *задачи подсчета* числа конфигураций определенного вида; *перечислительные задачи*, в результате решения которых получаются все конструкции определенного вида (например, получение всех независимых множеств графа); *оптимизационные комбинаторные задачи*, решением любой из которых является конструкция, обладающая оптимальным значением некоторого параметра среди всех конструкций данного вида (например, раскраска графа минимальным количеством цветов).

Главными средствами для подсчета числа решений комбинаторных задач являются аналитические методы (перестановки, размещения и сочетания, производящие функции, рекуррентные соотношения) и логические методы (метод включений и исключений).

Упорядоченная (n,r)-выборка, в которой элементы могут повторяться, называется (n,r)-размещением с повторениями. Число последних будем обозначать символом $\hat{A}(n,r)$.

Число размещений показывает, сколькими способами можно разместить n предметов по r ящикам. Для каждого из n предметов имеется r вариантов размещения. Следовательно,

$$\hat{A}(n,r) = n^r.$$

Числом перестановок $P(n)$ является число различных последовательностей, которые можно составить из n предметов. В последовательности всего n позиций. Зафиксируем один предмет. Его можно разместить в одной из n позиций, т. е. имеем n вариантов размещения. Для следующего предмета имеется $n - 1$ вариантов размещения по незанятым позициям и т. д. Таким образом,

$$P(n) = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n!.$$

Приведем формулу для числа различных перестановок из n элементов, среди которых r_1 элементов первого типа, r_2 элементов второго типа, ..., r_k элементов k-го типа:

$$P_n(r_1, r_2, \ldots, r_k) = \frac{n!}{r_1! \cdot \ldots \cdot r_k!}, \qquad \sum_{i=1}^{k} r_i = n.$$

По данной формуле подсчитывается число неупорядоченных разбиений n-множества на r_i-подмножества S_i, $i = 1, \ldots, k$ такие, что $\bigcup_{i=1}^{k} S_i = S$, $S_i \cap S_j = \varnothing$, для $i \neq j$, $\sum_{i=1}^{k} r_i = n$.

Если все элементы упорядоченной (n, r)-выборки различны, она называется (n, r)-размещением без повторений $A(n, r)$. *Число размещений без повторений* представляет собой число способов размещения r предметов по n ящикам не более чем по одному в ящик (при этом считается, что $n \geq r$). Путем рассуждений, подобных предыдущим, получим

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

Число сочетаний $C(n, r)$ показывает, сколькими способами из n предметов можно выбрать r предметов. В данном случае не важно, в каком порядке эти предметы выбираются, поэтому

$$C(n, r) = \frac{A(n, r)}{n!} = \frac{n!}{r!(n-r)!}.$$

Неупорядоченная (n, r)-выборка, в которой элементы могут повторяться, называется (n, r)-сочетанием с повторениями. Их число обозначается символом $\hat{C}(n, r)$ и подсчитывается по формуле

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

При подсчете числа комбинаторных решений используются два правила.

Правило суммы: если объект A может быть выбран n способами, а объект B – другими m способами, то выбор «*либо A, либо B*» может быть осуществлен $(n + m)$ способами.

Правило произведения: если объект A может быть выбран n способами и после каждого такого выбора объект B в свою очередь может быть выбран m способами, то выбор «*A и B*» в указанном порядке может быть осуществлен $m \cdot n$ способами.

Метод производящих функций

Производящие функции были введены в комбинаторный анализ в качестве средства, позволяющего с большей общностью подходить к комбинаторным задачам перечислительного типа [19]. Рассмотрим множество элементов $X = \{x_1, x_2, \ldots, x_n\}$.

Построим полином вида

$$\prod_{k=1}^{n} (1 + x_k t) = 1 + (x_1 + x_2 + \ldots + x_n)\, t + (x_1 x_2 + x_1 x_3 + \ldots + x_1 x_n + \ldots + x_{n-1} x_n)\, t^2 + \ldots$$

$$\ldots + x_1 x_2 \ldots x_n t^n = \sum_{r=0}^{n} a_r t^r,$$

где a_0, a_1, a_2, a_r – функции от (n, r)-выборки множества X, $a_0 = 1$, $a_1 = \sum_{i=1}^{n} x_i$, $a_2 = \sum_{i1 < i2} x_{i1} x_{i2}$, $a_r = \sum_{i1 < i2 < \ldots < ir} x_{i1} x_{i2} \ldots x_{ir}$; $r = 0, \ldots, n$.

Каждый бином вида $(1 + x_k t)$ отражает две возможности:

– элемент x_k вошел в выборку один раз (слагаемое $x_k t$);

– элемент x_k не вошел в выборку ни разу (слагаемое 1).

Если все $x_k = 1$, то $\prod_{k=1}^{n} (1 + x_k t) = (1 + t)^n = \sum_{r=0}^{n} C(n, r)\, t^r$ и коэффициенты a_r являются числами (n, r)-сочетаний без повторений. Функция $f(t) = (1 + t)^n$ называется производящей функцией для последовательности чисел $C(n, r)$, $r = 0, \ldots, n$. Она однозначно связана с этой последовательностью.

Дадим общее определение производящей функции. Рассмотрим последовательность целых чисел $a = \{a_1, a_2, \ldots, a_n\}$. Ей взаимно однозначно соответствует функция $f_n(t) = \sum_{r=0}^{n} a_r t^r$, называемая обычной производящей функцией для последовательности a. Функция вида $E_n(t) = \sum \frac{a^r t^r}{r!}$ называется экспоненциальной производящей функцией для последовательности a.

При конечном n функция $f_n(t)$ соответствует неупорядоченным (n, r)-выборкам или функциям от них, а функция $E_n(t)$ – упорядоченным (n, r)-выборкам или функциям от них. С помощью производящих функций можно находить число (n, r)-выборок с заданными свойствами.

Покажем на примерах возможности использования аппарата производящих функций при решении задач на подсчет числа выборок с заданными свойствами.

Пример 1. Рассмотрим задачу о нахождении числа (n, r)-сочетаний с неограниченным числом повторений из множества элементов $X = \{x_1, x_2, \ldots, x_n\}$. Для ее решения воспользуемся полиномом вида $\Pi(x_1 t + x_2{}^2 t^2 + \ldots + x_k{}^k t^k)$, каждый сомножитель которого отражает появление элемента x_k «либо 0, либо 2, ..., либо k раз». Полагая все $x_k = 1$, приведем полином к виду

$$\Pi(1 + t + t^2 + \ldots) = (1 + t + t^2 + \ldots)^n = f_{\text{н.п}}(t).$$

Функция $f_{\text{н.п}}(t)$ является производящей функцией для искомого числа сочетаний. Поскольку $\sum_{i=0}^{n} t^n = \dfrac{1}{t-1}$ при $|t| < 1$,

$$f_{\text{н.п}}(t) = (1 - t)^{-n} = \sum_0^\infty C(n, r)\,(-t)^r =$$
$$= \sum_0^\infty ((-n)(-n-1)\ldots(-n-r+1)/r!)(-1)^r t^r = \sum_0^\infty C(n+r-1, r)t^r.$$

Из полученной формулы для производящей функции видно, что число (n, r)-сочетаний с неограниченным числом повторений равно $C(n + r - 1, r)$. Этот результат согласуется с полученным ранее.

Пример 2. В урне находятся четыре красных, пять синих и два зеленых шара.

а) Сколько существует способов выбора семи шаров из урны?

б) Сколько существует способов выбора из урны семи шаров, если один шар красный и два синие?

Для части *а* производящая функция имеет вид

$$(1 + x + x^2 + x^3 + x^4)\,(1 + x + x^2 + x^3 + x^4 + x^5)\,(1 + x + x^2),$$

и количество способов выбора семи шаров равно коэффициенту при x^7 в разложении этой производящей функции.

В части *б* нужно учесть, что один вытянутый шар красный и соответствующий многочлен имеет вид $(x + x^2 + x^3 + x^4)$, что представляет вытягивание одного, двух, трех или четырех красных шаров. Точно так же, учитывая, что два вытянутых шара синие, соответствующий многочлен должен иметь вид $(x^2 + x^3 + x^4 + x^5)$, что представляет вытягивание двух, трех, четырех или пяти синих шаров. Таким образом, производящий многочлен имеет вид

$$(x + x^2 + x^3 + x^4)(x^2 + x^3 + x^4 + x^5)(1 + x + x^2),$$

и количество способов выбора семи шаров равно коэффициенту при x^7 в разложении производящей функции.

Метод включений и исключений

Данный метод применяется при разделении множества на подмножества, элементы которых обладают определенной совокупностью свойств, а также при подсчете числа элементов в этих подмножествах. Пусть дано n-множество S некоторых элементов и m-множество свойств $P = \{p_1, p_2, \ldots, p_m\}$, которыми элементы из S могут как обладать, так и не обладать. Выделим r-выборку свойств $\{p_{i1}, p_{i2}, \ldots, p_{ir}\}$. Обозначим через $n(p_{i1}, p_{i2}, \ldots, p_{ir})$ число элементов множества S, каждый из которых обладает всеми выделенными r свойствами. Отсутствие у элемента свойства p_r будем обозначать $\bar{p}_r$. Так, число элементов, обладающих свойствами p_1, p_3, p_5 и не обладающих свойствами p_2, p_4 из выборки свойств $P = \{p_1, p_2, p_3, p_4, p_5\}$ запишется как $n(p_1, \bar{p}_2, p_3, \bar{p}_4, p_5)$. Если имеется только одно свойство p, т. е. $P = \{p\}$, то $n(\bar{p}) = n - n(p)$ число элементов n-множества, не обладающих свойством p. Если $P = \{p_1, p_2, \ldots, p_m\}$ – конечное множество свойств, несовместимых друг с другом, то число элементов, не обладающих ни одним из этих свойств, определяется формулой

$$n(\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_m) = n - \sum_{i=1}^{n} n(p_i).$$

Если $P = \{p_1, p_2, \ldots, p_m\}$ – конечное множество совместимых между собой свойств, то имеет место формула

$$n(\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_m) = n - \sum_{i=1}^{n} n(p_i) + \sum_{i_1<i_2} n(p_{i_1}, p_{i_2}) - \sum_{i_1<i_2<i_3} n(p_{i_1}, p_{i_2}, p_{i_3}) + \ldots$$

$$\ldots + (-1)^m n(p_1, p_2, \ldots, p_m),$$

известная как формула включений и исключений. В общем случае соответствующая формула имеет следующий вид:

$$n(p_1, \ldots, p_k, \bar{p}_{k+1}, \ldots, \bar{p}_m) = n(p_1, p_2, \ldots, p_k) - \sum_{l_1>k} n(p_1, \ldots, p_k, p_{l_1}) +$$

$$+ \sum_{l_1<l_2<k} n(p_1, p_2, \ldots, p_k, p_{l_1}, p_{l_2}) + \ldots + (-1)^{m-k} n(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m).$$

Область применения метода включений и исключений довольно широка. С его помощью решаются задачи на подсчет числа перестановок с запретами, задачи теории вероятностей, задачи из теории чисел.

Рассмотрим одну из задач на перестановки с запретами. Ее принято называть задачей о беспорядках. Имеется упорядоченное множество чисел 1, 2, ..., n. Из них могут быть образованы перестановки $a_1, a_2, \ldots, a_n$. Число всех возможных перестановок из n элементов равно $n!$. Перестановки, в которых ни один

элемент не сохранил своего первоначального места ($a_i \neq i$, $i = 1 + n$), называются беспорядками. Число беспорядков из n элементов можно найти по формуле включений и исключений:

$$N(\overline{P}_1, ..., \overline{P}_n) = n! - \sum N(P_{i_1}) + \sum N(P_{i_1}, P_{i_2}) + \ldots + (-1)^n N(P_{i_1}, P_{i_2}, \ldots, P_n),$$

где P_i означает свойство «i-й элемент перестановки стоит на i-м»; P – множество свойств элементов оставаться на своих местах, $P = \{P_1, P_2, \ldots, P_n\}$; N – число перестановок из n элементов, в которых на своих местах стоят k элементов, $N(P_{i_1}, P_{i_2}, \ldots, P_{i_k})$.

Если k элементов из n закрепляются на своих местах, то $N(P_{i_1}, \ldots, P_{i_k}) = (n - k)!$. При этом «неподвижных» k элементов из n можно выбрать $C(n, k)$ способами, тогда по правилу произведения получим $\sum N(P_{i_1}, \ldots, P_{i_k}) = C(n, k)(n - k)!$. Окончательно имеем формулу для числа беспорядков:

$$N(\overline{P}_1, ..., \overline{P}_n) = n! - C(n, 1)(n - 1)! + \ldots + (-1)^n C(n, n)\, 0! =$$

$$= n!\left(1 - \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{(-1)^n}{n!}\right).$$

Задачи на подсчет числа перестановок с запретами можно решать и с помощью аппарата булевых матриц.

Задаче о беспорядках можно дать следующую техническую интерпретацию: пусть проектируется дискретное устройство, состоящее из нескольких узлов и работающее в различных режимах. Каждый режим работы устройства определяется вектором $(a_{i_1}^{(1)}, a_{i_2}^{(2)}, ..., a_{i_n}^{(n)})$, компоненты которого соответствуют состояниям узлов (верхний индекс обозначает номер узла, нижний – номер состояния). В процессе проектирования устройства следует исключить запретные состояния узлов, в которых ни один номер состояния узла не соответствует номеру узла устройства.

4.2. Особенности оптимизационных комбинаторных задач

Примерами оптимизационных комбинаторных задач, решение которых предполагает комбинаторный поиск, являются рассмотренные ранее поиск наибольшего независимого и наименьшего доминирующего множеств, раскраска графа.

В отличие от задач традиционной математики, где решение строится с помощью целенаправленной вычислительной процедуры, однозначно ведущей к цели, решение комбинаторной задачи зачастую сводится к *полному перебору* различных вариантов. Перебираются и испытываются конструкции определенного вида, среди которых должно находиться решение задачи. Как только вы-

ясняется, что очередная конструкция является решением, процесс поиска решения можно считать завершенным.

В традиционной математике трудоемкость задачи обычно не очень сильно зависит от размера области возможных решений, в то время как для комбинаторных задач эта зависимость очень велика.

Комбинаторные задачи характерны еще тем, что множество, среди элементов которого отыскивается решение, всегда конечно. Реализовав полный перебор, либо найдем решение, либо убедимся в том, что решения нет. Таким образом, любая подобная задача может быть решена за конечное время. Однако это не значит, что она может быть решена за практически приемлемое время даже с помощью самой быстродействующей вычислительной машины.

4.3. Вычислительная сложность

Трудоемкость алгоритма, или временна́я сложность, т. е. время, затрачиваемое на выполнение алгоритма, оценивается числом условных элементарных операций, которые необходимо выполнить при решении задачи. Естественно, эта величина зависит от объема исходных данных, который оценивается некоторым параметром. Например, для графа это может быть число вершин или число ребер. Трудоемкость алгоритма, таким образом, можно оценить некоторой функцией $f(n)$, где n – натуральное число, выражающее объем исходных данных.

Принято писать $f(n) = O(g(n))$, где $g(n)$ – некоторая конкретная функция от n, если найдется такая константа c, что $f(n) \le cg(n)$ для любого $n \ge 0$. При этом употребляют такие выражения: «трудоемкость алгоритма есть $O(g(n))$» или «алгоритм решает задачу за время $O(g(n))$». Если трудоемкость не зависит от объема исходных данных, то для ее обозначения используется символ $O(1)$. Алгоритм трудоемкости $O(n)$ называют *линейным*. Алгоритм трудоемкости $O(n^b)$, где b – константа (возможно, дробная), называется *полиномиальным*. Если $g(n)$ является показательной функцией, например, 2^n, то говорят, что алгоритм обладает *неполиномиальной*, или *экспоненциальной*, сложностью.

Оценка трудоемкости алгоритма позволяет судить о том, как влияет быстродействие вычислительной машины на время выполнения алгоритма. Пусть имеется пять алгоритмов, трудоемкость которых соответственно n, $n \log n$, n^2, n^3 и 2^n. Пусть условная элементарная операция, которая является единицей измерения трудоемкости алгоритма, выполняется за 1 мс. В табл. 4.1, заимствованной из работы [2], показано, какого размера задачи могут быть решены каждым из этих алгоритмов за 1 с, 1 мин и 1 ч. Из этой таблицы видно, например, что за 1 мин алгоритм с трудоемкостью n^2 решает задачу в шесть раз большую, чем алгоритм с трудоемкостью n^3.

Таблица 4.1

Связь трудоемкости алгоритма с максимальным размером задачи, решаемой за единицу времени

Временна́я сложность	Максимальный размер задачи		
	1 с	1 мин	1 ч
n	1000	$6 \cdot 10^4$	$3{,}6 \cdot 10^6$
$n \log n$	140	4893	$2{,}0 \cdot 10^5$
n^2	31	244	1897
n^3	10	39	153
2^n	9	15	21

Следует, однако, иметь в виду, что трудоемкость, выражаемая большей степенью полинома, может иметь меньший множитель c из неравенства $f(n) \le cg(n)$. Точно так же сложность алгоритма, которая носит экспоненциальный характер, может иметь множитель, меньший, чем у полиномиальной сложности. При разработке компьютерных программ для решения практических задач важно знать, при каких значениях параметра n время выполнения экспоненциального алгоритма оказывается меньше, чем время выполнения полиномиального алгоритма, решающего ту же задачу.

Для очень многих практических комбинаторных задач существуют алгоритмы только экспоненциальной трудоемкости. Может показаться, что с совершенствованием вычислительной техники и ростом быстродействия вычислительных машин проблема трудоемкости ослабевает. Однако данные, приведенные в табл. 4.2 [2], говорят, что это не так. Пусть следующее поколение вычислительных машин будет иметь быстродействие, в десять раз большее, чем у современных вычислительных машин. В табл. 4.2 показано, как благодаря увеличению быстродействия возрастут размеры задач, которые могут быть решены за некоторую фиксированную единицу времени. Задачи достаточно большого размера, решаемые только алгоритмами экспоненциальной трудоемкости, вообще не могут быть решены за практически приемлемое время, даже если надеяться на существенное увеличение быстродействия вычислительных машин в будущем.

Таблица 4.2

Связь размера задачи, решаемой за заданное время, с быстродействием вычислительной машины

Временна́я сложность	Максимальный размер задачи	
	до ускорения	после ускорения
n	s_1	$10\ s_1$
$n \log n$	s_2	$\approx 10\ s_2$
n^2	s_3	$3{,}16\ s_3$
n^3	s_4	$2{,}15\ s_4$
2^n	s_5	$s_5 + 3{,}3$

Иногда удается найти способы сокращения перебора благодаря некоторым особенностям конкретных исходных данных.

Другой путь выхода из такого положения – использование простых методов, дающих решения, достаточно близкие к оптимальным. Для практических задач не всегда требуется получать точное решение. Часто достаточно иметь решение, близкое к оптимальному. Если при таком методе удается оценить близость получаемого решения к оптимальному, метод называется *приближенным*. В противном случае его называют *эвристическим*, где используется локальная оптимизация. Пример эвристического метода рассмотрен ранее при решении задачи раскраски графа.

4.4. Методы комбинаторного поиска

Один из наиболее общих и плодотворных подходов к решению комбинаторных задач заключается в применении *дерева поиска*. В дереве выделяется вершина, которая называется *корнем дерева* и которая ставится в соответствие исходной ситуации в процессе решения задачи. Остальные вершины сопоставляются с ситуациями, которые можно достичь в данном процессе. Выделение корня придает дереву ориентацию, при которой все пути ведут из корня в остальные вершины. Дуги дерева соответствуют некоторым простым операциям, представляющим шаги процесса решения, и связывают вершины, соответствующие ситуациям, одна из которых преобразуется в другую в результате выполнения шага. Для ситуации характерно разнообразие вариантов выбора очередного шага, представленных дугами, исходящими из соответствующей вершины. Некоторые ситуации соответствуют решениям.

Дерево поиска не задается априори, а строится в процессе поиска: когда возникает некоторая ситуация, тогда и определяются возможные направления процесса, которые представляются исходящими из вершины дугами. Естественным является стремление сокращать число этих дуг, чтобы быстрее найти решение. Способы этого сокращения строятся с учетом особенностей конкретных задач.

Довольно общим средством повышения эффективности процесса решения задачи является *редуцирование*, т. е. упрощение текущей ситуации, сокращающее объем вычислений, проводимых при анализе множества вытекающих из нее вариантов. Способы редуцирования определяются особенностями конкретной задачи и исходных данных.

Процедуру комбинаторного поиска удобно проследить на примере решения задачи о кратчайшем покрытии, которую рассмотрим в подразд. 4.5.

4.5. Задача о кратчайшем покрытии и методы ее решения

Многие комбинаторные оптимизационные задачи сводятся к *задаче о кратчайшем покрытии*, которая ставится следующим образом. Пусть даны некоторое множество $A = \{a_1, a_2, \ldots, a_n\}$ и совокупность его подмножеств $B_1, B_2, \ldots, B_m$, т. е. $B_i \subseteq A$, $i = 1, 2, \ldots, m$, причем $B_1 \cup B_2 \cup \ldots \cup B_m = A$. Требуется среди данных подмножеств выделить такую совокупность $B_{i_1}, B_{i_2}, \ldots, B_{i_k}$ с минимальным k, чтобы каждый элемент из A попал хотя бы в одно из B_{i_j} ($j = 1, 2, \ldots, k$), т. е. $B_{i_1} \cup B_{i_2} \cup \ldots \cup B_{i_k} = A$.

Одной из интерпретаций этой задачи является задача о переводчиках. Из некоторого коллектива переводчиков, число которых m и каждый из которых владеет несколькими определенными языками, требуется скомплектовать минимальную по числу членов группу, такую, чтобы она смогла обеспечить перевод с любого из заданного множества языков, число которых n. Здесь A – множество языков, перевод с которых требуется обеспечить, а B_i – множество языков, которыми владеет i-й переводчик.

Удобно рассматривать матричную формулировку данной задачи, при которой совокупность $B_1, B_2, \ldots, B_m$ задается в виде булевой матрицы, строки которой соответствуют подмножествам из данной совокупности, а столбцы – элементам множества A. Элемент i-й строки и j-го столбца имеет значение 1, если и только если $a_j \in B_i$. В этом случае говорят, что i-я строка *покрывает* j-й столбец. Требуется найти такое множество строк данной матрицы, чтобы каждый ее столбец имел единицу хотя бы в одной строке из этого множества, и при этом мощность выбранного множества должна быть минимальной.

Существуют *эвристические* методы решения данной задачи. Например, ее можно решать с помощью «жадного» алгоритма, представляющего собой многошаговый процесс, где на каждом шаге выбирается и включается в покрытие строка заданной матрицы, покрывающая наибольшее число из еще не покрытых столбцов. Этот процесс заканчивается, когда все столбцы матрицы оказываются покрытыми. Применение «жадного» алгоритма иногда дает точное решение, но гарантии этому нет. Например, если задана матрица

$$\begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} & & & & & & \begin{matrix} B_1 \\ B_2 \\ B_3 \end{matrix} \end{matrix},$$

первой для включения в формируемое решение «жадный» алгоритм выберет строку B_1, после чего для покрытия оставшихся столбцов должны быть включены в решение обе строки B_2 и B_3. Кратчайшее же покрытие данной матрицы составляют только две строки – B_2 и B_3.

Более близкое к кратчайшему покрытие получается чаще всего с помощью «*минимаксного*» алгоритма. Он представляет собой многошаговый процесс, на каждом шаге которого выбирается столбец с минимальным числом единиц и из покрывающих его строк для включения в решение выбирается та, которая покрывает максимальное число непокрытых столбцов. Пусть, например, задана матрица

$$\begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \end{matrix} .$$

Одним из столбцов с минимальным числом единиц является столбец a_6. Из покрывающих его строк максимальное число столбцов покрывает строка B_6. Включим эту строку в решение и удалим ее и столбцы, которые она покрывает, в результате чего получим

$$\begin{array}{c} \begin{matrix} a_2 & a_4 & a_5 & a_7 & a_8 & a_{10} \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_7 \\ B_8 \\ B_9 \end{matrix} .$$

Из оставшихся столбцов минимальное число единиц имеет столбец a_{10}. Покрывающие его строки B_4 и B_9 имеют одинаковое число единиц, т. е. одинаковое число покрываемых ими, но еще не покрытых столбцов. Включаем в решение первую по порядку строку B_4 и получаем матрицу

$$\begin{array}{c} \begin{matrix} a_2 & a_5 & a_7 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} B_1 \\ B_2 \\ B_3 \\ B_5 \\ B_7 \\ B_8 \\ B_9 \end{matrix}.$$

В полученной матрице столбцом с минимальным числом единиц является столбец a_2, а из покрывающих его строк строка B_7 имеет максимальное число единиц. Включение этой строки в решение завершает процесс, в результате которого полученным покрытием является $\{B_4, B_6, B_7\}$. Как будет показано далее, это решение является точным.

Точный метод нахождения кратчайшего покрытия представляет собой обход дерева поиска. Текущая ситуация, соответствующая некоторой вершине дерева поиска, представляется переменной матрицей $\boldsymbol{X}$, которая показывает, какие столбцы еще не покрыты и какие строки можно использовать для их покрытия. В этой ситуации выбирается первый из столбцов с минимальным числом единиц – так минимизируется число вариантов продолжения поиска. Очередной шаг процесса состоит в выборе покрывающей строки для этого столбца и пробном включении ее в получаемое решение. Таким образом, вершины дерева поиска соответствуют некоторым столбцам исходной матрицы, а дуги – выбираемым для их покрытия строкам.

Начальное значение матрицы $\boldsymbol{X}$ совпадает с исходной матрицей. Последующие значения получаются удалением строк, включаемых в решение, и столбцов, покрытых этими строками. Кроме того, выполняются следующие *правила редукции* [5].

1. Если столбец k имеет единицы везде, где имеет единицы столбец l, то столбец k можно удалить. Любая строка, покрывающая столбец l, покрывает также столбец k. Поэтому при поиске покрытия столбец k можно не рассматривать. Достаточно, чтобы в покрытие была включена какая-либо из строк, покрывающих столбец l.

2. Если строка i имеет единицы везде, где имеет единицы строка j, то строку j можно удалить. Действительно, пусть в некотором кратчайшем покрытии имеется строка j. Очевидно, данное покрытие останется кратчайшим, если в нем строку j заменить строкой i.

Продемонстрируем описанный процесс на матрице из предыдущего примера:

$$
\begin{array}{cccccccccc}
a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} \\
\left[\begin{array}{cccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}\right] &
\begin{array}{c}
B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9
\end{array}
\end{array}.
$$

На первом шаге выбираем столбец a_6, содержащий две единицы. Среди покрывающих ее строк выбираем такую, которая покрывает наибольшее число столбцов. Такой строкой является строка B_6. Удалив эту строку и покрываемые ею столбцы, получим следующее значение матрицы $\boldsymbol{X}$:

$$
\begin{array}{cccccc}
a_2 & a_4 & a_5 & a_7 & a_8 & a_{10} \\
\left[\begin{array}{cccccc}
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1
\end{array}\right] &
\begin{array}{c}
B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_7 \\ B_8 \\ B_9
\end{array}
\end{array}.
$$

После удаления строк B_1, B_2 и B_8 согласно второму правилу редукции матрица $\boldsymbol{X}$ будет иметь следующий вид:

$$
\begin{array}{cccccc}
a_2 & a_4 & a_5 & a_7 & a_8 & a_{10} \\
\left[\begin{array}{cccccc}
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1
\end{array}\right] &
\begin{array}{c}
B_3 \\ B_4 \\ B_5 \\ B_7 \\ B_9
\end{array}
\end{array}.
$$

Одним из столбцов, обладающих минимальным числом единиц, является столбец a_2. Обе покрывающие его строки B_3 и B_7 содержат по три единицы. Вы-

бираем первую по порядку строку B_3 и включаем ее в формируемое покрытие. Теперь имеем множество $\{B_3, B_6\}$. Этот шаг приводит к матрице

$$\begin{matrix} a_4 & a_7 & a_{10} \\ \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \begin{matrix} B_4 \\ B_5 \\ B_7 \\ B_9 \end{matrix} \end{matrix}.$$

После удаления строки B_7 по второму правилу редукции получим матрицу, каждая строка и каждый столбец которой содержат ровно две единицы. Выбрав строку B_4, покрывающую столбец a_4, и проведя аналогичные преобразования, получим матрицу с одним столбцом a_7 и двумя строками B_5 и B_9, любая из которых покрывает оставшийся столбец. Таким образом, получено покрытие $\{B_3, B_4, B_5, B_6\}$, но пройдена пока только одна ветвь дерева поиска, и до совершения полного обхода дерева неизвестно, является ли это покрытие кратчайшим.

Возвращаемся к ситуации, когда очередным столбцом для покрытия взят a_2. Теперь вместо строки B_3 возьмем для покрытия столбца a_2 строку B_7. Действуя дальше аналогичным образом, получаем очередное покрытие $\{B_4, B_6, B_7\}$, которое вытесняет предыдущее, так как оно оказалось лучше, однако и его пока нельзя назвать кратчайшим.

Возвратившись к начальной вершине дерева поиска и следуя по дуге, соответствующей строке B_2, убеждаемся, что длина покрытия не может быть меньше трех. На этом поиск можно закончить и выдать в качестве решения множество $\{B_4, B_6, B_7\}$. На дереве поиска, обход которого совершался в процессе решения данного примера (рис. 4.1), вершинам приписаны столбцы, а дугам – строки.

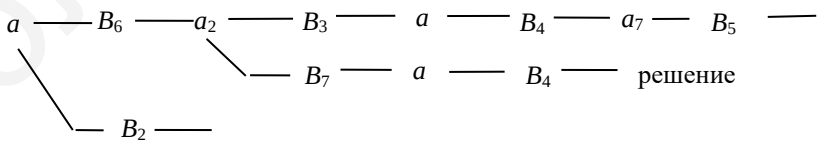


Рис. 4.1. Дерево поиска кратчайшего покрытия

4.6. Задача о вырожденности троичной матрицы

Троичной матрицей является матрица, элементы которой принимают значения из множества $\{0, 1, –\}$. Аналогично определяется троичный вектор. Троичные векторы $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ и $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ одинаковой размерности *ортогональны* по i-й компоненте, если $a_i = 0$ и $b_i = 1$ (или наоборот). Два троич-

ных вектора просто ортогональны, если они ортогональны хотя бы по одной компоненте.

Троичная матрица $\boldsymbol{U}$ является *вырожденной*, если не существует троичного вектора, ортогонального каждой ее строке. Поставим задачу следующим образом. Для заданной троичной матрицы $\boldsymbol{U}$ требуется найти троичный вектор $\boldsymbol{v}$, ортогональный каждой ее строке, или убедиться в том, что такого вектора не существует.

Троичную матрицу можно рассматривать как сжатую форму булевой матрицы, если считать, что всякий троичный вектор представляет множество булевых векторов, получаемых заменой значений «–» на всевозможные комбинации нулей и единиц. Троичный вектор, имеющий k компонент со значением «–», представляет множество 2^k булевых векторов. Будем говорить, что любой из этих булевых векторов *покрывается* данным троичным вектором. Например, матрица

$$\begin{bmatrix} 1 & - & - \\ - & 1 & 1 \end{bmatrix}$$

является сжатой формой следующей булевой матрицы (заметим, что, если специально не оговорено, рассматриваются матрицы, не имеющие одинаковых строк):

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Троичный вектор (0, 0, –) ортогонален обеим строкам приведенной троичной матрицы. Он представляет множество из двух булевых векторов, (0, 0, 0) и (0, 0, 1), ни один из которых не является строкой соответствующей булевой матрицы. Очевидно, что если некоторая троичная матрица с n столбцами является вырожденной, то для любого n-компонентного булева вектора в данной матрице имеется покрывающая его строка. Если же существует булев вектор, не покрываемый ни одной строкой троичной матрицы, число столбцов которой равно размерности данного вектора, то данная матрица не вырождена. Следовательно, решить задачу о вырожденности троичной матрицы можно простым перебором всех 2^n различных булевых векторов, сопровождаемым поиском для каждого вектора покрывающей его строки.

Однако более эффективным является *редукционный метод* [5]. Данный метод опирается на комбинаторный поиск. Текущая ситуация характеризуется двумя переменными величинами: троичным вектором $\boldsymbol{w}$, число компонент ко-

торого фиксировано и равно числу столбцов в заданной матрице ***U***, и троичной матрицей ***T***, значениями которой будут служить некоторые *миноры* матрицы ***U***. Под минором матрицы понимается ее часть, образованная заданным подмножеством строк и заданным подмножеством столбцов. Перебор значений вектора ***w*** должен привести к искомому вектору ***v***, если он существует.

Положим, что в текущей ситуации уже определены значения некоторых компонент вектора ***w***, т. е. им приписано значение 0 или 1, и отыскиваются значения остальных компонент, такие, чтобы вектор ***w*** стал ортогональным каждой из строк матрицы ***T*** (ее текущего значения), столбцы которой ставятся в соответствие этим компонентам. В начальной ситуации матрица ***T*** совпадает с матрицей ***U***, а вектор ***w*** полностью не определен, т. е. все его компоненты имеют значение «–».

Очередной шаг заключается в приписывании значения 0 или 1 некоторой компоненте вектора ***w*** или в упрощении матрицы ***T*** путем удаления некоторых строк и столбцов с сохранением обозначений остающихся. Каждый раз в матрице ***T*** остаются только те строки, которые еще не являются ортогональными вектору ***w***, и столбцы, которые соответствуют некоторым компонентам вектора ***w***. Это те компоненты, которые можно использовать для обеспечения ортогональности вектора ***w*** данным строкам. Перед выполнением очередного шага, если позволяют условия, текущая ситуация упрощается по следующим *правилам редукции*.

Правило 1. Из матрицы ***T*** удаляются столбцы, не содержащие ни значений 0, ни значений 1. (Какое бы значение ни приписывалось компонентам вектора ***w***, соответствующим таким столбцам, ни одна из строк матрицы ***T*** не будет ортогональной вектору ***w*** по этим компонентам.)

Правило 2. Из матрицы ***T*** удаляются строки, ортогональные вектору ***w***, а затем столбцы, которым соответствуют компоненты вектора ***w*** со значением 0 или 1.

Правило 3. Если в матрице ***T*** имеется строка, где лишь одна компонента обладает значением, отличным от «–», то соответствующей компоненте вектора ***w*** приписывается инверсное значение. (Только таким образом можно обеспечить в текущей ситуации ортогональность данной строки вектору ***w***.)

Правило 4. Если в матрице ***T*** существует столбец, не содержащий значения 0 (или 1), то это значение приписывается соответствующей компоненте вектора ***w***. (Если в столбце присутствуют как нули, так и единицы, то, приписывая соответствующей компоненте какое-то из этих значений, мы делаем одни строки ортогональными вектору ***w*** и теряем возможность использовать данную компоненту для обеспечения ортогональности других строк. Такой потери не происходит, когда выполняется данное правило при указанном условии. Текущая ситуация при этом упрощается.)

Когда редуцирование становится невозможным, производится расщепление текущей ситуации.

Правило расщепления предписывает перебор значений 0 и 1 некоторой компоненты вектора ***w***. При этом рекомендуется выбирать такую компоненту,

которая соответствует максимально определенному столбцу матрицы $\boldsymbol{T}$, т. е. столбцу, имеющему минимальное число значений «–».

Правило нахождения решения. Если непосредственно после удаления некоторой строки из матрицы $\boldsymbol{T}$ по правилу 2 матрица становится пустой, текущее значение вектора $\boldsymbol{w}$ представляет искомое решение $\boldsymbol{v}$.

Правило возврата. Если матрица $\boldsymbol{T}$ становится пустой непосредственно после удаления некоторого столбца или если она содержит строку без значений 0 и 1, то на данной ветви дерева поиска вектор $\boldsymbol{v}$ найти невозможно и следует продолжить обход дерева поиска, возвратившись к последней из точек ветвления с незавершенным перебором.

Правило прекращения поиска. Если при полном обходе дерева поиска вектор $\boldsymbol{v}$ найти не удалось, то это свидетельствует о вырожденности матрицы $\boldsymbol{U}$.

Рассмотрим для примера следующую троичную матрицу, столбцы которой для удобства обозначим теми же буквами *a, b, c, d, e, f*, что и соответствующие им компоненты вектора $\boldsymbol{w} = (a, b, c, d, e, f)$:

$$\begin{array}{cccccc} a & b & c & d & e & f \\ \left[\begin{array}{cccccc} 1 & – & – & – & – & 0 \\ 1 & – & 1 & – & 1 & – \\ 1 & 0 & – & – & 1 & 1 \\ 1 & – & – & – & 0 & 1 \\ 1 & 1 & 0 & – & – & 1 \\ 0 & 1 & – & – & – & 1 \\ 0 & – & – & – & 0 & 1 \\ 0 & 0 & – & – & 1 & – \\ 0 & – & – & 0 & 1 & 0 \\ 0 & 1 & – & 1 & 1 & 0 \\ 0 & 1 & – & – & 0 & 0 \\ 0 & – & 1 & – & 0 & 0 \\ 0 & 0 & 0 & – & 0 & 0 \end{array}\right. & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \end{array} \end{array}.$$

Начальная ситуация характеризуется матрицей $\boldsymbol{T}$, совпадающей с исходной матрицей и вектором $\boldsymbol{w} = (------)$. Непосредственное сокращение матрицы $\boldsymbol{T}$ невозможно, поскольку не выполняются условия применения правил редукции. Поэтому воспользуемся правилом расщепления и образуем точку ветвления процесса поиска вектора $\boldsymbol{v}$, соответствующую выбору значения компоненты a вектора $\boldsymbol{w}$. Положим $a = 1$. Тогда, согласно правилам 1 и 2, определение остальных значений компонент вектора $\boldsymbol{w}$ сведется к поиску вектора, ортогонального матрице

$$\boldsymbol{T} = \begin{matrix} b & c & e & f & \\ \left[\begin{matrix} - & - & - & 0 \\ - & 1 & 1 & - \\ 0 & - & 1 & 1 \\ - & - & 0 & 1 \\ 1 & 0 & - & 1 \end{matrix}\right. & & & & \left.\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix}\right] \end{matrix}.$$

Обратив внимание на строку 1 и применяя правило 3, припишем компоненте f вектора $\boldsymbol{w}$ значение 1, после чего матрица сокращается по правилу 2 до следующего вида:

$$\boldsymbol{T} = \begin{matrix} b & c & e & \\ \left[\begin{matrix} - & 1 & 1 \\ 0 & - & 1 \\ - & - & 0 \\ 1 & 0 & - \end{matrix}\right. & & & \left.\begin{matrix} 2 \\ 3 \\ 4 \\ 5 \end{matrix}\right] \end{matrix}.$$

Далее опять применяется правило 3, компонента e получает значение 1, т. е. теперь $\boldsymbol{w}$ = (1 – – – 1 1), и матрица $\boldsymbol{T}$ сокращается до

$$\boldsymbol{T} = \begin{matrix} b & c & \\ \left[\begin{matrix} - & 1 \\ 0 & - \\ 1 & 0 \end{matrix}\right. & & \left.\begin{matrix} 2 \\ 3 \\ 5 \end{matrix}\right] \end{matrix}.$$

Согласно правилу 3 полагаем $c = 0$ и $b = 1$, т. е. $\boldsymbol{w}$ = (1 1 0 – 1 1). Далее срабатывает правило возврата, поскольку матрица $\boldsymbol{T}$ становится пустой после удаления столбцов. Это означает, что, направляясь в дереве поиска по ветви, соответствующей $a = 1$, не получаем искомого вектора $\boldsymbol{v}$. При присвоении всех возможных значений оставшимся компонентам строка 5 остается не ортогональной вектору $\boldsymbol{w}$.

Возвратившись к точке ветвления, полагаем теперь $a = 0$. Последующее редуцирование приводит к следующему значению переменной матрицы $\boldsymbol{T}$:

$$\boldsymbol{T} = \begin{array}{c} \begin{matrix} b & c & d & e & f \end{matrix} \\ \begin{bmatrix} 1 & - & - & - & 1 \\ - & - & - & 0 & 1 \\ 0 & - & - & 1 & - \\ - & - & 0 & 1 & 0 \\ 1 & - & 1 & 1 & 0 \\ 1 & - & - & 0 & 0 \\ - & 1 & - & 0 & 0 \\ 0 & 0 & - & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \end{matrix}.$$

Поскольку дальнейшее редуцирование невозможно, применяем правило расщепления. Выберем компоненту e и положим $e = 1$. Матрица $\boldsymbol{T}$ сокращается до

$$\boldsymbol{T} = \begin{array}{c} \begin{matrix} b & d & f \end{matrix} \\ \begin{bmatrix} 1 & - & 1 \\ 0 & - & - \\ - & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} 6 \\ 8 \\ 9 \\ 10 \end{matrix}.$$

Далее по правилу 3 компонента b получает значение 1 и матрица $\boldsymbol{T}$ сокращается до

$$\boldsymbol{T} = \begin{array}{c} \begin{matrix} d & f \end{matrix} \\ \begin{bmatrix} - & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \end{array} \begin{matrix} 6 \\ 9 \\ 10 \end{matrix}.$$

Затем следует выбор значения 0 для компоненты f и получение остатка

$$\boldsymbol{T} = \begin{array}{c} \begin{matrix} d \end{matrix} \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{array} \begin{matrix} 9 \\ 10 \end{matrix},$$

который оказывается вырожденной матрицей: компонента d должна получить одновременно значения 0 и 1, что невозможно. Опять срабатывает правило возврата.

Рассмотрим теперь оставшийся вариант, положив $e = 0$. Здесь необходимо найти вектор, ортогональный матрице

$$\boldsymbol{T} = \begin{matrix} b & c & f & \\ \begin{bmatrix} 1 & - & 1 \\ - & - & 1 \\ 1 & - & 0 \\ - & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} & & & \begin{matrix} 6 \\ 7 \\ 11 \\ 12 \\ 13 \end{matrix} \end{matrix}.$$

В соответствии с правилом 3 последовательно выбираются значения $f = 0$, $b = 0$, $c = 0$, после чего становится очевидным, что нельзя сделать вектор $\boldsymbol{w}$ ортогональным строке 13.

Перебор значений вектора $\boldsymbol{w}$ завершен и установлено, что вектора $\boldsymbol{v}$, ортогонального всем строкам матрицы $\boldsymbol{U}$, не существует, т. е. матрица $\boldsymbol{U}$ оказывается вырожденной.

Дерево поиска, соответствующее описанному процессу, изображено на рис. 4.2, где вершины обозначены символами компонент вектора $\boldsymbol{w}$, а дуги – их значениями.

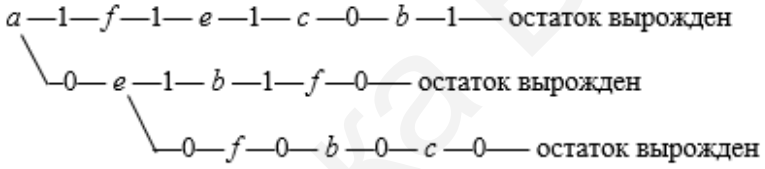


Рис. 4.2. Дерево поиска ортогонального вектора

5. БУЛЕВЫ ФУНКЦИИ

5.1. Способы задания булевой функции

Пусть $x_1, x_2, ..., x_n$ – некоторые *булевы переменные*, т. е. переменные, принимающие значение из множества $\{0, 1\}$. Упорядоченную совокупность булевых переменных $(x_1, x_2, ..., x_n)$ можно рассматривать как n-компонентный *булев вектор* $\boldsymbol{x}$. Число компонент вектора определяет его длину, или *размерность*. При фиксации значений всех переменных получается *набор значений* переменных $(x_1, x_2, ..., x_n)$, задаваемый булевым вектором длиной n, состоящим из констант 0 и 1. Очевидно, 2^n – число всех таких векторов. Они образуют *булево пространство*. *Булевой функцией* называется функция $f : \{0, 1\}^n \to \{0, 1\}$. Областью определения булевой функции является булево пространство $M = \{0, 1\}^n$, областью значений – множество $\{0, 1\}$.

Задание булевой функции f на булевом пространстве M делит его на две части: M_f^1 – область, где функция принимает значение 1, и M_f^0 – область, где функция принимает значение 0. Множество M_f^1 называется *характеристическим множеством* функции f.

Универсальным способом задания для любой дискретной функции является табличный способ. Таблица, представляющая функцию и называемая *таблицей истинности*, имеет два столбца. В левом столбце перечислены все наборы значений аргументов, в правом столбце – соответствующие им значения функции. Примером задания булевой функции от трех аргументов является табл. 5.1.

Таблица 5.1

Задание функции $f(x_1, x_2, x_3)$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Для задания булевой функции можно ограничиться перечислением элементов ее характеристического множества M_f^1. Множество M_f^1 задается булевой матрицей, строки которой представляют элементы этого множества. Следующая матрица, задающая приведенную выше функцию, является *матричным способом* представления булевой функции:

$$\begin{matrix} x_1 & x_2 & x_3 \end{matrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Компактность представления характеристического множества M_f^1 можно повысить, используя *троичные векторы*, компоненты которых могут принимать в качестве своих значений кроме символов 0 и 1 также символ «–». В этом случае значения булевой функции будут задаваться не на отдельных элементах, а на интервалах пространства переменных $x_1, x_2, ..., x_n$. Чтобы определить понятие интервала булева пространства, введем отношение «меньше или равно» на множестве булевых векторов.

Булевы векторы $\boldsymbol{a} = (a_1, a_2, \dots, a_n)$ и $\boldsymbol{b} = (b_1, b_2, \dots, b_n)$ находятся в отношении «меньше или равно» ($\boldsymbol{a} \le \boldsymbol{b}$), и тогда говорят, что $\boldsymbol{a}$ меньше $\boldsymbol{b}$, если $a_i \le b_i$ для любого $i = 1, 2, \dots, n$, в противном случае они *несравнимы*. При этом считается, что $0 \le 1$. Тогда *интервалом булева пространства* называется множество

векторов, среди которых есть минимальный и максимальный векторы, а также все векторы, меньшие максимального и большие минимального. Интервал представляется троичным вектором, который задает множество всех булевых векторов, получаемых заменой символа «–» на 1 или 0.

Троичная матрица *эквивалентна* булевой матрице, получаемой из нее заменой каждой троичной строки на представляемую ею совокупность булевых строк с последующим устранением повторяющихся строк. Приведенная выше булева матрица оказывается эквивалентной троичной матрице

$$\begin{matrix} x_1 & x_2 & x_3 \\ \left[\begin{matrix} - & - & 1 \\ 0 & 1 & - \end{matrix}\right. & & \left.\begin{matrix} \\ \\ \end{matrix}\right] \end{matrix},$$

которая представляет ту же булеву функцию $f(x_1, x_2, x_3)$. Такой способ задания булевой функции называют еще *интервальным*. Представление булевой функции троичной матрицей не однозначно, т. е. для одной и той же булевой матрицы существует в общем случае не одна эквивалентная ей троичная матрица.

Две троичные матрицы *эквивалентны*, если они эквивалентны одной и той же булевой матрице, т. е. если они представляют одну и ту же булеву функцию.

Векторное задание булевой функции представляет собой булев вектор, компоненты которого соответствуют наборам значений аргументов. Эти наборы упорядочиваются обычно согласно порядку чисел, двоичные коды которых они представляют. Рассмотренная выше булева функция $f(x_1, x_2, x_3)$ представляется вектором (0 1 1 1 0 1 0 1), показывающим, что функция принимает значение 0 на наборах (0 0 0), (1 0 0), (1 1 0) и значение 1 на наборах (0 0 1), (0 1 0), (0 1 1), (1 0 1), (1 1 1). Заметим, что этот вектор совпадает с правым столбцом табл. 5.1.

Если значения булевой функции определены для всех 2^n наборов значений вектора $\boldsymbol{x}$, она называется *полностью определенной*, в противном случае – *не полностью определенной*, или *частичной*. Задание не полностью определенной булевой функции f разбивает булево пространство на три множества: кроме M_f^1 и M_f^0 в нем присутствует множество M_f^-, где значения функции f не определены. Для задания частичной булевой функции необходимо задать не менее двух множеств. Обычно это M_f^1 и M_f^0.

В подразд. 5.2 будет рассмотрен *алгебраический способ* задания булевой функции.

5.2. Элементарные булевы функции и алгебраические формы

Рассматривая векторную форму задания булевой функции, легко определить число булевых функций от n переменных – это число всех 2^n-компонентных булевых векторов, т. е. 2^{2^n}. Однако это число учитывает так-

же функции и от меньшего числа аргументов. Любую функцию от n аргументов можно считать функцией от большего числа аргументов. Для этого вводится понятие *существенной зависимости* и *несущественной зависимости*. Функция $f(x_1, x_2, ..., x_n)$ существенно зависит от аргумента x_i, если

$$f(x_1, x_2, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \neq f(x_1, x_2, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n).$$

Переменная x_i в этом случае называется *существенным аргументом*. В противном случае она является *несущественным* или *фиктивным аргументом*.

Элементарными булевыми функциями являются функции от одной и двух переменных. Число функций от одной переменной равно $2^{2^1} = 4$. Эти функции представлены в табл. 5.2.

Таблица 5.2

Булевы функции от одного аргумента

x	0	1
$f_0 = 0$	0	0
$f_1 = x$	0	1
$f_2 = \bar{x}$	1	0
$f_3 = 1$	1	1

Две из них, f_0 и f_3, – константы 0 и 1, переменная x для них – несущественный аргумент. Функция f_1 также является тривиальной, любое ее значение совпадает со значением аргумента: $f_1(x) = x$. Нетривиальной функцией является функция $f_2(x) = \bar{x}$, называемая *отрицанием*, или *инверсией*. Ее значение всегда противоположно значению аргумента x. Из табл. 5.2 видно, что $\bar{0} = 1$ и $\bar{1} = 0$.

В табл. 5.3 приведены все булевы функции $f_i(x_1, x_2)$ от двух аргументов. В левом столбце показаны их выражения в терминах нескольких функций, принятых за основные. Таблица содержит имена функций и их обозначения, а также значения, принимаемые данными функциями на каждом из четырех наборов значений аргументов x_1 и x_2 (приведенных в верхней части таблицы), т. е. каждая из функций задана в векторном виде.

Элементарные функции имеют особое значение в теории булевых функций, поскольку каждая из них может быть выражена одно- или двухместной алгебраической операцией. Все операции, представленные в табл. 5.3, составляют *алгебру логики*. Любая булева функция от любого числа аргументов может быть представлена формулой алгебры логики. Формулу, содержащую более чем одну операцию, можно рассматривать как *суперпозицию* элементарных функций. Под суперпозицией функций понимается использование одних функций в качестве аргументов других функций. Для булевых функций это является возможным благодаря совпадению области значений функций с областями значений их аргументов. Таким образом, *алгебраическое задание* булевой функции представляет собой формулу, по которой вычисляется значение этой функции.

Таблица 5.3

Булевы функции от двух аргументов

x_1	0	0	1	1
x_2	0	1	0	1
$f_0 = 0$ – константа 0	0	0	0	0
$f_1 = x_1 \wedge x_2$ – конъюнкция	0	0	0	1
f_2 – отрицание импликации	0	0	1	0
$f_3 = x_1$	0	0	1	1
f_4 – отрицание обратной импликации	0	1	0	0
$f_5 = x_2$	0	1	0	1
$f_6 = x_1 \oplus x_2$ – сложение по модулю 2	0	1	1	0
$f_7 = x_1 \vee x_2$ – дизъюнкция	0	1	1	1
$f_8 = x_1 \uparrow x_2$ – стрелка Пирса	1	0	0	0
$f_9 = x_1 \sim x_2$ – эквиваленция	1	0	0	1
$f_{10} = \bar{x}_2$	1	0	1	0
$f_{11} = x_2 \to x_1$ – обратная импликация	1	0	1	1
$f_{12} = \bar{x}_1$	1	1	0	0
$f_{13} = x_1 \to x_2$ – импликация	1	1	0	1
$f_{14} = x_1 \mid x_2$ – штрих Шеффера	1	1	1	0
$f_{15} = 1$ – константа 1	1	1	1	1

Понятие *формулы* определим индуктивно следующим образом:

1) каждый символ переменной есть формула;

2) если A и B – формулы, то формулами являются $\bar{A}$ и $(A * B)$, где $*$ – любая операция алгебры логики;

3) других формул нет.

Для установления порядка выполнения операций в формулах используются скобки. При отсутствии скобок порядок устанавливается согласно приоритетам операций. Первым приоритетом обладает операция отрицания, затем выполняется операция $\wedge$. Третьим приоритетом обладают операции $\vee$ и $\oplus$, четвертым приоритетом – операция $\to$, пятым – операция $\sim$, что эквивалентно $\leftrightarrow$. Для упрощения написания формул иногда символ конъюнкции опускается. Процесс вычисления значений функции $f(x_1, x_2, x_3) = \overline{(x_1 \vee x_2)x_3} \to \bar{x}_1 \vee x_2 x_3$ по ее формуле покажем с помощью табл. 5.4, где последовательность столбцов соответствует порядку выполнения операций.

Таблица 5.4

Вычисление по формуле

x_1	x_2	x_3	$x_1 \vee x_2$	$(x_1 \vee x_2)\, x_3$	$\overline{(x_1 \vee x_2)x_3}$	$\bar{x}_1$	$x_2 x_3$	$\bar{x}_1 \vee x_2 x_3$	$f(x_1, x_2, x_3)$
0	0	0	0	0	1	1	0	1	1
0	0	1	0	0	1	1	0	1	1
0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	1	1	1
1	0	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	0	1
1	1	0	1	0	1	0	0	0	0
1	1	1	1	1	0	0	1	1	1

Две разные формулы могут представлять одну и ту же функцию. Такие формулы называются *равносильными*. Если A и B – равносильные формулы, то $A = B$. В этом случае, если A является частью другой формулы, то вместо нее можно подставить B, в результате чего получится формула, равносильная исходной.

Алгебра, содержащая только три операции $\bar{}$, $\wedge$ и $\vee$, называется *булевой*, так же как и формула, представляющая некоторую композицию этих операций. Легко проверить по табл. 5.4 следующие основные законы булевой алгебры.

Коммутативность:
$x \vee y = y \vee x$; $\quad x\,y = y\,x$.

Ассоциативность:
$x \vee (y \vee z) = (x \vee y) \vee z$; $\quad x\,(y\,z) = (x\,y)\,z$.

Дистрибутивность:
$x\,(y \vee z) = x\,y \vee x\,z$; $\quad x \vee y\,z = (x \vee y)\,(x \vee z)$.

Идемпотентность:
$x \vee x = x$; $\quad x\,x = x$.

Законы де Моргана:
$\overline{x \vee y} = \bar{x}\,\bar{y}$; $\quad \overline{xy} = \bar{x} \vee \bar{y}$.

Законы операций с константами:
$x \vee 0 = x$; $\quad x\,1 = x$;
$x\,0 = 0$; $\quad x \vee 1 = 1$;
$x \vee \bar{x} = 1$; $\quad x\,\bar{x} = 0$.

Закон двойного отрицания:
$\bar{\bar{x}} = x$.

Здесь действует принцип двойственности, т. е. для каждой пары формул, представляющих тот или иной закон, справедливо следующее утверждение: одна из формул получается из другой взаимной заменой всех символов конъюнкции на символы дизъюнкции и всех единиц – на нули.

На основании этих формул выводятся следующие соотношения.

Закон поглощения:
$x \vee x\,y = x$; $\quad x\,(x \vee y) = x$.

Действительно,

$x \vee x\,y = x\,1 \vee x\,y = x\,(1 \vee y) = x1 = x;$
$x\,(x \vee y) = x\,x \vee x\,y = x \vee x\,y = x.$

Закон простого склеивания:
$x\,y \vee x\,\bar{y} = x;$ $\qquad\qquad (x \vee y)\,(x \vee \bar{y}) = x.$

Левая формула выводится с помощью закона дистрибутивности конъюнкции относительно дизъюнкции: $x\,y \vee x\,\bar{y} = x(y \vee \bar{y}) = x$. Для вывода правой формулы достаточно раскрыть скобки и применить законы идемпотентности и поглощения.

Закон обобщенного склеивания:
$x\,y \vee \bar{x}\,z = x\,y \vee \bar{x}\,z \vee y\,z.$

Эту формулу можно вывести следующим образом:

$$x\,y \vee \bar{x}\,z \vee y\,z = x\,y \vee \bar{x}\,z \vee y\,z\,(x \vee \bar{x}) = x\,y\,(1 \vee z) \vee \bar{x}\,z\,(1 \vee y) = x\,y \vee \bar{x}\,z,$$

а если подставить $y = 1$, то получим

$$x \vee \bar{x}\,z = x\,1 \vee \bar{x}\,z \vee 1\,z = x \vee z.$$

Все операции алгебры логики можно выразить через булевы операции. Справедливость следующих формул можно доказать подстановкой значений из табл. 5.3:

$$x \oplus y = x\,\bar{y} \vee \bar{x}\,y;$$
$$x \sim y = \bar{x}\,\bar{y} \vee x\,y;$$
$$x \to y = \bar{x} \vee y.$$

Пользуясь этими формулами, построим булево выражение, эквивалентное формуле $((x \to y) \vee (x \oplus z))\,\bar{y}$:

$$((x \to y) \vee (x \oplus z))\,\bar{y} = (\,\bar{x} \vee y \vee x\,\bar{z} \vee \bar{x}\,z)\,\bar{y} = (\,\bar{x} \vee y \vee \bar{z})\,\bar{y} = \bar{x}\,\bar{y} \vee \bar{y}\,\bar{z}.$$

5.3. Интерпретации булевой алгебры

Рассматриваемая абстрактная булева алгебра имеет ряд интерпретаций, используемых в различных приложениях.

В *булевой алгебре множеств* значениями переменных служат подмножества универсального множества U. Константам 1 и 0 соответствуют множества U и $\varnothing$. Все соотношения, приведенные в разд. 1, совпадут с основными законами абстрактной булевой алгебры, если операцию дополнения множества заме-

нить на операцию отрицания, а операции $\cap$ и $\cup$ (пересечения и объединения множеств) – соответственно на операции $\wedge$ и $\vee$ (конъюнкции и дизъюнкции).

Интерпретацией абстрактной булевой алгебры является также *алгебра событий*, используемая в теории вероятностей. Алгебру событий составляют семейство подмножеств множества элементарных событий U и определяемые над этими подмножествами операции отрицания ($\neg$), объединения ($\cup$) и пересечения ($\cap$). Любое событие может произойти или не произойти (наступить или не наступить). Отсутствие события A обозначается как $\neg A$. Событие, состоящее в наступлении обоих событий A и B, называется *произведением* событий A и B и обозначается $A \cap B$ или AB. Событие, состоящее в наступлении хотя бы одного из событий A и B, называется *суммой* событий A и B и обозначается $A \cup B$.

Еще одной интерпретацией является *алгебра переключательных схем*. Переменным этой алгебры соответствуют элементы переключательной схемы – переключатели. Переключательный элемент, состояние которого представляется булевой переменной a, может быть замкнут, тогда через него течет ток и $a = 1$. Если он разомкнут, то тока нет и $a = 0$. По состояниям переключателей в схеме можно определить, проходит ли по данной схеме ток. На рис. 5.1, *а* изображено последовательное соединение двух переключателей a и b. Данная схема будет пропускать ток в том и только в том случае, когда оба переключателя замкнуты, т. е. если $a \wedge b = 1$. На рис. 5.1, *б* изображено параллельное соединение переключателей a и b. Ток будет протекать, если замкнут хотя бы один из переключателей, т. е. если $a \vee b = 1$.

Рис. 5.1. Примеры соединения переключателей:
а – последовательное; *б* – параллельное

Два или более переключателей можно условно связать таким образом, чтобы они замыкались и размыкались одновременно. Каждому переключателю можно поставить в соответствие другой переключатель так, чтобы когда один из них замкнут, другой был разомкнут. Если один из них обозначить буквой a, то другой примет обозначение $\bar{a}$. В схеме на рис. 5.2 пойдет ток, если и только если $a\,b \vee b\,\bar{c} \vee \bar{a}\,\bar{b} = 1$. Левая часть этого уравнения представляет структуру схемы.

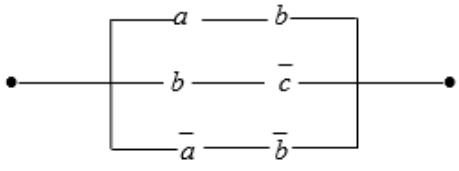


Рис. 5.2. Пример переключательной схемы

В *исчислении высказываний* переменными являются высказывания, принимающие истинные или ложные значения, которые соответствуют константам 1 и 0. Символы операций и их названия в данном случае совпадают не случайно. На основе исчисления высказываний можно выделить *булеву алгебру высказываний*, которая является одной из интерпретаций абстрактной булевой алгебры. Высказывание $\neg a$ истинно тогда и только тогда, когда a ложно. Оно читается как «не a» или «не верно, что a». Высказывание $a \wedge b$, читаемое как «a и b», истинно тогда и только тогда, когда истинны оба высказывания a и b. Высказывание $a \vee b$ читается как «a или b». Оно истинно, если хотя бы одно из высказываний a и b истинно, и ложно, если оба высказывания ложны.

Другие операции алгебры логики также могут иметь интерпретации в исчислении высказываний. Союз «или» может быть использован при прочтении высказывания $a \oplus b$. Наряду с прочтением «a либо b» его можно читать как «или a, или b». Оно истинно, когда истинно только одно из высказываний a и b, и ложно, когда оба высказывания истинны или оба ложны. Высказывание $a \leftrightarrow b$ истинно тогда и только тогда, когда значения истинности высказываний a и b совпадают. Это высказывание может быть прочитано следующим образом: «a равносильно b», «a, если и только если b», «a тогда и только тогда, когда b». Импликация $a \to b$ читается как «если a, то b». Это высказывание ложно, когда a истинно, а b ложно. Во всех остальных случаях оно истинно.

Познакомимся еще с одной алгеброй на множестве логических функций – *алгеброй Жегалкина*. Алгебра над множеством логических функций с двумя бинарными операциями $\wedge$ и $\oplus$, двумя константами 1 и 0 называется алгеброй Жегалкина, если в ней выполняются следующие законы:

$$x \oplus y = y \oplus x;\ x\,(y \oplus z) = x\,y \oplus x\,z;\ x \oplus x = 0;\ x \oplus 0 = x;\ x \oplus 1 = \bar{x};$$
$$x\,y = y\,x;\ x\,(y\,z) = (x\,y)\,z;\ x\,x = x.$$

В алгебре Жегалкина дизъюнкция $x \vee y$ выражается формулой $x\,y \oplus x \oplus y$, из которой видно, что $x \vee y = x \oplus y$ тогда, когда $x\,y = 0$ (когда x и y ортогональны).

Действительно,

$$x \vee y = \overline{\overline{x \vee y}} = \overline{\bar{x}\,\bar{y}} = \overline{(x \oplus 1)(y \oplus 1)} = (x \oplus 1)(y \oplus 1) \oplus 1 =$$
$$= x\,y \oplus x \oplus y \oplus 1 \oplus 1 = x\,y \oplus x \oplus y.$$

Всякую формулу алгебры Жегалкина можно представить в виде полинома Жегалкина.

$$f(x_1, \ldots, x_n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus \ldots \oplus c_nx_n \oplus c_{n+1}x_1x_2 \oplus c_{n+2}x_1x_3 \oplus \ldots$$
$$\ldots \oplus c_{2^n-1}x_1x_2\ldots x_{n-1}x_n,$$

где $c_i \in \{0, 1\}$.

Для всякой логической функции существует единственный полином Жегалкина.

5.4. Функциональная полнота

Система булевых функций $\{f_1, f_2, \dots, f_m\}$ называется *функционально полной*, или просто *полной*, если любая булева функция может быть представлена в виде суперпозиции этих функций. Полную систему булевых функций называют еще *базисом*.

Минимальным базисом называется такой базис $\{f_1, f_2, \dots, f_m\}$, для которого удаление хотя бы одной из функций $f_1, f_2, \dots, f_m$ превращает его в неполную систему.

Функции от двух переменных, представляемые булевыми операциями $\bar{}$ (отрицание), $\wedge$ (конъюнкция) и $\vee$ (дизъюнкция), образуют полную систему. Действительно, из теоремы Шеннона следует, что любую булеву функцию можно представить в виде совершенной дизъюнктивной нормальной формы (СДНФ), которая представляет собой суперпозицию отрицания, конъюнкции и дизъюнкции.

Базис $\{\bar{}, \wedge, \vee\}$ не является минимальным. Одну из операций, $\wedge$ или $\vee$, из него можно удалить. Пользуясь правилами де Моргана и законом двойного отрицания, можно дизъюнкцию выразить через отрицание и конъюнкцию, а конъюнкцию – через отрицание и дизъюнкцию:

$$a \vee b = \overline{\overline{a \vee b}} = \bar{\bar{a} \wedge \bar{b}};$$
$$a \wedge b = \overline{\overline{a \wedge b}} = \bar{\bar{a} \vee \bar{b}}.$$

Система $\{\wedge, \vee\}$ не является полной, так как операцию отрицания нельзя выразить через операции $\wedge$ и $\vee$.

Чтобы убедиться в полноте некоторой системы функций, достаточно через эти функции выразить любую функцию из некоторой известной полной системы. Покажем, что каждая из операций $|$ (штрих Шеффера) и $\uparrow$ (стрелка Пирса) составляет полную систему, использовав для этого базис $\{\bar{}, \wedge\}$:

$$\bar{a} = a \mid a;\ a \wedge b = \overline{\overline{a \wedge b}} = \overline{a \mid b} = (a \mid b) \mid (a \mid b);$$
$$\bar{a} = a \uparrow a;\ a \wedge b = \overline{\overline{a \wedge b}} = \bar{\bar{a} \vee \bar{b}} = \bar{a} \uparrow \bar{b} = (a \uparrow a) \uparrow (b \uparrow b).$$

Примером полной системы булевых функций является система, содержащая константу 1, а также функции, выражаемые операцией $\wedge$ и операцией $\oplus$ (сложение по модулю два), что представляет алгебру Жегалкина.

Вопрос о функциональной полноте системы булевых функций имеет практический смысл: набор логических элементов, из которых строятся разнообразные схемы, должен содержать элементы, реализующие все функции из заданного базиса.

Рассмотрим функции $g(y_1, y_2, \ldots, y_k)$, $f_1(x_1, x_2, \ldots, x_n)$, …, $f_k(x_1, x_2, \ldots, x_n)$. Будем считать, что функции $f_1, f_2, \ldots, f_k$зависят от одних и тех же аргументов $x_1, x_2, \ldots, x_n$. Этого можно достигнуть, добавив при необходимости к аргументам некоторых функций фиктивные аргументы.

Некоторый класс A логических функций назовем *замкнутым*, если для всяких функций $g(y_1, y_2, \ldots, y_k)$, $f_1, f_2, \ldots, f_k$ из A их суперпозиция

$$h(x_1, x_2, \ldots, x_n) = g(f_1(x_1, x_2, \ldots, x_n), \ldots, f_k(x_1, x_2, \ldots, x_n))$$

содержится в A.

Перечислим пять замкнутых классов логических функций:

1. Класс функций T_0, сохраняющих константу 0, содержит функции, обладающие свойством $f(0, 0, ..., 0) = 0$.

2. Класс функций T_1, сохраняющие константу 1, содержит функции, обладающие свойством $f(1, 1, ..., 1) = 1$.

3. Класс линейных функций L, для которых полином Жегалкина линеен:

$$f(x_1, x_2, \ldots, x_n) = c_0 \oplus c_1x_1 \oplus c_2x_2 \oplus \ldots \oplus c_nx_n, \quad c_i \in \{0, 1\}.$$

Алгоритм построения полинома Жегалкина (совершенной полиномиальной формы Жегалкина) логической функции состоит из следующих шагов:

– построить формулу с использованием связок $\{\wedge, \neg\}$ или построить СДНФ функции;

– заменить всюду $\bar{x}$ на $x \oplus 1$; если построена СДНФ, заменить в ней все операции $\vee$ на операции $\oplus$, так как для ортогональных элементарных конъюнкций имеет место соотношение $x \vee y = x \oplus y$;

– раскрыть скобки, пользуясь законом $(x \oplus y)\, z = x\, z \oplus y\, z$, и привести подобные члены, используя правило алгебры Жегалкина: $x \oplus x = 0$.

4. Класс самодвойственных функций S, для которых выполняется условие $f(x_1, x_2, \ldots, x_n) = \bar{f}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$, т. е. на всех инверсных наборах значения функции различны.

5. Класс монотонных функций M, для которых выполняется условие монотонности $f(\boldsymbol{x}) \geq f(\boldsymbol{x}')$ при $\boldsymbol{x} \geq \boldsymbol{x}'$.

*Теорема Поста о функциональной полноте (критерий полноты системы логических функций).*Система функций $f_1, f_2, \ldots, f_k$ является полной тогда и только тогда, когда она целиком не содержится ни в одном из пяти замкнутых классов T_0, T_1, L, S, т. е. когда она содержит хотя бы одну функцию, не сохраняющую 0, хотя бы одну функцию, не сохраняющую 1, хотя бы одну несамодвойственную функцию, хотя бы одну немонотонную функцию, хотя бы одну нелинейную функцию.

6. НОРМАЛЬНЫЕ ФОРМЫ

6.1. Дизъюнктивные нормальные формы

Переменные $x_1, x_2, \ldots, x_n$ и их инверсии назовем *литералами* и введем обозначение a^σ, где $a^\sigma = a$, если $\sigma = 1$, и $a^\sigma = \bar{a}$, если $\sigma = 0$. *Элементарной конъюнкцией* K_i является многоместная конъюнкция попарно различных литералов, т. е. $K_i = x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} \ldots x_{i_r}^{\sigma_r}$. К элементарным конъюнкциям относятся также одиночные литералы и константа 1 – конъюнкция, состоящая из пустого множества литералов. Число литералов r элементарной конъюнкции называется ее *рангом*. Элементарная конъюнкция называется *полной* относительно переменных $x_1, x_2, \ldots, x_n$, если она содержит символы всех переменных (со знаком отрицания или без него). Ранг таких конъюнкций равен n.

Дизъюнктивная нормальная форма (ДНФ) – это выражение вида $\bigvee_{i=1}^{m} K_i$, т. е. дизъюнкция элементарных конъюнкций. Примером дизъюнктивной нормальной формы является выражение $x_1 \bar{x}_2 \vee x_2 x_3 x_4 \vee \bar{x}_1 x_3$, где две конъюнкции имеют ранг 2 и одна конъюнкция – ранг 3. Одна элементарная конъюнкция также может считаться ДНФ.

6.2. Дизъюнктивное разложение Шеннона

Т е о р е м а Ш е н н о н а. Любая булева функция $f(x_1, x_2, \ldots, x_n)$ при любом m $(1 \le m \le n)$ может быть представлена в следующем виде:

$$f(x_1, x_2, \ldots, x_n) = \bigvee_{\sigma_1, \sigma_2, \ldots, \sigma_m} x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_m^{\sigma_m} f(\sigma_1, \sigma_2, \ldots, \sigma_m, x_{m+1}, \ldots, x_n), \qquad (6.1)$$

где дизъюнкция берется по всевозможным 2^m наборам значений переменных $x_1, x_2, \ldots, x_m$.

Для доказательства теоремы подставим в обе части равенства (6.1) произвольный набор $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ значений всех n переменных. Заметим, что $x^\sigma = 1$ только при $x = \sigma$ и из всех 2^m конъюнкций $x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_m^{\sigma_m}$ правой части формулы (6.1) значение 1 примет единственная конъюнкция, а именно та, для которой $\sigma_1 = \alpha_1, \sigma_2 = \alpha_2, \ldots, \sigma_m = \alpha_m$. Остальные конъюнкции будут равны 0. Отсюда получим тождество

$$f(\alpha_1, \alpha_2, \ldots, \alpha_n) = \alpha_1^{\alpha_1} \alpha_2^{\alpha_2} \ldots \alpha_m^{\alpha_m} f(\alpha_1, \alpha_2, \ldots, \alpha_m, \alpha_{m+1}, \ldots, \alpha_n).$$

Представление (6.1) называется *дизъюнктивным разложением функции* $f(x_1, x_2, \ldots, x_n)$ по переменным $x_1, x_2, \ldots, x_m$. Получаемые в результате подстановки констант $\alpha_1, \alpha_2, \ldots, \alpha_m$ вместо переменных $x_1, x_2, \ldots, x_m$ функции

$f(\alpha_1, \alpha_2, \ldots, \alpha_m, x_{m+1}, \ldots, x_n)$, являющиеся коэффициентами разложения, не зависят от переменных $x_1, x_2, \ldots, x_m$.

Из теоремы Шеннона непосредственно вытекают два следующих утверждения, соответствующие двум крайним значениям числа m: $m = 1$ и $m = n$.

Любая булева функция $f(x_1, x_2, \ldots, x_n)$ при любом $i = 1, 2, \ldots, n$ может быть представлена в следующем виде:

$$f(x_1, x_2, \ldots, x_n) = x_i f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \vee \bar{x}_i f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n).$$

Любая булева функция $f(x_1, x_2, \ldots, x_n)$ может быть представлена в следующем виде:

$$f(x_1, x_2, \ldots, x_n) = \bigvee_{\sigma_1, \sigma_2, \ldots, \sigma_n} x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n} f(\sigma_1, \sigma_2, \ldots, \sigma_n). \qquad (6.2)$$

Последнее выражение является представлением булевой функции $f(x_1, x_2, \ldots, x_n)$ в СДНФ. Здесь $f(\sigma_1, \sigma_2, \ldots, \sigma_n)$ является значением функции на наборе значений аргументов $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, т. е. константой 0 или 1.

Легко построить СДНФ, представляющую произвольную булеву функцию, заданную в табличной форме. Для этого достаточно выделить наборы $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, на которых функция принимает значение 1, и для каждого из них ввести в СДНФ полную элементарную конъюнкцию, где любая переменная x_i присутствует с отрицанием, если $\sigma_i = 0$, и без отрицания, если $\sigma_i = 1$.

Очевидно, для любой булевой функции $f(x_1, x_2, \ldots, x_n)$, кроме константы 0, существует единственная СДНФ (с точностью до порядка литералов и конъюнкций). Поэтому данная форма представления булевой функции является *канонической*. Например, СДНФ для функции от трех аргументов, заданной табл. 6.1, имеет следующий вид:

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 .$$

Константа 1 представляется в виде СДНФ, которая содержит все различные полные элементарные конъюнкции, называемые *конституентами единицы* (в литературе используется также термин *минтерм*). Конституента единицы принимает значение 1 на единственном наборе значений переменных.

Таблица 6.1

Задание функции $f(x, y, z)$

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

6.3. Конъюнктивные нормальные формы

Элементарной дизъюнкцией D_i является многоместная конъюнкция попарно различных литералов, т. е. $D_i = x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee \ldots \vee x_{i_r}^{\sigma_r}$. К элементарным дизъюнкциям относятся также одиночные литералы и константа 0 – дизъюнкция, состоящая из пустого множества литералов. Число литералов r элементарной дизъюнкции называется ее *рангом*. Элементарная дизъюнкция называется *полной* относительно переменных $x_1, x_2, \ldots, x_n$, если она содержит символы всех переменных (со знаком отрицания или без него). Ранг таких дизъюнкций равен n.

Конъюнктивная нормальная форма (КНФ) – это выражение вида $\bigwedge_{i=1}^{m} D_i$, т. е. конъюнкция элементарных дизъюнкций. Примером конъюнктивной нормальной формы является выражение $(x_2 \vee \bar{x}_3 \vee x_4)(x_1 \vee \bar{x}_2)$. Одна элементарная дизъюнкция также может считаться КНФ.

Согласно принципу двойственности формулу (6.1) можно преобразовать в следующее выражение, которое также справедливо:

$$f(x_1, x_2, \ldots, x_n) = \bigwedge_{\sigma_1, \sigma_2, \ldots, \sigma_m} (x_1^{\bar{\sigma}_1} \vee x_2^{\bar{\sigma}_2} \vee \ldots \vee x_m^{\bar{\sigma}_m} \vee f(\sigma_1, \sigma_2, \ldots, \sigma_m, x_{m+1}, \ldots, x_n)).$$

Эта формула называется *конъюнктивным разложением функции* $f(x_1, x_2, \ldots, x_n)$ по переменным $x_1, x_2, \ldots, x_m$. Справедливость ее может быть доказана так же, как справедливость формулы (6.1). Также крайними случаями конъюнктивного разложения являются разложение по одной переменной и по всем переменным. Последнее называется *совершенной конъюнктивной нормальной формой* (СКНФ) и имеет вид

$$f(x_1, x_2, \ldots, x_n) = \bigwedge_{\sigma_1, \sigma_2, \ldots, \sigma_n} (x_1^{\bar{\sigma}_1} \vee x_2^{\bar{\sigma}_2} \vee \ldots \vee x_n^{\bar{\sigma}_n} \vee f(\sigma_1, \sigma_2, \ldots, \sigma_n)).$$

СКНФ, представляющую произвольную булеву функцию, так же, как ее СДНФ, легко построить по табличному заданию этой функции. Согласно формуле достаточно выделить наборы $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, на которых функция принимает значение 0 (если $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$, то весь сомножитель $(x_1^{\bar{\sigma}_1} \vee x_2^{\bar{\sigma}_2} \vee \ldots \vee x_n^{\bar{\sigma}_n} \vee 1)$ обращается в 1), и для каждого из них ввести в СДНФ полную элементарную дизъюнкцию, где любая переменная x_i присутствует с отрицанием, если $\sigma_i = 1$, и без отрицания, если $\sigma_i = 0$.

Очевидно, для любой булевой функции $f(x_1, x_2, \ldots, x_n)$, кроме константы 1, существует единственная СКНФ (с точностью до порядка литералов и дизъюнкций). Так же, как СДНФ, эта форма представления булевой функции является *канонической*. СКНФ для функции, которую задает табл. 6.1, имеет следующий вид:

$$(x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3).$$

Константа 0 представляется в виде СКНФ, которая содержит все различные полные элементарные дизъюнкции, называемые *конституентами нуля* (в литературе используется также термин *макстерм*). Конституента нуля принимает значение 0 на единственном наборе значений переменных.

6.4. Локальные упрощения ДНФ

Дизъюнктивная нормальная форма *безызбыточна* [6], если из нее нельзя удалить ни одной элементарной конъюнкции и ни одного литерала из какой-либо конъюнкции. Это равносильно тому, что из представляющей данную ДНФ троичной матрицы нельзя удалить ни одну из строк и нельзя ни одно из значений 0 или 1 заменить на «–». Локальные упрощения ДНФ сводятся к поиску и последовательному удалению таких элементарных конъюнкций и литералов до тех пор, пока данная ДНФ не станет безызбыточной. Простейшие случаи сокращения определяются формулами:

$$A\,x \vee A = A; \quad A\,\bar{x} \vee x = A \vee x; \quad A\,x \vee B\,\bar{x} \vee AB = A\,x \vee B\,\bar{x}.$$

Более сложный случай представляет ДНФ

$$\bar{x}_1\,\bar{x}_2\,x_3 \vee x_1\,\bar{x}_2\,x_4 \vee x_1\,x_2\,x_3 \vee\vee\,\bar{x}_1\,x_2\,x_4 \vee x_3\,x_4,$$

где конъюнкция $x_3\,x_4$ является избыточной. Действительно, если ее заменить на $x_3\,x_4\,1 = x_3\,x_4\,(x_1\,x_2 \vee x_1\,\bar{x}_2 \vee \bar{x}_1\,x_2 \vee \bar{x}_1\,\bar{x}_2)$, а затем раскрыть скобки, то каждая из конъюнкций ранга 4 окажется поглощаемой некоторой из конъюнкций ранга 3, присутствующей в исходной ДНФ.

Исходя из сказанного, отметим два вида избыточности:

$$D = k \vee D' = D' \quad \text{и} \quad D = xk \vee D' = k \vee D',$$

где k – элементарная конъюнкция; x – литерал, входящий в элементарную конъюнкцию xk; D – некоторая ДНФ; D' – ДНФ, получаемая из D удалением конъюнкции k или удалением литерала x.

В первом случае элементарная конъюнкция k избыточна, если $k \vee D' = D'$. Это значит, что k и D' находятся в отношении *формальной импликации*, т. е. $k \Rightarrow D'$. Функция g *имплицирует* функцию f, если f имеет значение 1 везде, где имеет значение 1 функция g. В рассматриваемом случае ДНФ D' обращается в единицу при любом наборе значений переменных, обращающем конъюнкцию k в единицу, независимо от того, какие значения принимают переменные, не входящие в k.

Пусть троичная матрица V представляет ДНФ D', а троичный вектор v – элементарную конъюнкцию k. Тогда результатом подстановки в D' значений переменных, обращающих конъюнкцию k в единицу, является минор матрицы V, образованный строками, не ортогональными вектору v, и столбцами, соответствующими компонентам вектора v и имеющими значение «–». Если этот минор является вырожденной матрицей, т. е. D' тождественно равна единице, то конъюнкция k избыточна. В противном случае вектор, ортогональный всем строкам полученного минора, представляет набор значений переменных, обращающий D' в нуль.

Рассмотрим следующую троичную матрицу и проверим на избыточность ее первую строку:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\ \left[\begin{matrix} 0 & 1 & - & 0 & 1 & - \\ 0 & 1 & 1 & 0 & - & 0 \\ 0 & - & 0 & 0 & 1 & - \\ - & 1 & 1 & - & 1 & 1 \\ - & 1 & 0 & - & - & 1 \\ 1 & - & 1 & - & 0 & - \end{matrix}\right] & \begin{matrix} 1 \\ 2 \\ 3. \\ 4 \\ 5 \\ 6 \end{matrix} \end{matrix}$$

Минор, образованный столбцами x_3 и x_6, где элементы первой строки имеют значение «–», и строками 2, 3, 4 и 5, не ортогональными первой строке, имеет вид

$$\begin{matrix} x_3 & x_6 & \\ \left[\begin{matrix} 1 & 0 \\ 0 & - \\ 1 & 1 \\ 0 & 1 \end{matrix}\right] & \begin{matrix} 2 \\ 3. \\ 4 \\ 5 \end{matrix} \end{matrix}$$

Эта матрица является вырожденной, следовательно, первая строка избыточна. Любой входящий в нее булев вектор принадлежит некоторому интервалу, представляемому какой-либо из строк данной матрицы.

Удалив строку 1, получим матрицу, в которой строка 2 ортогональна всем остальным ее строкам. Это значит, что никакой булев вектор, принадлежащий интервалу, представляемому данной строкой, не принадлежит никакому из других интервалов, представляемых остальными строками. Соответствующий минор является пустой матрицей (с пустым множеством строк). Такая матрица представляет константу 0. Таким образом, строка 2 не является избыточной.

Что касается строки 3, то соответствующий минор является однострочной невырожденной матрицей:

$$\begin{matrix} x_2 & x_6 \\ \begin{bmatrix} 1 & 1 \end{bmatrix} & 5. \end{matrix}$$

Ортогональным вектором для данной строки является (0 –). Подставив 0 во вторую компоненту строки 3, получим вектор, ортогональный всем строкам матрицы. Строка 3 также является неизбыточной для заданной матрицы.

Выполняя подобные построения над остальными строками, убедимся, что они также не являются избыточными.

Рассмотрим второй вид избыточности в ДНФ, когда $D = xk \vee D' = k \vee D'$. Здесь избыточным является литерал x. Правую часть этого равенства можно представить следующим образом:

$$k \vee D' = k(x \vee \bar{x}) \vee D' = xk \vee D' \vee \bar{x}k = D \vee \bar{x}k.$$

Отсюда видно, что литерал x в выражении $xk \vee D'$ является избыточным, если конъюнкция $\bar{x}k$ является избыточной в выражении $D \vee \bar{x}k$. Следовательно, задача определения избыточности литерала в ДНФ сводится к предыдущей задаче – задаче определения избыточности элементарной конъюнкции.

Удаление литерала из ДНФ в матричном представлении выражается в замене нуля или единицы в троичной матрице на значение «–». На основании предыдущих рассуждений это можно сделать, если вектор, полученный из строки, содержащей данный нуль или единицу, заменой этого значения на противоположное ему значение (т. е. 0 на 1 или 1 на 0), является избыточным для рассматриваемой матрицы.

Таким образом, для того чтобы решить вопрос о том, можно ли заменить 0 (или 1) в i-й строке и j-м столбце на значение «–», надо построить минор, образованный столбцами, где i-я строка имеет значения «–», и строками, не ортогональными вектору, полученному из i-й строки заменой 0 (или 1) в j-м столбце на противоположное значение. Если полученный минор оказался вырожденной матрицей, то такая замена возможна.

Рассмотрим матрицу

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\ \begin{bmatrix} 0 & 1 & 1 & 0 & - & 0 \\ 0 & - & 0 & 0 & 1 & - \\ - & 1 & 1 & - & 1 & 1 \\ - & 1 & 0 & - & - & 1 \\ 1 & - & 1 & - & 0 & - \end{bmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} \end{matrix}.$$

Чтобы узнать, является ли 0 в строке 1 и столбце x_6 избыточным, построим минор, образованный единственным столбцом x_5, где строка 1 имеет значение «–», и единственной строкой 3, не ортогональной вектору (0 1 1 0 – 1).

Единственный элемент в этом миноре имеет значение 1. Он является невырожденной матрицей. Следовательно, нуль в строке 1 и столбце x_6 нельзя заменить на «–».

Рассмотрим теперь единицу в строке 3 и столбце x_3. Минор, образованный столбцами x_1 и x_4 и строками 2 и 4, не ортогональными вектору (– 1 0 – 1 1), имеет вид

$$\begin{matrix} x_1 & x_4 & \\ \begin{bmatrix} 0 & 0 \\ - & - \end{bmatrix} & & \begin{matrix} 2. \\ 4 \end{matrix} \end{matrix}$$

Вырожденность этого минора говорит о том, что данную единицу можно заменить значением «–». Выполнив такую замену, получим матрицу, эквивалентную исходной матрице:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\ \begin{bmatrix} 0 & 1 & 1 & 0 & - & 0 \\ 0 & - & 0 & 0 & 1 & - \\ - & 1 & - & - & 1 & 1 \\ - & 1 & 0 & - & - & 1 \\ 1 & - & 1 & - & 0 & - \end{bmatrix} & & & & & & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} \end{matrix}.$$

7. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ БУЛЕВА ПРОСТРАНСТВА И БУЛЕВЫХ ФУНКЦИЙ

7.1. Булев гиперкуб

Булево пространство M можно представить в виде графа, вершины которого соответствуют элементам пространства, а ребра представляют *отношение соседства* между элементами пространства. Два вектора являются соседними, если они отличаются друг от друга значением только одной компоненты. Например, векторы (1 0 0 1) и (1 1 0 1), значения одноименных компонент которых, кроме одной второй компоненты, совпадают, являются соседними. Данный граф, представляющий n-мерное булево пространство, имеет 2^n вершин и $n2^{n-1}$ ребер. Он называется *полным булевым графом*, или *n-мерным гиперкубом*. Рассмотрим построение такого гиперкуба для различных значений размерности пространства.

Одномерный гиперкуб состоит из двух вершин, связанных ребром. Одной из этих вершин приписывается константа 0, другой – константа 1, являющиеся кодами данных вершин. Чтобы получить двумерный гиперкуб, необходимо продублировать одномерный гиперкуб и каждую вершину исходного гиперкуба соединить ребром с ее дублем. Коды вершин построенного двумерного гиперкуба получаются добавлением нулей справа к кодам вершин исходного гипер-

куба и единиц – к кодам дублей вершин. Аналогично получаются трехмерный гиперкуб, четырехмерный гиперкуб и т. д.

Сформулируем общее правило увеличения размерности гиперкуба: для перехода от m-мерного гиперкуба к $(m + 1)$-мерному необходимо исходный m-мерный гиперкуб продублировать и каждую вершину исходного гиперкуба соединить ребром с ее дублем. В полученном гиперкубе к кодам вершин исходного m-мерного гиперкуба добавляются справа нули, а к кодам их дублей – единицы.

Последовательность гиперкубов от одномерного до четырехмерного представлена на рис. 7.1.

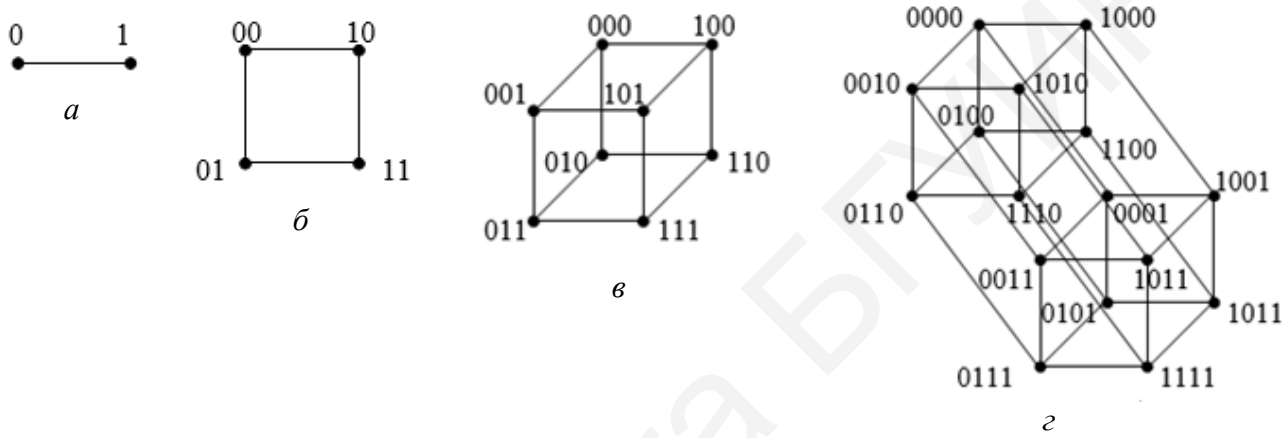


Рис. 7.1. Графическое представление булева пространства:
а – одномерное; *б* – двумерное; *в* – трехмерное; *г* – четырехмерное

В гиперкубе выделяются гиперграни, которые являются порожденными подграфами, представляющими собой гиперкубы меньшей размерности, чем рассматриваемый гиперкуб. Это может быть отдельное ребро, двумерная грань, трехмерный куб и т. п. Подграф, представляющий гипергрань, порождается множеством вершин, составляющих интервал булева пространства.

7.2. Представление булевых функций на гиперкубе

Любой интервал булева пространства является характеристическим множеством функции, выражаемой в алгебраической форме одной элементарной конъюнкцией. Например, конъюнкции $x_1\,\bar{x}_3\,x_4$ соответствует интервал четырехмерного пространства, представляемый троичным вектором (1 – 0 1). Интервалу приписывается ранг той конъюнкции, которую он представляет.

На гиперкубе булева функция $f(x_1, x_2, \ldots, x_n)$ задается выделением вершин, представляющих элементы ее характеристического множества M_f[1]. Например, задание функции $f(x_1, x_2, \ldots, x_n) = \bar{x}_1\,\bar{x}_2\,x_3 \vee \bar{x}_1\,x_2\,\bar{x}_3 \vee \bar{x}_1\,x_2\,x_3 \vee x_1\,\bar{x}_2\,x_3 \vee x_1\,x_2\,x_3$ может быть показано светлыми кружками, как на рис. 7.2.

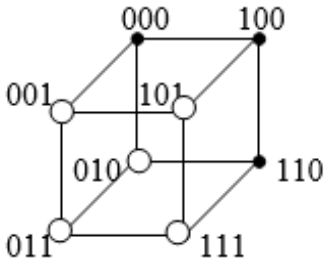


Рис. 7.2. Трехмерный гиперкуб с заданной на нем булевой функцией

В изображенном гиперкубе легко заметить две гиперграни, составляющие множество M_f^1. Они представляют интервалы, задаваемые троичными векторами (– – 1) и (0 1 –), которые являются характеристическими множествами элементарных конъюнкций x_3 и $\bar{x}_1 x_2$ соответственно. Поэтому рассматриваемую функцию можно задать как $f(x_1, x_2, x_3) = x_3 \vee \bar{x}_1 x_2$. Выполнение простого склеивания над исходной формулой дает тот же результат. Таким образом, графическое представление булевой функции дает возможность непосредственно получить ее задание в виде компактной формулы.

На рис. 7.3, *а* видно, что ребра, представляемые векторами (0 1 –) и (1 1 –), образуют гиперграни более высокой размерности, представляемую вектором (– 1 –). Это соответствует простому склеиванию: $\bar{x}_1 x_2 \vee x_1 x = x_2$.

На рис. 7.3, *б* видно, что интервал, соответствующий конъюнкции $x_1 x_2$, поглощает элемент булева пространства, соответствующий конъюнкции $x_1 x_2 \bar{x}_3$. Тем самым демонстрируется простое поглощение, выражаемое формулой $x_1 x_2 \vee x_1 x_2 \bar{x}_3 = x_1 x_2$.

На рис. 7.3, *в* продемонстрирована формула обобщенного склеивания: $\bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_3 = \bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_3 \vee \bar{x}_2 \bar{x}_3$. Продукт обобщенного склеивания (0 – 0) поглощается совокупностью интервалов (0 0 –) и (– 1 0).

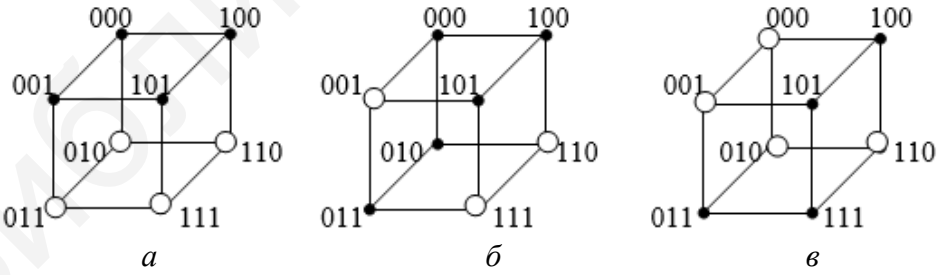


Рис. 7.3. Графическое представление некоторых формул булевой алгебры:
а – простое склеивание; *б* – простое поглощение; *в* – обобщенное склеивание

7.3. Развертка гиперкуба на плоскости. Карта Карно

Как видно из рис. 7.1, с увеличением размерности булева пространства его графическое представление в виде гиперкуба быстро становится трудным

для восприятия. Более удобным является развертка гиперкуба на плоскости. Пример такой развертки трехмерного гиперкуба, которая получена удалением двух ребер и расположением верхних и нижних ребер в две параллельных линии, показан на рис. 7.4. На этом же рисунке представлена функция $f(x_1, x_2, x_3) = x_1 x_2 \vee \bar{x}_1 \bar{x}_2 x_3$.

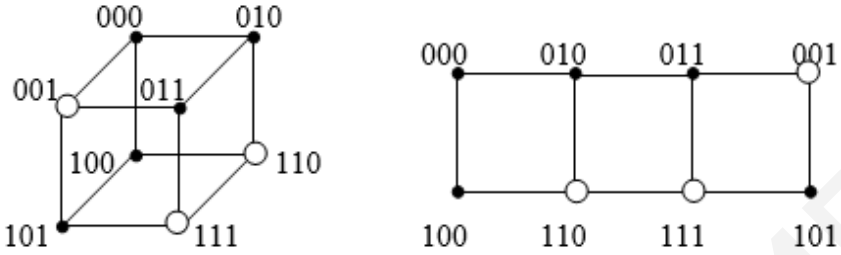


Рис. 7.4. Трехмерный гиперкуб и его развертка на плоскости

Еще более удобной формой представления булева пространства является двумерная таблица, которую принято называть *картой Карно*. Карта Карно имеет ту же структуру, что и развертка гиперкуба на плоскости. Каждая ее клетка соответствует элементу булева пространства. Булеву функцию можно задать расположением нулей и единиц в клетках в соответствии с теми значениями, которые принимает функция на соответствующих элементах булева пространства. Другим способом задания булевой функции на карте Карно является разметка клеток, соответствующих элементам множества M_f^1. При этом клетки, соответствующие элементам множества M_f^0, остаются пустыми. Оба способа представлены на рис. 7.5, где показан пример задания функции $f(x_1, x_2, x_3) = x_1 x_2 \vee \bar{x}_1 \bar{x}_2 x_3$. Коды строк и столбцов, из которых составляются коды клеток, представлены отрезками прямых. Отрезок вертикальной прямой возле нижней строки показывает, что переменная x_1 в коде этой строки имеет значение 1, а его отсутствие у верхней строки говорит, что в ее коде $x_1 = 0$. Аналогично горизонтальные отрезки показывают значения переменных x_2 и x_3 в кодах столбцов.

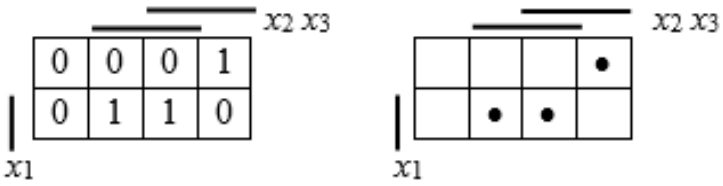


Рис. 7.5. Задание булевой функции с помощью карты Карно

Удобство работы с картами Карно обеспечивается применением *кода Грея* для кодирования строк и столбцов.

Пусть необходимо закодировать в коде Грея последовательность некоторых объектов, число которых равно N. Коды этих объектов, так же как и коды в виде двоичных чисел, являются булевыми векторами. Длина кода n должна

быть такой, чтобы выполнялось $N \le 2^n$, или $n = \lceil \log_2 N \rceil$, где $\lceil a \rceil$ – ближайшее к a сверху целое число. Первому объекту присваивается код, состоящий только из нулей, – 0 0 … 0. Далее коды определяются по следующему правилу.

Для получения следующего кода берется последний код и в нем меняется значение той правой компоненты, изменение значения которой приводит к новому коду. Коды соседних в последовательности объектов оказываются, таким образом, отличающимися только значением одной компоненты.

Другой способ построения кода Грея заключается в следующем. Сначала берется последовательность из двух однокомпонентных кодов: (0), (1). Приписав к этой последовательности те же коды в обратном порядке, получим (0), (1), (1), (0). Добавляем слева 0 к элементам исходной последовательности и 1 – к приписанной последовательности. Получим (0 0), (0 1), (1 1), (1 0). К этой последовательности приписываем (1 0), (1 1), (0 1), (0 0) и снова добавляем к исходной части слева 0, а к приписанной части – 1. Получаем (0 0 0), (0 0 1), (0 1 1), (0 1 0), (1 1 0), (1 1 1), (1 0 1), (1 0 0). Эти действия повторяем необходимое число раз в зависимости от количества кодируемых объектов.

Благодаря коду Грея два соседних элемента булева пространства или два соседних интервала расположены на карте Карно симметрично некоторой оси, т. е. отношение соседства элементов булева пространства представляется отношением симметрии в карте Карно. На рис. 7.6 показана шестимерная карта Карно с осями симметрии. Оси симметрии проходят в местах изменения значений переменных в кодах строк и столбцов. Каждая ось имеет свою *зону симметрии*, ширина которой определяется *рангом* оси.

Оси, связанной с переменной, наиболее часто меняющей свое значение в последовательности кодов строк (или столбцов), придается ранг 1. Если ось связана с переменной, которая меняет свое значение в два раза меньше, ей приписывается ранг 2, если в четыре раза меньше, – ранг 3 и т. д. Ширина оси симметрии ранга k равна 2^k (см. рис. 7.6).

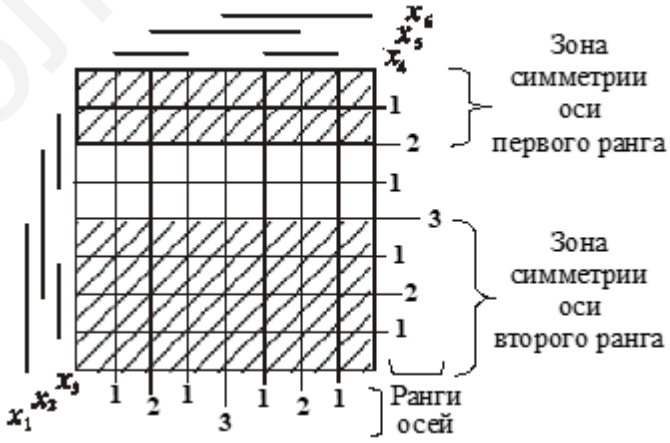


Рис. 7.6. Зоны симметрии карты Карно

По карте Карно легко построить упрощенную ДНФ функции, которая задана с помощью этой карты. Для этого необходимо выделить интервалы, на которых функция принимает значение 1. На карте Карно они представлены единичными областями, симметричными относительно некоторых осей. Каждый интервал должен быть *максимальным*, т. е. не быть собственным подмножеством другого интервала. Элементарная конъюнкция, соответствующая такому интервалу, не содержит переменных, с которыми связаны данные оси. Если на всем интервале некоторая переменная x имеет значение 0, то она берется с отрицанием, если значение 1, – то без отрицания.

Рекомендуется в первую очередь выделять те максимальные интервалы, где имеется элемент, для которого данный максимальный интервал является единственным, его содержащим. Такие интервалы называются *обязательными*, а соответствующие элементы – *определяющими*.

Если пользоваться «жадным» способом, т. е. стараться покрыть одним интервалом как можно большее число элементов, то в полученной ДНФ может оказаться избыточная элементарная конъюнкция, как, например, для функции $f(x_1\, x_2\, x_3\, x_4) = x_1\, \bar{x}_2\, x_3 \vee x_1\, x_2\, x_4 \vee \bar{x}_1\, x_2\, x_3 \vee \bar{x}_1\, \bar{x}_2\, x_4$, представленной картой Карно на рис. 7.7. Самый большой интервал, представленный конъюнкцией $x_3 x_4$, покрывается остальными интервалами, поэтому является избыточным.

Примером получения упрощенной ДНФ является получение ее для функции, представленной картой Карно на рис. 7.8, где определяющие элементы отмечены кружками. ДНФ этой функции имеет вид

$$x_2\, \bar{x}_3\, x_4\, \bar{x}_6 \vee \bar{x}_1\, x_3\, \bar{x}_4\, x_5 \vee x_2\, x_3\, \bar{x}_4\, x_5 \vee x_2\, x_3\, x_5\, x_6 \vee \bar{x}_1\, x_3\, x_4\, x_6 \vee \bar{x}_1\, x_2\, \bar{x}_3\, x_5\, \bar{x}_6.$$

Данная ДНФ является минимальной, поскольку из шести интервалов, покрывающих множество M_f^1, пять являются обязательными.

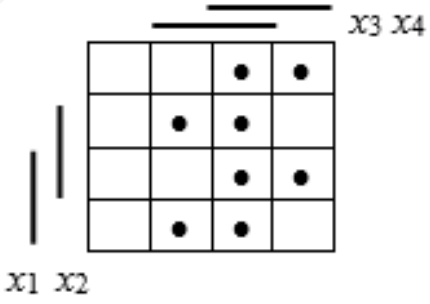


Рис. 7.7. Область M_f^1 с избыточным максимальным интервалом

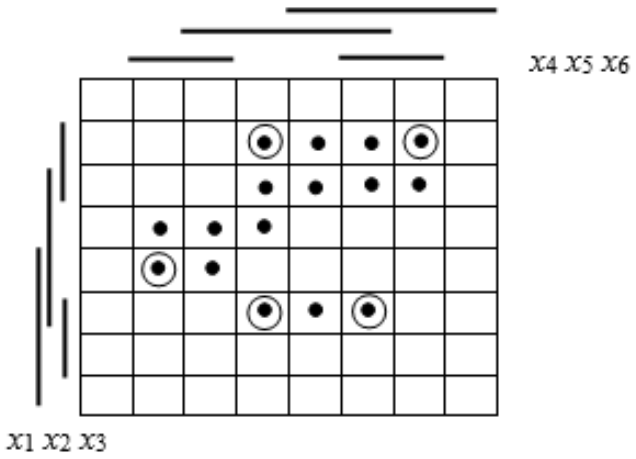


Рис. 7.8. Задание булевой функции с помощью карты Карно

8. МИНИМИЗАЦИЯ ДНФ

Задача минимизации ДНФ заключается в нахождении такой ДНФ для заданной булевой функции, которая содержала бы минимальное число элементарных конъюнкций или литералов. В первом случае результат решения называется *кратчайшей* ДНФ, во втором – *минимальной*.

8.1. Метод Квайна – МакКласки

Для описания метода введем некоторые понятия. В разд. 7 было введено понятие формальной импликации: функция g имплицирует функцию f, т. е. $g \Rightarrow f$, если f имеет значение 1 везде, где это значение имеет g. В этом случае функция g называется *импликантой* функции f. Очевидно, что всякая элементарная конъюнкция, входящая в ДНФ некоторой функции, является импликантой этой функции. Дизъюнкция любого множества импликант является также импликантой.

Простая импликанта – это импликанта в виде элементарной конъюнкции, которая перестает быть импликантой при удалении любого литерала. Заметим, что удаление литералов до пустого множества приводит к конъюнкции, представляющей константу 1. Характеристическим множеством простой импликанты является *максимальный интервал*, т. е. интервал, целиком содержащийся в единичной области M^1 функции f и не являющийся подмножеством другого интервала из M^1.

Дизъюнкция всех простых импликант некоторой булевой функции называется *сокращенной* ДНФ этой функции.

Метод Квайна – МакКласки требует представление заданной булевой функции в виде совершенной ДНФ, т. е. такой ДНФ, каждая конъюнкция которой имеет ранг, равный числу аргументов функции. Процесс минимизации состоит из двух этапов: 1) нахождение множества всех простых импликант задан-

ной функции; 2) выделение из этого множества минимального подмножества, составляющего ДНФ данной функции.

Первый этап выполняется путем применения операции простого склеивания над конъюнкциями. Пусть n – число аргументов заданной функции f. Рассмотрим последовательность множеств $C_0, C_1, \ldots, C_k$, где C_0 – множество конъюнкций ранга n, составляющих совершенную ДНФ функции f, C_i – множество конъюнкций ранга $n - i$, полученных путем склеивания конъюнкций из множества C_{i-1}, и C_k – множество конъюнкций ранга $n - k$, где нет ни одной пары соседних конъюнкций и дальнейшее склеивание невозможно. Если склеиваемым конъюнкциям приписывать некоторую метку, то неотмеченные конъюнкции составят множество всех простых импликант.

Выразим этот процесс через операции простого склеивания над булевыми и троичными векторами. Булева функция при этом задана своим характеристическим множеством M^1 – множеством элементов булева пространства, на которых она имеет значение 1. Удобно при этом сгруппировать исходные булевы векторы в подмножества, состоящие из векторов, имеющих одинаковое число единиц, и упорядочить эти подмножества по возрастанию (или убыванию) числа единиц в векторе. Тогда для каждого вектора соседние с ним векторы будут находиться только в соседних подмножествах в полученной последовательности.

В качестве примера рассмотрим булеву функцию, заданную следующей матрицей:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}$$
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Используя для множеств векторов те же обозначения, что использовались для соответствующих конъюнкций, получим последовательность матриц, где склеиваемые строки отмечены знаком «*»:

$$C_0 = \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{matrix} * \\ * \\ * \\ *, \\ * \\ * \\ * \\ * \end{matrix} \quad C_1 = \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 0 & 0 & - & 0 \\ - & 0 & 0 & 0 \\ 0 & 0 & 1 & - \\ 1 & 0 & 0 & - \\ 0 & - & 1 & 1 \\ - & 0 & 1 & 1 \\ 1 & 0 & - & 1 \\ - & 1 & 1 & 1 \\ 1 & - & 1 & 1 \end{bmatrix}\begin{matrix} * \\ * \\ * \\ * \\ * \\ * \\ * \\ * \\ * \end{matrix}, \quad C_2 = \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} - & - & 1 & 1 \end{bmatrix}.$$

Сокращенная ДНФ, которая является результатом выполнения первого этапа минимизации, представится следующей матрицей:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 0 & 0 & - & 0 \\ - & 0 & 0 & 0 \\ 0 & 0 & 1 & - \\ 1 & 0 & 0 & - \\ 1 & 0 & - & 1 \\ - & - & 1 & 1 \end{bmatrix}.$$

Второй этап сводится к задаче кратчайшего покрытия, описанной в подразд. 4.5. В данном случае множество M^1 надо покрыть минимальным числом интервалов, представленных строками троичной матрицы, задающей сокращенную ДНФ.

Продолжая решать пример и пользуясь обозначениями из подразд. 4.5, поставим задачу следующим образом. Заданы множество $A = M^1$ и совокупность подмножеств $B_1, B_2, \ldots, B_m$ множества A в виде матриц

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{matrix} \quad \text{и} \quad \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix}\begin{bmatrix} 0 & 0 & - & 0 \\ - & 0 & 0 & 0 \\ 0 & 0 & 1 & - \\ 1 & 0 & 0 & - \\ 1 & 0 & - & 1 \\ - & - & 1 & 1 \end{bmatrix}\begin{matrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{matrix}.$$

Требуется выделить минимум подмножеств B_i, покрывающих все множество A. Перейдя к матричной форме этой задачи, получим следующую матрицу:

$$\begin{array}{c} a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8 \\ \left[\begin{array}{cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{array} \end{array}.$$

Здесь надо выбрать минимальное количество строк так, чтобы каждый столбец имел единицу хотя бы в одной из них.

Строка B_6 является единственной строкой, которая покрывает столбцы a_6 и a_8. Поэтому ее включаем в решение. Удалив эту строку и покрываемые ею столбцы, получим матрицу

$$\begin{array}{c} a_1\ a_2\ a_3\ a_5 \\ \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array}\right] \begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{array} \end{array}.$$

Строки B_3 и B_5 удаляются по правилу редукции, как покрываемые строками B_1 и B_4 соответственно. Из оставшихся строк строки B_1 и B_4 покрывают оставшиеся столбцы. Таким образом, искомое кратчайшее покрытие составляют строки B_1, B_4 и B_6, а кратчайшая ДНФ, которая является также минимальной для заданной булевой функции, представляется матрицей

$$\begin{array}{c} x_1\ x_2\ x_3\ x_4 \\ \left[\begin{array}{cccc} 0 & 0 & - & 0 \\ 1 & 0 & 0 & - \\ - & - & 1 & 1 \end{array}\right] \end{array}.$$

В алгебраической форме это решение имеет вид $\bar{x}_1\,\bar{x}_2\,\bar{x}_4 \vee x_1\,\bar{x}_2\,\bar{x}_3 \vee x_3\,x_4$.

Иногда есть возможность уменьшить размерность матрицы покрытия, выделив интервалы, входящие в любую безызбыточную ДНФ. Если некоторый элемент булева пространства $m_i \in M^1$ принадлежит лишь одному из максимальных интервалов $U \subseteq M^1$, то очевидно, что любое кратчайшее покрытие множества M^1 максимальными интервалами содержит этот интервал. Элемент m_i в этом случае назовем, как это было при рассмотрении карты Карно, *определяющим элементом*, а интервал U – *обязательным интервалом*.

Чтобы определить, является ли некоторый элемент m_i определяющим, достаточно найти в M^1 все соседние с ним элементы, а затем построить содержащий их *минимальный поглощающий интервал*.

Минимальный поглощающий интервал U для элементов $m_1, m_2, \ldots, m_k$ булева пространства M представляется вектором $\boldsymbol{u}$, который получается следующим образом: если i-я компонента во всех векторах $m_1, m_2, \ldots, m_k$ имеет значение 0, то вектор $\boldsymbol{u}$ имеет в этой компоненте 0, если значение 1, то и вектор $\boldsymbol{u}$ имеет значение 1. Если i-я компонента имеет различные значения в этих векторах, то i-я компонента вектора $\boldsymbol{u}$ имеет значение «–».

Например, для элементов булева пространства (0 0 1 0 1 0), (0 0 1 0 1 1) и (0 0 0 0 1 1) минимальный поглощающий интервал представляется вектором (0 0 – 0 1 –).

Если все элементы полученного таким образом интервала U принадлежат M^1, то он является максимальным в M^1 и притом обязательным, а m_i является определяющим элементом. В противном случае U не содержится целиком в M^1, а m_i не является определяющим ни для какого интервала.

Чтобы определить, содержится ли интервал U во множестве M^1, достаточно для матрицы, представляющей множество M^1, построить минор, определяемый столбцами, где вектор $\boldsymbol{u}$ имеет значение «–», и строками, не ортогональными вектору $\boldsymbol{u}$. Число строк в этом миноре не превышает 2^p, где p – число компонент вектора $\boldsymbol{u}$, имеющих значение «–». Очевидно, интервал U целиком содержится в M^1 тогда и только тогда, когда число строк в этом миноре равно 2^p.

Обратимся к описанному примеру. Элемент (0 0 0 0) не является определяющим. Действительно, минимальный поглощающий интервал для него и соседних с ним элементов (0 0 1 0) и (1 0 0 0) задается троичным вектором (– 0 – 0). Соответствующий минор матрицы, представляющей M^1, имеет вид

$$\begin{matrix} x_1 & x_3 \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \end{matrix},$$

где число строк меньше чем $4 = 2^2$.

Аналогичным образом устанавливается, что элементы (0 0 1 0), (1 0 0 0), (0 0 1 1) и (1 0 0 1) также не являются определяющими. Что касается элемента (0 1 1 1), то для него и соседних с ним элементов (0 0 1 1) и (1 1 1 1) соответствующим троичным вектором является (– – 1 1) и соответствующим минором –

$$\begin{matrix} x_1 & x_2 \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \end{matrix},$$

число строк которого равно 4. Следовательно, элемент (0 1 1 1) является определяющим, а интервал, представляемый вектором (– – 1 1), – обязательным.

Этот интервал включается в решение, и все покрываемые им элементы исключаются из рассмотрения. Очевидно, что если среди них имеется какой-либо определяющий элемент, то он определяет тот же самый интервал. Затем отыскиваются интервалы, содержащие непокрытые элементы. В подразд. 8.2 будет рассмотрен метод Блейка – Порецкого, использующий для получения сокращенной ДНФ операции обобщенного склеивания и простого поглощения. Этот же прием можно использовать для получения интервалов, содержащих непокрытые элементы. Применяется операция обобщенного склеивания векторов, представляющих полученные обязательные интервалы, с векторами, представляющими непокрытые элементы. Далее представлена последовательность матриц, демонстрирующая соответствующий процесс.

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{matrix}, \quad \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & - \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{matrix}, \quad \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & - & 0 \\ 0 & 0 & 1 & - \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{matrix},$$

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & - & 0 \\ 0 & 0 & 1 & - \\ - & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{matrix}, \quad \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & - & 0 \\ 0 & 0 & 1 & - \\ - & 0 & 0 & 0 \\ 1 & 0 & 0 & - \end{bmatrix} \end{matrix}, \quad \begin{matrix} x_1 & x_2 & x_3 & x_4 \\ \begin{bmatrix} - & - & 1 & 1 \\ 0 & 0 & - & 0 \\ 0 & 0 & 1 & - \\ - & 0 & 0 & 0 \\ 1 & 0 & 0 & - \\ 1 & 0 & - & 1 \end{bmatrix} \end{matrix}.$$

Последняя матрица представляет сокращенную ДНФ. Далее решается задача покрытия, представляемая матрицей

$$\begin{array}{c} a_1\ a_2\ a_3\ a_5 \\ \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array}\right] \begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{array} \end{array},$$

где a_1,a_2,a_3,a_5 – векторы (0 0 0 0), (0 0 1 0), (1 0 0 0), (1 0 0 1) и B_1, B_2, B_3, B_4, B_5 – интервалы, представляемые векторами (0 0 – 0), (– 0 0 0), (0 0 1 –), (1 0 0 –), (1 0 – 1) соответственно.

Дальнейший ход решения не отличается от предыдущего, и в итоге получается тот же результат: $\bar{x}_1\ \bar{x}_2\ \bar{x}_4 \vee x_1\ \bar{x}_2\ \bar{x}_3 \vee x_3\, x_4$.

8.2. Метод Блейка – Порецкого

Метод Квайна – МакКласки, требующий представление исходной булевой функции в совершенной ДНФ, может оказаться не очень удобным, если булева функция задана в произвольной ДНФ, а ее совершенная ДНФ является довольно громоздкой. Например, совершенная ДНФ функции, заданной произвольной ДНФ $x_1\, x_2\, x_3\, x_5 \vee x_2\, x_3\, x_4\, x_5 \vee \bar{x}_1$, имеет 18 элементарных конъюнкций. В то же время минимальную ДНФ легко получить, применив операции обобщенного склеивания и поглощения:

$$x_1\, x_2\, x_3\, x_5 \vee x_2\, x_3\, x_4\, x_5 \vee \bar{x}_1 = x_1\, x_2\, x_3\, x_5 \vee x_2\, x_3\, x_4\, x_5 \vee \bar{x}_1 \vee x_2\, x_3\, x_5 = \bar{x}_1 \vee x_2\, x_3\, x_5.$$

Метод Блейка – Порецкого основан на применении операций обобщенного склеивания и простого поглощения, определяемых формулами

$$A\, x \vee B\, \bar{x} = A\, x \vee B\, \bar{x} \vee A\, B \quad \text{и} \quad A \vee A\, B = A.$$

Чтобы получить сокращенную ДНФ, необходимо для всех пар смежных элементарных конъюнкций получить продукты обобщенного склеивания и проверить, не поглощаются ли они другими конъюнкциями, входящими в ДНФ, и не поглощают ли они сами некоторые конъюнкции в ДНФ. Поглощаемые конъюнкции удаляются. Вновь получаемые конъюнкции также участвуют в операциях обобщенного склеивания. Процесс заканчивается, когда не удается ввести в ДНФ новую конъюнкцию. В результате описанного процесса получаются все простые импликанты, т. е. получается сокращенная ДНФ.

Если исходная ДНФ задана в виде троичной матрицы, то рекомендуется организовать процесс получения сокращенной ДНФ (троичной матрицы, где каждая строка представляет простую импликанту) следующим образом. Строки

исходной матрицы просматриваются сверху вниз, и для очередной строки отыскиваются смежные с ней строки, расположенные сверху от нее. Получаемые продукты обобщенного склеивания добавляются в матрицу снизу. Поглощаемые строки удаляются. Процесс заканчивается на самой нижней строке, когда результаты обобщенного склеивания со смежными с ней строками не дают новых строк для матрицы.

Пусть булева функция задана следующей троичной матрицей:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & \\ \left[\begin{matrix} 1 & 1 & - & 0 & 0 \\ 1 & 0 & - & 1 & 1 \\ 0 & 0 & 0 & - & 1 \\ 0 & - & 1 & 0 & - \\ - & 1 & 1 & 1 & 1 \\ 1 & - & 1 & 0 & 0 \\ 1 & 1 & - & 0 & 1 \end{matrix}\right. & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \left.7\right] \end{matrix} \end{matrix}.$$

В этой матрице находим следующие пары смежных строк: (2, 3), (1, 4), (3, 4), (2, 5), (4, 5). Выполняя над ними операцию обобщенного склеивания, получим строки

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & \\ \left[\begin{matrix} - & 0 & 0 & 1 & 1 \\ - & 1 & 1 & 0 & 0 \\ 0 & 0 & - & 0 & 1 \\ 1 & - & 1 & 1 & 1 \\ 0 & 1 & 1 & - & 1 \end{matrix}\right. & \begin{matrix} 8(2,3) \\ 9(1,4) \\ 10(3,4) \\ 11(2,5) \\ \left.12(4,5)\right] \end{matrix} \end{matrix},$$

которые не поглощаются заданными строками и сами не поглощают другие строки. Справа приведена нумерация строк, которая продолжает исходную нумерацию, а в скобках приведены номера склеиваемых строк.

Дальнейший поиск дает пары смежных строк (4, 6), (1, 7), (4, 7), (5, 7), (6, 7), для которых получим строки

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & \\ \left[\begin{matrix} - & - & 1 & 0 & 0 \\ 1 & 1 & - & 0 & - \\ - & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & - & 1 \\ 1 & 1 & - & 0 & - \end{matrix}\right. & \begin{matrix} 13(4,6) \\ 14(1,7) \\ 15(4,7) \\ 16(5,7) \\ \left.17(6,7)\right] \end{matrix} \end{matrix}.$$

Строка 13 поглощает строки 6 и 9, а строка 17 – строки 1 и 7. Строка 17 совпадает со строкой 14. Поэтому строки 1, 6, 7, 9 и 17 удаляются. Далее получаются следующие непоглощаемые строки:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ \left[\begin{matrix} - & 1 & 1 & 0 & - \\ - & 1 & 1 & - & 1 \end{matrix}\right. & & & & \left.\begin{matrix} 18(4{,}14) \\ 19(5{,}15) \end{matrix}\right. \end{matrix}.$$

Последняя строка поглощает строки 5, 12, 15 и 16. Окончательно получаем следующую матрицу (с естественной нумерацией строк), представляющую сокращенную ДНФ:

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ \left[\begin{matrix} 1 & 0 & - & 1 & 1 \\ 0 & 0 & 0 & - & 1 \\ 0 & - & 1 & 0 & - \\ - & 0 & 0 & 1 & 1 \\ 0 & 0 & - & 0 & 1 \\ 1 & - & 1 & 1 & 1 \\ - & - & 1 & 0 & 0 \\ 1 & 1 & - & 0 & - \\ - & 1 & 1 & 0 & - \\ - & 1 & 1 & - & 1 \end{matrix}\right. & & & & \left.\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{matrix}\right. \end{matrix}.$$

Второй этап процесса минимизации можно выполнить так же, как это делается в методе Квайна – МакКласки, т. е. свести его к решению задачи покрытия, разбив полученные интервалы на булевы векторы – элементы множества M^1. Комбинаторный поиск при решении задачи покрытия будет значительно сокращен, если удастся выделить *ядро* – множество обязательных интервалов. При существовании ядра иногда удается выделить *антиядро* – множество интервалов, покрываемое ядром. Тогда элементы ядра включаются в решение, элементы антиядра исключаются из рассмотрения и оставшимися интервалами покрываются элементы множества M^1, не покрытые ядром.

Если максимальный интервал не принадлежит ядру, то после его удаления из всей совокупности максимальных интервалов оставшиеся интервалы будут покрывать множество M^1. Таким образом, задача поиска ядра сводится к нахождению избыточных конъюнкций в ДНФ.

В полученной матрице, представляющей сокращенную ДНФ, выделим ядро путем последовательной проверки всех строк на избыточность, как это делалось в подразд. 6.4. Для i-й строки строим минор $\boldsymbol{M}_i$, образованный строками, не ортогональными i-й строке, и столбцами, где она имеет элемент «–», и проверяем каждый из них на вырожденность. Миноры имеют следующий вид:

$$\boldsymbol{M}_1 = \begin{matrix} x_3 \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 4 \\ 6 \end{matrix}, \boldsymbol{M}_2 = \begin{matrix} x_4 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{matrix} \begin{matrix} 4 \\ 5 \end{matrix}, \boldsymbol{M}_3 = \begin{matrix} x_2\ x_5 \\ \begin{bmatrix} 0\ 1 \\ -\ 0 \\ 1\ - \\ 1\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 5 \\ 7 \\ 9 \\ 10 \end{matrix}, \boldsymbol{M}_4 = \begin{matrix} x_1 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{matrix} \begin{matrix} 1 \\ 2 \end{matrix}, \boldsymbol{M}_5 = \begin{matrix} x_3 \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 2, \\ 3 \end{matrix}$$

$$\boldsymbol{M}_6 = \begin{matrix} x_2 \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 1 \\ 10 \end{matrix}, \boldsymbol{M}_7 = \begin{matrix} x_1\ x_2 \\ \begin{bmatrix} 0\ - \\ 1\ 1 \\ -\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 3 \\ 8, \\ 9 \end{matrix} \boldsymbol{M}_8 = \begin{matrix} x_3\ x_5 \\ \begin{bmatrix} 1\ 0 \\ 1\ - \\ 1\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 7 \\ 9 \\ 10 \end{matrix}, \boldsymbol{M}_9 = \begin{matrix} x_1\ x_5 \\ \begin{bmatrix} 0\ - \\ -\ 0 \\ 1\ - \\ -\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 3 \\ 7 \\ 8 \\ 10 \end{matrix}, \boldsymbol{M}_{10} = \begin{matrix} x_1\ x_4 \\ \begin{bmatrix} 0\ 0 \\ 1\ 1 \\ 1\ 0 \\ -\ 0 \end{bmatrix} \end{matrix} \begin{matrix} 3 \\ 6. \\ 8 \\ 9 \end{matrix}$$

Невырожденными матрицами являются $\boldsymbol{M}_7$, $\boldsymbol{M}_8$ и $\boldsymbol{M}_{10}$ с ортогональными их строкам векторами (1 0), (0 –) и (0 1). Следовательно, строки 7, 8 и 10 составляют ядро, и соответствующие интервалы вносятся в решение. Для поиска антиядра можно использовать те же матрицы (кроме $\boldsymbol{M}_7$, $\boldsymbol{M}_8$ и $\boldsymbol{M}_{10}$), удалив из них строки, не принадлежащие ядру. При этом, если матрица оказалась невырожденной или пустой, это означает, что соответствующий интервал не покрывается ядром. В данном примере антиядро состоит только из одного интервала, представленного строкой 9. Кратчайшее покрытие векторов (1 0 0 1 1), (1 0 1 1 1), (0 0 0 0 1), (0 0 0 1 1) и (0 0 1 0 1) из множества M^1, не покрытых ядром, составляют строки 1, 2 и 3. Минимальная ДНФ представляется матрицей

$$\begin{matrix} x_1\ x_2\ x_3\ x_4\ x_5 \\ \begin{bmatrix} 1\ 0\ -\ 1\ 1 \\ 0\ 0\ 0\ -\ 1 \\ 0\ -\ 1\ 0\ - \\ -\ -\ 1\ 0\ 0 \\ 1\ 1\ -\ 0\ - \\ -\ 1\ 1\ -\ 1 \end{bmatrix} \end{matrix} \begin{matrix} 1 \\ 2 \\ 3, \\ 7 \\ 8 \\ 10 \end{matrix}$$

где сохранена нумерация строк матрицы, представляющей сокращенную ДНФ.

9. ЭЛЕМЕНТЫ МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Математическая логика представляет собой формальный математический аппарат, изучающий различные способы логических рассуждений. Простейшей из формальных логических теорий называют *алгебру высказываний*. Исчисление высказываний – следующая ступень в иерархии формальных теорий. Еще выше стоят алгебра предикатов и исчисление предикатов.

9.1. Алгебра высказываний

Высказывание – некоторое утверждение, относительно которого нас интересует только его истинность или ложность. Значения истинности и ложности могут обозначаться буквами И и Л, *T* (truth) и *F* (false) или цифрами 1 и 0.

Предложения, представляющие вопрос, приказ или восклицание либо внутренне противоречивое утверждение, не являются высказываниями.

Высказывание называют простым (элементарным), если его истинность не зависит от истинности других высказываний (в утверждении не используются связки). Высказывание называют сложным (составным), если оно зависит от истинности других высказываний (используются связки).

В алгебре высказываний объектом исследования является множество высказываний, а предметом – логические операции, каждую из которых можно рассматривать как сложное высказывание. Данные операции ($\neg$, $\wedge$, $\vee$, $\oplus$, $\rightarrow$, $\leftrightarrow$) определяются таблицами истинности (см. табл. 5.3).

В дальнейшем каждое простое высказывание можно связать с некоторой двоичной переменной: $x = 1$, если простое высказывание истинно, и $x = 0$, если простое высказывание ложно. Поэтому сложное высказывание можно связать с некоторой булевой функцией: $f(x_1, x_2, \ldots, x_n) = 1$, если сложное высказывание, состоящее из n простых высказываний, истинно, и $f(x_1, x_2, \ldots, x_n) = 0$, если оно ложно.

Используя подобную связь, можно сказать, что алгебра высказываний есть одна из интерпретаций алгебры логических функций.

В дальнейшем будем говорить, что формула истинна или ложна в зависимости от того, истинно или ложно соответствующее ей высказывание.

Формула может быть истинной при одном наборе значений переменных и ложной при другом наборе. Формула, которая является истинной хотя бы при одном наборе значений переменных, называется *выполнимой*. Формула, ложная при всех наборах значений переменных, называется *противоречием (тождественно ложной)*, формула, истинная при всех наборах значений переменных, – *тавтологией (тождественно истинной).*

Установить тип формулы можно с помощью таблиц истинности. При большом числе переменных x_i (при большом числе простых высказываний) таблицы истинности функций, соответствующих этим формулам, очень громоздки. Установить тип формулы (выполнима, тавтология, противоречие) удобно с помощью нормальных форм.

Для этого необходимо:

а) заменить все логические операции булевыми операциями, используя равносильные формулы;

б) заменить знак отрицания, относящийся ко всему выражению, на знаки, относящиеся к отдельным переменным, используя закон де Моргана;

в) избавиться от знаков двойного отрицания;

г) применить в случае необходимости закон дистрибутивности и формулы поглощения.

Формула алгебры высказываний является *тождественно истинной*, когда каждый множитель ее КНФ содержит пару слагаемых, одно из которых является элементарным высказыванием, а другое – его отрицанием.

Формула алгебры высказываний является *тождественно ложной*, когда каждое слагаемое её ДНФ содержит пару сомножителей, один из которых является элементарным высказыванием, а другой – его отрицанием.

Формулы A и B находятся в отношении формальной импликации, точнее A имплицирует B, если формула B истинна на всех наборах переменных, на которых истинна формула A. В таких случаях говорим, что формула B логически следует из формулы A.

Формулы A и B *равносильны* (логически эквивалентны), если любая из них следует из другой. Очевидно, что таблицы истинности равносильных формул совпадают.

Рассмотрим основные *тавтологии* использования высказываний:

1) Закон тождества: $a \to a$. Всякое высказывание логично следует из самого себя.

2) Закон противоречия: $\neg(a \wedge \neg a)$. Всякое высказывание не может быть одновременно истинным и ложным.

3) Закон исключения третьего: $a \vee \neg a$. Для всякого высказывания истинно либо оно само, либо его отрицание.

4) Закон двойного отрицания: $\neg\neg a \leftrightarrow a$. Отрицание отрицания любого высказывания равносильно самому высказыванию.

5) Закон «истинно из чего угодно»: $a \to (b \to a)$. Если а – истинное высказывание, то формула $b \to a$ истинна.

6) Закон «из ложного что угодно»: $\neg a \to (a \to b)$. Если a – ложное высказывание, то из a следует все, что угодно.

7) Закон *modus ponens*: $(a \wedge (a \to b)) \to b\,(a \wedge (a \to b)) \to b$. Если истинно то, что из a следует b, и a истинно, то b истинно.

8) Закон *modus tollens*: $((a \to b) \wedge \neg b) \to \neg a$. Если из a следует b, а b ложно, то a тоже ложно.

9) Закон силлогизма: $((a \to b) \wedge (b \to c)) \to (a \to c)$. Если из a следует b, а из b следует c, то из a следует c.

9.2. Логические отношения и проверка правильности рассуждений

Имеем два высказывания P и Q. Рассмотрим типы логических отношений:

1) Отношение следствия ($P \to Q$). Говорим, что из P следует Q, если Q истинно всякий раз, когда истинно P. Q называют следствием P.

Заметим, что между отношением следствия и импликацией существует тесная связь, но это не одно и то же. Импликация – сложное высказывание, со-

ставленное из двух данных, а следствие – отношение между двумя высказываниями. Импликация выражает отношение следствия только тогда, когда таблица истинности импликации содержит одни единицы. Отметим, что высказывания, связанные с импликацией, при отсутствии смысловой связи между посылкой и заключением могут звучать парадоксально. Например, высказывание «Если я не приду на лекцию, то река впадает в Белое море» звучит парадоксально. Между посылкой и заключением в подобных случаях не существует отношения следствия.

2) Два высказывания P и Q эквивалентны, если таблица истинности содержит только единицы. Импликация $P = A \to B$ и контрапозиция импликации $Q = \overline{B} \to \overline{A}$ эквивалентны. Эти формулы в рассуждении заменяют друг друга.

3) Два высказывания P и Q называются несовместимыми, если не существует логической возможности, при которой оба высказывания были бы одновременно истинными, т. е. при истинном значении одного из них другое обязательно ложно.

Чтобы установить совместимость высказываний, необходимо построить их таблицы истинности. Если найдется хотя бы одна строка, в которой все высказывания принимают значение истинно, то данные высказывания будут совместимы, и несовместимы в противном случае.

Рассуждение есть утверждение того, что некоторое высказывание (заключение) следует из других высказываний (посылок). Рассуждение будет правильным, если из конъюнкции посылок следует заключение, т. е. между конъюнкцией посылок и заключением установлено отношение следствия. В этом случае импликация должна быть тавтологией.

Правильность рассуждения можно установить, построив таблицу истинности высказывания $P_1 \wedge P_2 \wedge \ldots \wedge P_n \to Q$ и убедившись в том, что оно тождественно истинно.

Правильность рассуждения можно установить и методом «от противного». Этот метод заключается в том, что, полагая заключение Q ложным, а некоторые посылки P_i истинными, проверяем, найдется ли хотя бы одна посылка, принимающая значение ложно. Если да, то рассуждение правильно.

9.3. Решение логических задач с помощью уравнений

Логическую задачу можно решить, составив и решив логическое уравнение. Для этого необходимо выполнить следующие действия:

1) ввести булевы переменные $x_1, x_2, \ldots, x_n$, соответствующие простым высказываниям;

2) записать условие задачи в виде уравнений $f_i(x_1, x_2, \ldots, x_n) = 1$, $i = 1, \ldots, m$, где $f_i(x_1, x_2, \ldots, x_n)$ – логическая функция;

3) свести систему уравнений к уравнению $\bigcap_{i=1}^{m} f_i(x_1, x_2, \ldots, x_n) = 1$, множество корней которого совпадает с множеством корней системы из предыдущего пункта;

4) привести левую часть характеристического уравнения к ДНФ и решить его: приравнять каждое слагаемое ДНФ (СДНФ), независимо от других, к единице, извлечь из уравнения значения переменных. Каждый их набор является решением задачи. Если после упрощения в ДНФ осталось только одно слагаемое, задача имеет единственное решение. В случае когда в левой части уравнения все слагаемые уничтожены, задача не имеет решения.

Рассмотрим применение этого алгоритма на примере одной из логических задач.

Задача Р. М. Смаллиана «Принцесса и тигр»[16].

В некотором царстве правил король. Однажды он предложил узнику отгадать, в какой комнате находится принцесса, а в какой тигр. Узнику было объявлено, что в каждой комнате находится либо принцесса, либо тигр, однако может оказаться, что сразу в обеих комнатах будет обнаружено по тигру или по принцессе.

На табличках, прикрепленных к двери каждой из комнат, были надписи, представленные в табл. 9.1.

Таблица 9.1

Задание исходных условий

В этой комнате находится принцесса, а в другой комнате сидит тигр. $f_1 = x_1\,\bar{x}_2\,\bar{y}_1 y_2$	В одной из этих комнат находится принцесса, кроме того, в одной из этих комнат сидит тигр. $f_2 = (x_1\,\bar{x}_2 \vee \bar{x}_1 x_2)(y_1\,\bar{y}_2 \vee \bar{y}_1 y_2)$

Король сообщил узнику, что на одной из таблиц написана правда, на другой – ложь. Какую дверь следует открыть узнику, если он предпочитает принцессу тигру?

Решение:

1) Формулируем простые высказывания:

– x_i – «принцесса находится в комнате i»: $i = 1, 2$;

– y_i – «тигр сидит в комнате i»: $i = 1, 2$.

2) Формулируем сложные высказывания, соответствующие условию задачи:

$$f_1 = x_1\,\bar{x}_2\,\bar{y}_1\,y_2, \quad f_2 = (x_1\,\bar{x}_2 \vee \bar{x}_1\,x_2)(y_1\,\bar{y}_2 \vee \bar{y}_1\,y_2),$$

$$f_0 = x_1\,x_2\,\bar{y}_1\,\bar{y}_2 \vee \bar{x}_1\,\bar{x}_2\,y_1\,y_2 \vee x_1\,\bar{x}_2\,\bar{y}_1\,y_2 \vee \bar{x}_1\,x_2\,y_1\,\bar{y}_2.$$

Получаем систему уравнений:

$$f_0 = 1;$$

$$f_1\ \bar{f}_2 \vee\ \bar{f}_1 f_2 = 1.$$

3) Составляем уравнение:

$$(f_1\ \bar{f}_2 \vee\ \bar{f}_1 f_2)\, f_0 = 1.$$

4) Приводим левую часть характеристического уравнения к ДНФ. Для выполнения операций над булевыми функциями используем карты Карно (рис. 9.1 и 9.2).

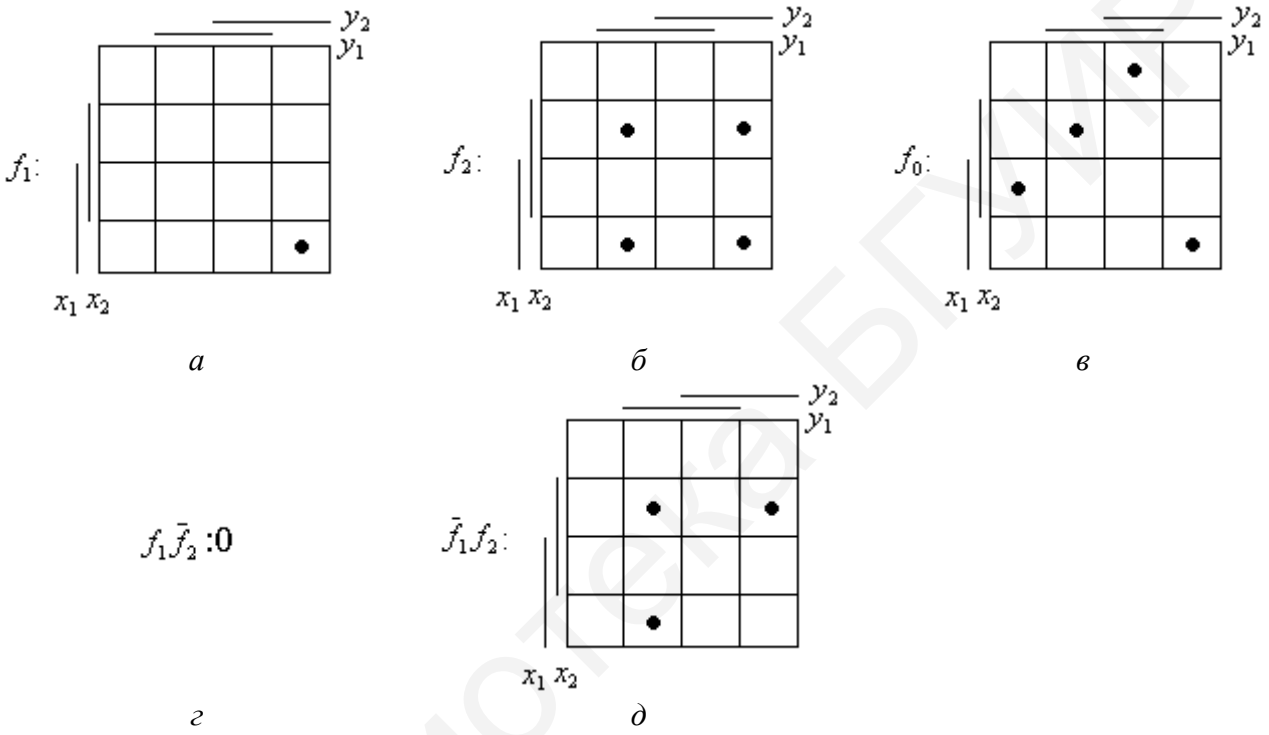


Рис. 9.1. Задание булевых функций с помощью карт Карно:
а – функции f_1; *б* – функции f_2; *в* – функции f_0; *г* – функции $f_1\bar{f}_2$; *д* – функции $\bar{f}_1 f_2$

Из последней матрицы видно, что ДНФ содержит только одно слагаемое.

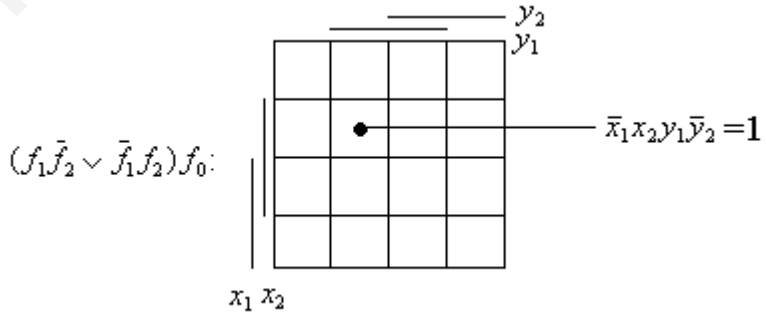


Рис. 9.2. Решение задачи с помощью карты Карно

Решением уравнения $\bar{x}_1 x_2 y_1 \bar{y}_2 = 1$ является набор 0110, т. е. принцесса находится в комнате 2, а тигр в комнате 1.

9.4. Алгебра предикатов

Логика предикатов представляет собой развитие логики высказываний. С помощью формул логики высказываний, например, алгебры логики, можно описать и исследовать структуру сложных высказываний, установить их истинность или ложность в зависимости от истинности или ложности входящих в нее простых высказываний. Для описания внутренней логической структуры простых высказываний (т. е. высказываний, не содержащих связок) используется понятие предиката.

Предикат – это повествовательное предложение, содержащее предметные переменные, определенные на соответствующих множествах. При замене переменных конкретными значениями (элементами) данных множеств предложение обращается в высказывание, т. е. принимает значение «истинно» или «ложно».

N-местный предикат – это функция $P(x_1, x_2, ..., x_n)$ от n переменных, принимающих значения из некоторых заданных предметных областей, так что $x_1 \in M_1$, $x_2 \in M_2$, ..., $x_n \in M_n$, а функция P принимает два логических значения – «истинно» или «ложно» ({И, Л}, {1, 0}). Таким образом, предикат $P(x_1, x_2, ..., x_n)$ является функцией типа $P : M_1 \times M_2 \times ... \times M_n \rightarrow B$, где множества $M_1, M_2, ..., M_n$ называются предметными областями предиката; $x_1, x_2, ..., x_n$ – предметные переменные предиката; B – двухэлементное множество {И, Л} или {1,0}.

В качестве примера рассмотрим три высказывания:

A – «Рубль – валюта России»;

B – «Доллар – валюта России»;

C – «Доллар – валюта США».

Высказывания A и C – истинны, а B – ложно. Если вместо конкретных наименований валюты в выражениях A, B (и, может быть, аналогичных им) подставить предметную переменную x и определить ее на множестве наименований денежных единиц {рубль, доллар, фунт стерлингов, ..., марка}, то получим одноместный предикат $P(x)$ – «x – валюта России».

Если в выражениях A, B, C (или аналогичных им) вместо конкретных наименований валюты и государства подставить соответственно переменные x и y, где $y \in$ {Россия, США, Англия, ..., Германия}, получим двухместный предикат $P(x, y)$ – «x – валюта y». Общим для этих предикатов является то, что, приписав значения входящим в них переменным из соответствующих областей определения, получим высказывания, обладающие свойством «истинно» или «ложно».

С помощью логических связок (и скобок) предикаты могут объединяться в разнообразные логические формулы – предикатные формулы. Исследование предикатных формул и способов установления их истинности является основным предметом логики предикатов. Логика предикатов вместе с входящей в нее

логикой высказываний является основой логического языка математики. С ее помощью удается формализовать и точно исследовать основные методы построения математических теорий. Логика предикатов является важным средством построения развитых логических языков и формальных систем (формальных теорий).

Логика предикатов, как и логика высказываний, может быть построена в виде алгебры логики предикатов и исчисления предикатов. Здесь, как и в случае логики высказываний, для знакомства с основными понятиями логики предикатов воспользуемся языком алгебры.

Выражение $P(a_1, a_2, ..., a_n)$ будем понимать как высказывание «$P(a_1, a_2, ..., a_n) = 1$» или «$P(a_1, a_2, ..., a_n)$ истинно», а выражение $P(x_1, x_2, ..., x_n)$ – как переменное высказывание, истинность которого определяется подстановкой элементов множества M вместо переменных $x_1, x_2, ..., x_n$. При этом будем также называть $P(x_1, x_2, ..., x_n)$ логической (двоичной) переменной, в отличие от $x_1, x_2, ..., x_n$ – предметных (нелогических) переменных.

Соответствия между предикатами, отношениями и функциями:

1) Для любых M и n существует взаимно однозначное соответствие между n-местными отношениями $R \subseteq M^n$ и n-местными предикатами $P(x_1, x_2, ..., x_n)$, $P : M^n \to B$:

– каждому n-арному отношению R соответствует предикат $P(x_1, x_2, ..., x_n)$, такой, что $P(a_1, a_2, ..., a_n) = 1$, если и только если $(a_1, a_2, ..., a_n) \in R$;

– всякий предикат $P(x_1, x_2, ..., x_n)$ определяет отношение R, такое, что $(a_1, a_2, ..., a_n) \in R$, если и только если $P(a_1, a_2, ..., a_n) = 1$.

При этом R задает область истинности предиката P.

2) Всякой функции $f(x_1, x_2, ..., x_n)$, $f : M^n \to M$, соответствует предикат $P(x_1, x_2, ..., x_n, x_{n+1})$, $P : M^{n+1} \to B$, такой, что $P(a_1, a_2, ..., a_n, a_{n+1}) = 1$, если и только если $f(a_1, a_2, ..., a_n) = a_{n+1}$.

Понятие предиката шире понятия функции, поэтому обратное соответствие (от $(n+1)$-местного предиката к n-местной функции) возможно не всегда, а только для таких предикатов P', для которых выполняется условие (связанное с требованием однозначности функции): если $P'(a_1, a_2, ..., a_n, a_{n+1}) = 1$, то для любого

$$a'_{n+1} \neq a_{n+1} P'(a_1, a_2, ..., a_n, a_{n+1}) = 0. \qquad (9.1)$$

Аналогичное соответствие (взаимно однозначное) имеется между подмножеством отношений $\{R'\} \subset \{R\}$ и множеством функций. Для этого класса отношений выполняется аналогичное условие: если $(a_1, a_2, ..., a_n, a_{n+1}) \in R'$, то для любого

$$a'_{n+1} \neq a_{n+1} (a_1, a_2, ..., a_n, a_{n+1}) \notin R'. \qquad (9.2)$$

Из предикатов как высказываний можно образовывать составные высказывания – формулы логики предикатов.

9.5. Кванторы

Важную роль в логических представлениях систем, процессов, явлений с использованием предикатов играют собственные связки логики предикатов: общности $\forall$ и существования $\exists$.

Пусть $P(x)$ – предикат, определенный на M, т. е. $x \in M$.

Высказывание «для всех x из $MP(x)$ истинно» обозначается как $\forall x\, P(x)$. Высказывание «существует такой x из M, что $P(x)$ истинно» обозначается как $\exists x\, P(x)$.

Переход от $P(x)$ к $\forall x\, P(x)$ или $\exists x\, P(x)$ называется связыванием переменной x, или навешиванием квантора на переменную x (или на предикат P), или квантификацией переменной x.

Переменная, на которую навешен квантор, называется связанной, несвязанная квантором переменная называется свободной.

Выражения $\forall x\, P(x)$ и $\exists x\, P(x)$ не зависят от x и при фиксированных P и M имеют вполне определенные значения, представляя вполне конкретные высказывания относительно всех x предметной области M.

Навешивать кванторы можно и на многоместные предикаты и в целом на любые логические выражения. Выражение, на которое навешивается квантор $\forall x$ или $\exists x$, называется областью действия квантора; все вхождения переменной x в это выражение являются связанными.

9.6. Эквивалентные соотношения. Префиксная нормальная форма

Формулы называются эквивалентными, если при любых подстановках констант они принимают одинаковые значения. В частности, все тождественно-истинные формулы (ТИ-формулы) и все тождественно-ложные формулы (ТЛ-формулы) эквивалентны.

Множество ТИ-формул логики предикатов входит в любую теорию, исследование этого множества – важная цель логики предикатов. При этом выделяются две проблемы:

1) получение ТИ-формул (проблема построения порождающей процедуры для множества ТИ-формул);

2) проверка формулы на истинность (проблеморазрешающая процедура).

В отличие от логики (алгебры) высказываний, где есть стандартная разрешающая процедура (вычисление формул на наборах значений переменных), с помощью которой очевидна организация порождающей процедуры построения множества ТИ-формул, в логике предикатов прямой перебор всех значений переменных может быть невозможен, если предметные переменные имеют бесконечные области определения.

Поэтому в логике предикатов используются различные косвенные приемы, в том числе эквивалентные соотношения, позволяющие выполнить кор-

ректные преобразования предикатных формул. В логике (алгебре) предикатов справедливы все эквивалентные соотношения логики (алгебры) высказываний, а также собственные эквивалентные соотношения, включающие связки $\forall$ и $\exists$ (в формулах (9.3)–(9.12) под Y будем понимать переменное высказывание или формулу, не содержащую x):

$$\neg\exists x\, P(x) \sim \forall x\, \neg P(x); \tag{9.3}$$
$$\neg\forall x\, P(x) \sim \exists x\, \neg P(x); \tag{9.4}$$
$$\forall x\, (P_1(x) \wedge P_2(x)) \sim (\forall x P_1(x) \wedge \forall x P_2(x)); \tag{9.5}$$
$$\exists x\, (P_1(x) \vee P_2(x)) \sim (\exists x P_1(x) \vee \exists x\, P_2(x)); \tag{9.6}$$
$$\forall x\, \forall y P(x, y) \sim \forall y \forall x\, P(x, y); \tag{9.7}$$
$$\exists x\, \exists y\, P(x, y) \sim \exists y \exists x\, P(x, y); \tag{9.8}$$
$$\forall x\, (P(x) \wedge Y) \sim (\forall x P(x) \wedge Y); \tag{9.9}$$
$$\forall x\, (P(x) \vee Y) \sim (\forall x P(x) \vee Y); \tag{9.10}$$
$$\exists x\, (P(x) \wedge Y) \sim (\exists x P(x) \wedge Y); \tag{9.11}$$
$$\exists x\, (P(x) \vee Y) \sim (\exists x P(x) \vee Y). \tag{9.12}$$

Используя соотношения (9.3) и (9.4), можно выразить один квантор через другой. Соотношения (9.5) и (9.6) показывают дистрибутивность квантора общности $\forall x$ относительно конъюнкции $\wedge$ и квантора существования $\exists x$ относительно дизъюнкции $\vee$. Если в указанных выражениях поменять местами кванторы $\forall x$ и $\exists x$, то получим соотношения, верные лишь в одну сторону:

$$\exists x\, (P_1(x) \wedge P_2(x)) \rightarrow (\exists x\, P_1(x) \wedge \exists x\, P_2(x));$$
$$(\forall x\, P_1(x) \vee \forall x\, P_2(x)) \rightarrow \forall x\, (P_1(x) \vee P_2(x)).$$

Поэтому в таких случаях эквивалентных преобразований применяют переименование переменной x в одном из предикатов на новую переменную:

$$(\exists x\, P_1(x) \wedge \exists y\, P_2(y)) \sim \exists x\, \exists y\, (P_1(x) \wedge P_2(y));$$
$$(\forall x\, P_1(x) \vee \forall y\, P_2(y)) \sim \forall x\, \forall y\, (P_1(x) \vee P_2(y)).$$

Соотношения (9.7) и (9.8) отражают в некотором смысле коммутативность одноименных кванторов (возможность менять местами одноименные кванторы), что несправедливо для разноименных кванторов, например $\forall x\, \exists y\, P(x, y)$ и $\exists y\, \forall x\, P(x, y)$ не эквивалентны. Соотношения (9.9)–(9.12) позволяют формулу, не содержащую переменную x, выносить за пределы действия квантора, связывающего эту переменную. Указанных соотношений (9.3)–(9.12), а также эквивалентных соотношений логики (алгебры) высказываний достаточно для выполнения преобразований формул логики (алгебры) предикатов.

Как и в логике высказываний, в логике предикатов существуют эквивалентные нормальные формы представления любых предикатных формул, в том числе префиксная нормальная форма (ПНФ).

ПНФ называется формула, имеющая вид

$$Q_1x_1\ Q_2x_2 \ldots\ Q_nx_n\ F,$$

где Q_1x_1, Q_2x_2, …,Q_nx_n – кванторы; F – формула, не имеющая кванторов, с операциями $\{\wedge, \vee, \neg\}$.

В логике предикатов для любой формулы существует эквивалентная ей префиксная нормальная форма.

Процедура получения ПНФ:

1. Используя формулы

$$P_1 \sim P_2 = (P_1 \to P_2) \wedge (P_2 \to P_1); \quad (9.13)$$

$$P_1 \to P_2 = \neg P_1 \vee P_2, \quad (9.14)$$

заменить операции $\to$, $\sim$ на $\wedge$, $\vee$, $\neg$.

2. Воспользовавшись выражениями (9.3) и (9.4), а также правилом двойного отрицания и правилами де Моргана,

$$\neg\neg P = P; \quad (9.15)$$

$$\neg(P_1 \vee P_2) = \neg P_1 \wedge \neg P_2; \quad (9.16)$$

$$\neg(P_1 \wedge P_2) = \neg P_1 \vee \neg P_2, \quad (9.17)$$

представить предикатную формулу таким образом, чтобы символы отрицания были расположены непосредственно перед символами предикатов.

3. Для формул, содержащих подформулы вида

$$\forall x\ P_1(x) \vee \forall x\ P_2(x),\ \ \exists x\ P_1(x) \wedge \exists x\ P_2(x),$$

ввести новые переменные, позволяющие использовать соотношения (9.9)–(9.12).

4. С помощью формул (9.5)–(9.12) получить формулы в виде ПНФ.

10. ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ

10.1. Интуитивное понятие об алгоритме

Крупнейшим достижением науки XX века является теория алгоритмов – новая математическая дисциплина.

Слово «алгоритм» происходит от имени узбекского ученого-математика Хорезми (Ал-Хорезми), который в IX веке н. э. разработал правила четырех арифметических действий над числами в десятичной системе счисления.

В 30-е годы XX века понятие алгоритма стало объектом математического изучения, а с появлением ЭВМ получило широкую известность. Разработка алгоритмов является необходимым этапом автоматизации.

Алгоритм – это правило, сформированное на некотором языке и определяющее процесс переработки допустимых исходных данных в искомые результаты. Допустимыми исходными данными для этого правила являются предложения языка исходных данных.

Под алгоритмом понимается процесс последовательного построения (вычисления) величин, протекающий в дискретном времени так, что в каждый следующий момент времени система величин получается по определенному закону из системы величин, имевшихся в предыдущий момент.

Правила описания алгоритмов:

– понятность для исполнителя;

– массовость (т. е. допустимость для него всех предложений языка исходных данных);

– определенность (все шаги алгоритма детерминированы и четко определены);

– результативность.

Говорят, что алгоритм применим к допустимому исходному данному, если с его помощью, отправляясь от этого исходного данного, можно получить искомый результат. При этом алгоритмический процесс оканчивается после конечного числа шагов, если на каждом шаге нет препятствий для его выполнения.

Поскольку требование завершения алгоритмического процесса за конечное число шагов не учитывает реальных возможностей, связанных с затратами времени и ресурсов, то говорят, что при этом алгоритм *потенциально выполним.*

Основные требования, предъявляемые к алгоритмам:

– алгоритм имеет дело с *данными* и выдает *результат,* т. е. алгоритм имеет *вход* и *выход*;

– данные для своего размещения требуют памяти; единицы измерения объема данных и памяти согласованы;

– алгоритм состоит из отдельных элементарных шагов или действий (дискретность), причем множество этих шагов конечно;

– последовательность шагов алгоритма детерминирована;

– алгоритм применяется не к одной задачи, а к классу задач (массовость).

Связь между шагами можно представить в виде ориентированного графа, называемого блок-схемой алгоритма. В этом орграфе вершины соответствуют шагам, а дуги – переходам между шагами. Вершины блок-схем могут быть двух видов:

1) вершины, из которых выходит одна дуга, так называемые операторы;

2) вершины, их которых выходят две дуги, так называемые *предикаты* или *логические условия*.

Важной особенность блок-схем является то, что связи, которые она описывает, не зависят от того, являются ли шаги элементарными или представляют собой самостоятельные алгоритмы.

Декомпозиция алгоритма – способ разбиения алгоритма на блоки – широко используется в практике программирования. С помощью блок-схем можно несколько алгоритмов, рассматриваемых как блоки, связать в один большой алгоритм.

Как и в повседневной жизни, роль алгоритмов в науке и технике очень велика. Алгоритм – это «братство» науки и техники. Каждый новый алгоритм немедленно вписан в золотой фонд науки. При этом интересны как новые алгоритмы, так и алгоритмы для решения вновь поставленных проблем.

10.2. Три типа алгоритмических моделей

Если имеем дело с задачами, решения которых неизвестны и относительно которых имеются предположения, что они по сути своей не могут быть решены алгоритмическими методами, интуитивных представлений об алгоритме недостаточно. Доказать алгоритмическую неразрешенность задач на основе интуитивных представлений об алгоритме невозможно. Различный выбор исходных средств формализации приводит к моделям алгоритмов разного вида. Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм.

Еще в 30-е годы XX века были предприняты попытки формализовать это понятие и предложены различные модели алгоритмов [13]:

- рекурсивные функции;
- машины Тьюринга;
- алгоритмы Маркова.

Первый тип связывает понятие алгоритма с вычислениями и числовыми функциями. Наиболее развитая и изученная модель этого типа – рекурсивные функции – первый способ формализации понятия алгоритма.

Второй тип основан на представлении об алгоритме как о некотором детерминированном устройстве, способном выполнять в каждый отдельный момент лишь примитивные операции (машина Тьюринга).

Третий тип алгоритмических моделей – это преобразование слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т. е. замена части слова (подслова) другим словом (нормальный алгоритм Маркова, каноническая система Поста).

Тезис Чёрча: Класс задач, решаемых в любой из этих формальных моделей, и есть класс всех задач, которые могут быть решены интуитивно алгоритмическими методами.

Одна из причин, побудивших заняться теорией алгоритмов, – это подозрительность математиков в связи с накоплением упорно не поддающихся решению задач на нахождение алгоритмов, например:

1. *Задача о квадратуре круга*. Требуется найти алгоритм построения с помощью циркуля и линейки квадрата, равновеликого данному кругу.

2. *Задача трисекции угла*. Требуется найти алгоритм деления произвольного угла с помощью циркуля и линейки на три равные части.

3. *Задача удвоения куба*. Найти алгоритм, позволяющий на стороне любого куба с помощью циркуля и линейки построить сторону куба, объем которого вдвое больше объема заданного куба.

Невозможность решения этих задач строго доказана. Появилась необходимость выявления неразрешимых проблем. В теории алгоритмов есть раздел – разработка методов доказательства несуществований алгоритмов.

Вторая причина разработки теории алгоритмов – необходимость обоснования математики, поскольку появление антиномий привели к тому, что все в математике стало казаться неустойчивым. Многие математики стали искать выход или пути преодоления противоречий. Среди них выделялись те, кто усмотрел причину кризиса математики в том, что ряд математических объектов и методов является неконструктивным. Примером является актуально бесконечное множество. Расходуя ограниченное количество ресурсов на каждом шаге, имеющем фиксированную длительность, построить такое множество нереально и потенциально нельзя; нельзя проверить, обладают ли все элементы такого множества каким-либо свойством из-за ограниченной скорости проверки.

10.3. Машины Тьюринга как модели алгоритмов

В 1937 году английский математик Тьюринг опубликовал работу, в которой он, уточняя понятие алгоритма, прибегал к воображаемой вычислительной машине.

Машина должна перерабатывать какие-то объекты в исходные результаты. Этими объектами являются слова, построенные из символов-букв.

Машина состоит из бесконечной в обе стороны ленты, разбитой на ячейки, и рабочей головки, работает в дискретные моменты времени $t = 0, 1, 2, \ldots$. В каждый момент времени во всякой ячейке ленты записана одна буква из некоторого конечного алфавита $A = \{a_0, a_1, \ldots, a_n\}$, называемого внешним алгоритмом машины, а головка находится в одном из состояний конечного множества внутренних состояний $Q = \{q_0, q_1, \ldots, q_z\}$. Будем считать, что a_0 – пустой символ, т. е. в ячейке ничего не написано.

Основной частью машины является логический блок, который работает следующим образом. В каждый момент времени рабочая головка обозревает

одну ячейку ленты. В зависимости от того, что написано в ячейке и от внутреннего состояния головки, она заменяет символ в обозреваемой ячейке, переходит в новое состояние и сдвигается на одну ячейку влево или вправо, или остается на месте. Введем для обозначения этого движения символы d_{-1}, d_{+1}, d_0 соответственно. Таким образом, работа машины Тьюринга задается системой команд вида

$$\sum_T = \{q_j\, a_i \rightarrow q_k\, a_x\, d_y\}.$$

Все комбинации q_j и a_i для разных j и i и все реакции машины на них можно представить в виде функциональной таблицы с двумя входами q_j и a_i. Если головка в состоянии q_j читает символ a_i, то в следующий момент она запишет в эту ячейку символ a_x, перейдет в состояние q_k и в зависимости от того, каково d_y, головка сдвинется на одну ячейку влево, вправо или останется на месте (табл. 10.1).

Таблица 10.1

Задание исходных условий

	$q_1, q_2 \ldots$	q_j	q_z
a_0			
a_i		$q_k a_x d_y$	
a_n			

Примеры построения машин Тьюринга:

1. Построить машину Тьюринга, реализующую «сложение» чисел.

В машине Тьюринга все числа представляются в единичном коде, состоящем из X единиц. Поэтому сложить a и b значит переработать слова $1^a * 1^b$ в слово 1^{a+b}.

Зададим алфавит входных и выходных символов $A = \{\lambda, 1, \forall\}$, где λ – символ содержимого пустой ячейки, $\lambda = a_0$; $\forall$– символ разделения слов.

Пусть $a = 2$, $b = 3$. Тогда можно ленту машины Тьюринга представить следующим образом:

...	λ	1	1	∀	1	1	1	…

В начальный момент времени головка машины может обозревать символ λ, либо 1, либо ∀. Тогда система команд выглядит следующим образом:

$$q_1 * \rightarrow q_z \lambda d_+$$
$$q_1 1 \rightarrow q_2 \lambda d_+$$
$$q_2 1 \rightarrow q_2 1 d_+$$
$$q_2 * \rightarrow q_3 1 d_-$$
$$q_3 1 \rightarrow q_3 1 d_-$$
$$q_3 \lambda \rightarrow q_z \lambda d_+.$$

На рис. 10.1 представлен граф состояний и переходов машины.

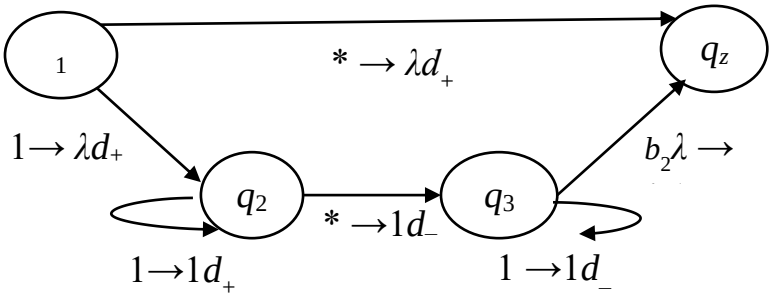


Рис. 10.1. Граф состояний и переходов машины Тьюринга для сложения чисел

2. Построить машину Тьюринга, вычисляющую предикат P = «α – четное число». Число α задано в единичном коде. Зададим алфавит входных и выходных символов A = {λ, 1, И (истина), Л (ложь)} и множество состояний $Q = \{q_1, q_2, q_3, q_4, q_5\}$. Граф состояний и переходов представлен на рис. 10.2.

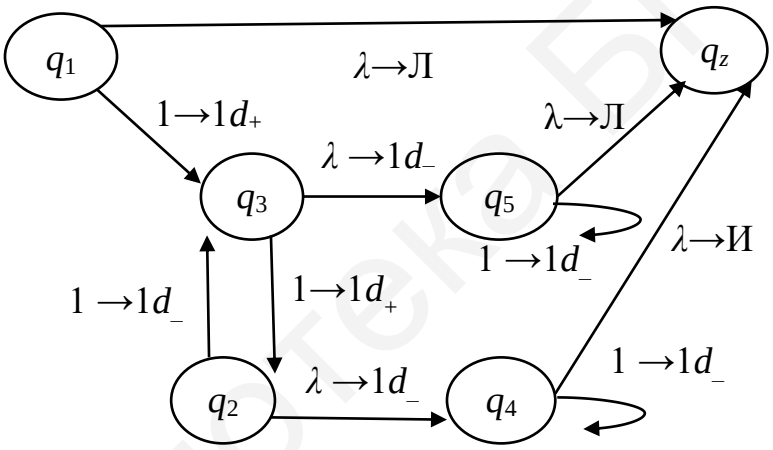


Рис. 10.2. Граф состояний и переходов машины Тьюринга для вычисления предиката

Система команд выглядит следующим образом:

$$q_1\lambda \to q_z Л$$
$$q_1 1 \to q_3 1 d_+$$
$$q_3 1 \to q_2 1 d_+$$
$$q_3\lambda \to q_5 1 d_-$$
$$q_2 1 \to q_3 1 d_-$$
$$q_2\lambda \to q_3 1 d_-$$
$$q_4 1 \to q_4 1 d_-$$
$$q_4\lambda \to q_z И$$
$$q_5 1 \to q_5 1 d_-$$
$$q_5\lambda \to q_z Л.$$

При работе машины возможны два случая:

1) работа машины после конечного числа шагов прекратилась после подачи сигнала «останов»;

2) машина никогда не останавливается, никакого результата она не дает.

В первом случае говорят, что машина применима к исходному данному, во втором – не применима. Соответствия устанавливаются между теми исходными данными, к которым они применимы, и результат ее работы представляет собой некоторую функцию (в математическом смысле).

Если для функции $f(x)$ можно построить машину Тьюринга, которая к ней применима, то говорят, что $f(x)$ вычислена по Тьюрингу. Функцию, для вычисления которой существует машина Тьюринга, называют вычислимой.

Тезис Тьюринга: любая вычислимая функция вычислима по Тьюрингу, или любой алгоритм может быть реализован машиной Тьюринга.

Проблема остановки: можно ли построить машину T_0, такую, что для любой машины Тьюринга T и любых исходных данных α для машины T $T_0(\sum_T, \alpha)$ = И, если машина $T(\alpha)$ останавливается, и $T_0(\sum_T, \alpha)$ = Л, если машина $T(\alpha)$ не останавливается?

Ответ на этот вопрос отрицательный, так как сформулирована и доказана следующая теорема.

Теорема Тьюринга: не существует машины T_0, решающей проблему остановки для произвольной машины Тьюринга T.

Неразрешимость проблемы остановки можно интерпретировать как несуществование общего алгоритма для отладки программы.

10.4. Алгоритмы решения некоторых задач теории графов на графах

Нахождение наибольшего независимого множества – достаточно сложная алгоритмическая задача, поэтому чаще всего ищут близкое к наибольшему независимому множеству и применяют более простые алгоритмы, как, например:

1. В графе выбрать вершину с наименьшей степенью в качестве элемента исходного множества.

2. Удалить выбранную вершину из графа вместе с ее окрестностью (смежные вершины и инцидентные им ребра).

Шаги 1–2 повторять до тех пор, пока множество вершин не станет пустым (в подразд. 3.7 показано, что не всегда такой алгоритм приведет к наибольшему независимому множеству).

Алгоритм поиска наибольшего паросочетания

Паросочетанием графа называется множество его несмежных ребер (не имеющих общих концов).

1. Выбрать ребро с наименьшей степенью (число инцидентных вершин) и включить в искомое паросочетание. Если таковых несколько, выбрать то из них, для которого степень второй граничной вершины наименьшая.

2. Удалить из графа это ребро и ребра, смежные с ним.

Шаги 1–2 повторять до тех пор, пока множество ребер не станет пустым.

Для решения *задачи о нахождении доминирующего множества графа* применяется следующий алгоритм:

1. Дополнить матрицу смежности единицами по главной диагонали.

2. Выбрать из матрицы смежности строку с наибольшим числом единиц и ввести ее в искомое покрытие.

3. Вычеркнуть строку из матрицы, а также столбцы, покрываемые этой строкой.

Повторять шаги 2 и 3 до тех пор, пока в матрице множество столбцов не окажется пустым.

Для решения *задачи о наименьшем вершинном покрытии* необходимо найти кратчайшее строчное покрытие для матрицы инцидентности по алгоритму:

1. В матрице инцидентности графа отыскать столбец с наименьшим числом единиц (если таких столбцов несколько, выбрать самый левый).

2. Среди строк, покрывающих столбец, отыскать строку с наибольшим числом единиц и включить в искомое покрытие (если таких строк несколько, выбрать самую верхнюю).

3. Из матрицы вычеркнуть выбранную строку и столбцы, которые она покрывает.

Повторять шаги 1–3 до тех пор, пока в матрице множество столбцов не окажется пустым.

Всякое множество одноцветных вершин графа является независимым множеством, поэтому *задачу о раскраске графа в минимальное количество цветов* можно осуществить следующим образом:

1. Найти все максимальные независимые множества.

2. Получить кратчайшее покрытие множества вершин графа максимальными независимыми множествами.

3. Удалить повторяющиеся вершины в различных независимых множествах в полученном покрытии так, чтобы любая вершина находилась ровно в одном из множеств.

11. КОНЕЧНЫЙ АВТОМАТ. ТИПЫ

11.1. Автомат с памятью

Рассмотренная в подразд. 5.3 алгебра переключательных схем используется в проектировании цифровых устройств, не обладающих памятью. Такие устройства в любой момент времени на любую комбинацию входных двоичных

сигналов реагируют однозначно, т. е. каждой комбинации двоичных сигналов на входе устройства соответствует определенная комбинация двоичных сигналов на выходе независимо от того, какие сигналы приходили на вход раньше. Отсюда происходят названия «*комбинационный автомат*» и «*комбинационная схема*».

Общепринятой моделью поведения такого устройства является система булевых функций. Комбинации двоичных сигналов иногда удобно обозначать абстрактными символами или буквами некоторого заданного алфавита. Пусть имеется входной алфавит $A = \{a_1, a_2, \ldots, a_\alpha\}$ и выходной алфавит $B = \{b_1, b_2, \ldots, b_\beta\}$. Тогда общий вид описания поведения комбинационного автомата можно представить в виде табл. 11.1, где $b_{i_j} \in B$. Данная таблица является аналогом таблицы истинности. Она полностью определяет алгоритм функционирования заданного устройства.

Таблица 11.1

Пример задания комбинационного автомата

Вход	Выход
a_1	b_{i_1}
a_2	b_{i_2}
…	…
a_α	b_{i_α}

Часто алгоритм функционирования устройства бывает таким, что простым соответствием вида табл. 11.1 его задать невозможно. Для его описания необходимо ввести время, и работа устройства рассматривается во времени. Время носит дискретный характер, т. е. течение времени представляется как последовательность моментов. В любой момент времени устройство принимает сигнал на входе и выдает сигнал на выходе. В различные моменты устройство может реагировать на один и тот же входной сигнал по-разному. Пусть, например, на вход устройства с входным и выходным алфавитами $A = \{a_1, a_2, a_3, a_4\}$ и $B = \{b_1, b_2, b_3\}$ пришла последовательность сигналов

$$a_1\ a_1\ a_3\ a_2\ a_2\ a_4\ a_3,$$

а на выходе наблюдается последовательность сигналов

$$b_2\ b_3\ b_2\ b_1\ b_3\ b_2\ b_1.$$

В первый момент времени устройство на входной сигнал a_1 реагировало выходным сигналом b_2, а во второй момент на тот же сигнал – сигналом b_3. На

сигналы a_2 и a_3 в разные моменты оно реагировало также по-разному, т. е. поведение устройства нельзя описать тем же способом, что использовался для описания поведения комбинационного автомата.

Эта неоднозначность устраняется, если ввести понятие *внутреннего состояния* или просто *состояния*. Тогда то, что устройство реагирует на один и тот же сигнал в разные моменты по-разному, можно объяснить так, что оно находится при этом в разных состояниях. Приняв входной сигнал, устройство выдает выходной сигнал и может перейти в другое состояние.

Состояния образуют множество $Q = \{q_1, q_2, \dots, q_\gamma\}$. Тогда алгоритм функционирования устройства можно представить совокупностью строк вида

$$(a_i, q_j) \to (q_s, b_t),$$

где $a_i \in A$, $b_t \in B$, q_j, $q_s \in Q$.

Устройство, поведение которого можно задать в таком виде, носит название «*автомат с памятью*» или «*последовательностный автомат*». Последний термин заключает в себе смысл отображения входной последовательности в другую.

Соответствие $(a_i, q_j) \to (q_s, b_t)$ можно задать таблицей, подобной табл. 11.1, где фактически представлены две функции, определенные на множестве $A \times Q$. Областью значений одной функции является множество Q, другой функции – множество B.

Моделью описанного устройства является *конечный автомат*, представляющий собой совокупность следующих объектов:

$A = \{a_1, a_2, \dots, a_\alpha\}$ – множество входных символов, или входной алфавит;

$B = \{b_1, b_2, \dots, b_\beta\}$ – множество выходных символов, или выходной алфавит;

$Q = \{q_1, q_2, \dots, q_\gamma\}$ – множество состояний, или внутренний алфавит;

$\Psi: A \times Q \to Q$ – функция переходов;

$\Phi: A \times Q \to B$ – функция выходов.

Для конечного автомата используется обозначение (A, B, Q, Ψ, Φ). Слово «конечный» подчеркивает, что все три множества, входящие в состав данной модели, конечны. В том виде, в каком эта модель здесь представлена, она носит название «*автомат Мили*». Другой разновидностью данной модели является *автомат Мура*, отличающийся от автомата Мили только тем, что функция выходов не зависит от входного символа, т. е. представляет собой отображение $\Phi: Q \to B$.

Значением функции $\Psi(a, q)$ является состояние q^+, в которое переходит автомат из состояния q, если на вход его подан символ a. Значением функции $\Phi(a, q)$ является выходной символ, выдаваемый автоматом в состоянии q при поступлении на его вход символа a, а значением функции $\Phi(q)$ автомата Мура – выходной символ b, который выдает автомат, находясь в состоянии q.

11.2. Представления автомата

Конечный автомат удобно представлять *таблицей переходов* и *таблицей выходов*. Строкам этих таблиц соответствуют состояния автомата, столбцам – входные символы. На пересечении строки, соответствующей состоянию q, и столбца, соответствующего входному символу a, в таблице переходов записывается значение $\Psi(a, q)$, а в таблице выходов – значение $\Phi(a, q)$. Другими словами, в первом случае в клетке таблицы указывается состояние, в которое автомат переходит из состояния q при поступлении на его вход символа a, а во втором случае – выходной символ, который при этом выдает автомат. В табл. 11.2 и 11.3 представлены примеры описанного представления автомата Мили, у которого $A = \{a_1, a_2, a_3, a_4\}$, $B = \{b_1, b_2\}$ и $Q = \{q_1, q_2, q_3\}$.

Таблица 11.2
Функция Ψ

	a_1	a_2	a_3	a_4
q_1	q_1	q_2	q_1	q_2
q_2	q_3	q_1	q_3	q_1
q_3	q_3	q_1	q_1	q_1

Таблица 11.3
Функция Φ

	a_1	a_2	a_3	a_4
q_1	b_1	b_1	b_2	b_1
q_2	b_1	b_2	b_2	b_1
q_3	b_2	b_2	b_1	b_1

Зная начальное состояние автомата и входную последовательность, нетрудно получить по этим таблицам соответствующую последовательность выходных символов. Приведем пример такого соответствия для автомата, заданного табл. 11.2 и 11.3, на вход которого поступила последовательность символов $a_1, a_2, a_2, a_1, a_3, a_4, a_1, a_4$ при начальном состоянии q_1. Покажем также состояния, которые проходит автомат:

a_1	a_2	a_2	a_1	a_3	a_4	a_1	a_4
q_1	q_1	q_2	q_1	q_1	q_1	q_2	q_3
b_1	b_1	b_2	b_1	b_2	b_1	b_1	b_1.

Автомат Мура представляется одной таблицей переходов, к которой добавлен один столбец со значениями функции выходов (табл. 11.4).

Можно свести таблицу переходов и таблицу выходов автомата Мили в одну таблицу, которую называют *таблицей переходов и выходов*. Такая таблица для автомата, заданного в виде табл. 11.2 и 11.3, имеет вид табл. 11.5.

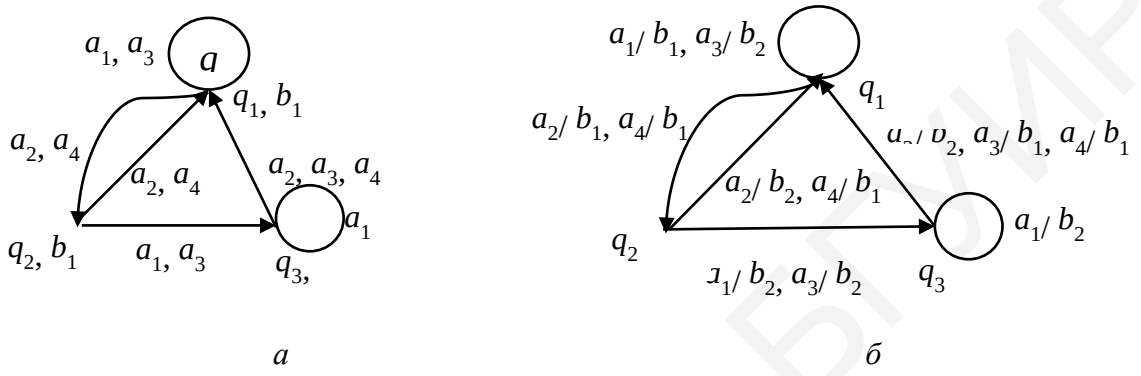
Таблица 11.4
Таблица переходов автомата Мура

	a_1	a_2	a_3	a_4	Φ
q_1	q_1	q_2	q_1	q_2	b_1
q_2	q_3	q_1	q_3	q_1	b_1
q_3	q_3	q_1	q_1	q_1	b_2

Таблица 11.5
Таблица переходов и выходов автомата Мили

	a_1	a_2	a_3	a_4
q_1	q_1,b_1	q_2,b_1	q_1,b_2	q_2,b_1
q_2	q_3,b_1	q_1,b_2	q_3,b_2	q_1,b_1
q_3	q_3,b_2	q_1,b_2	q_1,b_1	q_1,b_1

Более наглядным при небольшом числе состояний является представление автомата в виде *графа поведения автомата*, который представляет собой ориентированный граф. Его вершины соответствуют состояниям автомата, а дуги – переходам между состояниями. При этом дуга помечается всеми входными символами, которые вызывают соответствующий переход, и выходными символами, сопровождающими данный переход (в случае автомата Мили). В случае автомата Мура выходными символами помечаются вершины, соответствующие состояниям, в которых находится автомат при выдаче данных символов. На рис. 11.1 изображены графы переходов автоматов, заданных табл. 11.4 и 11.5.



а *б*

Рис. 11.1. Примеры графов поведения:
а – автомата Мура; *б* – автомата Мили

Еще одним способом представления автомата является *матрица поведения*, представляющая собой квадратную матрицу, строки и столбцы которой помечаются состояниями автомата. В случае автомата Мура на пересечении строки q_i и столбца q_j матрицы поведения записываются входные символы, переводящие автомат из состояния q_i в состояние q_j, а строки помечаются также и выходными символами. В случае автомата Мили элементы матрицы поведения, кроме входных символов, вызывающих соответствующие переходы, содержат выходные символы, которые сопровождают эти переходы. Если из состояния q_i нет перехода в состояние q_j, то на пересечении строки q_i и столбца q_j ставится прочерк. Для рассмотренных автоматов далее представлены матрицы поведения, причем первая из них – матрица поведения автомата Мура, вторая – матрица поведения автомата Мили:

$$\begin{array}{c} \\ q_1, b_1 \\ q_2, b_1 \\ q_3, b_2 \end{array} \begin{array}{c} \begin{array}{ccc} q_1 & q_2 & q_3 \end{array} \\ \begin{bmatrix} a_1, a_3 & a_2, a_4 & - \\ q_2, a_4 & - & a_1 \\ a_2, a_3, a_4 & - & a_1 \end{bmatrix} \end{array},$$

$$\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \end{array}\begin{array}{c} \begin{array}{ccc} q_1 & q_2 & q_3 \end{array} \\ \begin{bmatrix} a_1/b_1,\ a_3/b_2 & a_2/b_1,\ a_4/b_1 & - \\ a_2/b_2,\ a_4/b_1 & - & a_1/b_1 \\ a_2/b_3,\ a_3/b_1,\ a_4/b_1 & - & a_1/b_2 \end{bmatrix} \end{array}.$$

11.3. Связь между моделями Мили и Мура

Всякое отображение входных последовательностей в выходные может быть реализовано как с помощью модели Мили, так и с помощью модели Мура. Определим преобразование, переводящее любой автомат Мили в эквивалентный ему автомат Мура, а также преобразование, переводящее любой автомат Мура в эквивалентный ему автомат Мили.

Пусть задан автомат Мура $M = (A, B, Q, \Psi, \Phi)$ и требуется получить эквивалентный ему автомат Мили $M^* = (A^*, B^*, Q^*, \Psi^*, \Phi^*)$.

Очевидно, $A^* = A$ и $B^* = B$. Положим $Q^* = Q$ и $\Psi^* = \Psi$, а Φ^* определим следующим образом. Пусть $\Psi(a, q) = q'$ и $\Phi(q') = b$, где $q, q' \in Q$, $a \in A$ и $b \in B$. Это означает, что автомат, будучи в состоянии q, отвечает на входной символ a выходным символом b, который выдается в следующий момент времени, когда автомат окажется в состоянии q'. Следовательно, можно считать, что $\Phi^*(a, q) = b$. Автомат Мура и эквивалентный ему автомат Мили представлены в табл. 11.6 и 11.7 соответственно.

Таблица 11.6

Таблица переходов автомата Мура

	a_1	a_2	a_3	Φ
q_1	q_3	q_2	q_2	0
q_2	q_1	q_4	q_3	1
q_3	q_2	q_2	q_1	0
q_4	q_3	q_4	q_4	1

Таблица 11.7

Таблица переходов и выходов автомата Мили

	a_1	a_2	a_3
q_1	q_3,0	q_2,1	q_2,1
q_2	q_1,0	q_4,1	q_3,0
q_3	q_2,1	q_2,1	q_1,0
q_4	q_3,0	q_4,1	q_4,1

Пусть теперь задан автомат Мили $M = (A, B, Q, \Psi, \Phi)$ и требуется получить эквивалентный ему автомат Мура $M^* = (A^*, B^*, Q^*, \Psi^*, \Phi^*)$. Как и в предыдущем случае, имеем $A^* = A$ и $B^* = B$. Определим Q^* следующим образом. Рассмотрим все такие пары вида (q, b), где $q \in Q$, $b \in B$, что для каждой (q, b) имеется такая пара (a, q'), что $\Psi(a, q') = q$ и $\Phi(a, q') = b$ $(a \in A, q' \in Q)$. Каждой паре (q, b) поставим в соответствие состояние $q^* \in Q^*$ и определим функции Ψ^* и Φ^* следующим образом:

$$\Psi^*(a, q^*) = \Psi^*(a, (q, b)) = (\Psi(a, q), \Phi(a, q)); \quad \Phi^*(q^*) = \Phi^*((q, b)) = b.$$

Если автомат имеет состояние, в которое он никогда не переходит (это может быть начальное состояние), то всякому такому состоянию ставится в соответствие состояние автомата Мура, переходы из него определяются аналогично, а выходной символ при нем не определен.

Если автомат является частичным, то достаточно ввести новое состояние, соответствующее неопределенному состоянию, и новый выходной символ, соответствующий неопределенному выходному символу, и после описанных преобразований вернуться к неопределенному состоянию и неопределенному выходному символу. Переходы из такого состояния не определены. Автомат Мили и эквивалентный ему автомат Мура представлены в табл. 11.8 и 11.9 соответственно.

Таблица 11.8

Таблица переходов частичного автомата Мили

	a_1	a_2	a_3	a_4
q_1	$-,b_1$	q_2,b_1	$-,-$	q_2,b_1
q_2	q_3,b_1	$-,b_2$	$q_3,-$	q_2,b_1
q_3	q_3,b_2	$-,-$	q_2,b_1	q_2,b_1

Таблица 11.9

Таблица переходов и выходов частичного автомата Мура

			a_1	a_2	a_3	a_4	Φ^*
q_1	$\to$	q^*_1	q^*_6	q^*_2	$-$	q^*_2	$-$
q_2,b_1	$\to$	q^*_2	q^*_3	q^*_7	q^*_5	q^*_2	b_1
q_3,b_1	$\to$	q^*_3	q^*_4	$-$	q^*_2	q^*_2	b_1
q_3,b_2	$\to$	q^*_4	q^*_4	$-$	q^*_2	q^*_2	b_2
$q_3,-$	$\to$	q^*_5	q^*_4	$-$	q^*_2	q^*_2	$-$
$-,b_1$	$\to$	q^*_6	$-$	$-$	$-$	$-$	b_1
$-,b_2$	$\to$	q^*_7	$-$	$-$	$-$	$-$	b_2

11.4. Автомат с абстрактным состоянием. Булев автомат

Широко распространенным типом автомата является модель, описываемая одной многозначной внутренней переменной q и многими входными и выходными булевыми переменными $x_1, x_2, \dots, x_n$ и $y_1, y_2, \dots, y_m$. Поведение такого автомата задается системой уравнений

$$\begin{aligned} q^+ &= \psi(x_1, x_2, \dots, x_n; q);\\ y_1 &= \varphi_1(x_1, x_2, \dots, x_n; q);\\ y_2 &= \varphi_2(x_1, x_2, \dots, x_n; q);\\ &\dots\\ y_m &= \varphi_m(x_1, x_2, \dots, x_n; q), \end{aligned}$$

более компактно представляемой в векторной форме

$$\begin{aligned} q^+ &= \boldsymbol{\psi}(\boldsymbol{x}, q);\\ \boldsymbol{y} &= \boldsymbol{\varphi}(\boldsymbol{x}, q). \end{aligned}$$

Функции $\boldsymbol{\psi}$ и $\boldsymbol{\varphi}$ отличаются от введенных ранее Ψ и Φ только тем, что многозначные входная и выходная переменные оказались замененными на соответствующие булевы векторы, но внутренняя переменная осталась многозначной.

Описанная модель называется *автоматом с абстрактным состоянием*. Ею удобно пользоваться на начальных этапах логического проектирования дискретных устройств, когда вход и выход устройства описываются как некоторые множества булевых переменных, имеющих конкретную техническую интерпретацию, в то время как множество внутренних переменных представляется пока в простейшей форме, в виде одной многозначной переменной q. Число значений переменной q полагается равным числу различных состояний автомата, при котором он может реализовать заданное функциональное отношение между входом и выходом.

Если заменить внутреннюю переменную q на соответствующий булев вектор $\boldsymbol{z} = (z_1, z_2, \ldots, z_k)$, то получится система уравнений, в которой все переменные и все функции оказываются булевыми:

$$
\begin{aligned}
z_1^+ &= \psi_1(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k);\\
z_2^+ &= \psi_2(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k);\\
&\ldots\\
z_k^+ &= \psi_k(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k);\\
y_1 &= \varphi_1(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k);\\
y_2 &= \varphi_2(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k);\\
&\ldots\\
y_m &= \varphi_m(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k).
\end{aligned}
$$

Эта модель называется *булевым автоматом*. Ее также можно представить в компактной векторной форме:

$$
\begin{aligned}
\boldsymbol{z}^+ &= \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{z});\\
\boldsymbol{y} &= \boldsymbol{\varphi}(\boldsymbol{x}, \boldsymbol{z}).
\end{aligned}
$$

Булев автомат в определенном смысле ближе к реальным дискретным устройствам, поскольку его переменные непосредственно реализуются физическими переменными устройства, в частности, на типичных для современной техники элементах с двумя устойчивыми состояниями. Векторы $\boldsymbol{x}$, $\boldsymbol{y}$ и $\boldsymbol{z}$ показывают структуру абстрактных символов a и b и состояния q. Приведенная система функций соответствует структуре, изображенной на рис. 11.2, где КС – комбинационная схема, реализующая приведенную выше систему, а П – блок памяти, осуществляющий задержку на период между соседними моментами времени.

Переменная z_i представляет состояние i-го двоичного элемента памяти, а выражение

$$z_i^+ = \psi_i(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_k)$$

надо понимать так, что состояние i-го элемента памяти определяется значениями входных символов и состояниями элементов памяти в предыдущий момент времени.

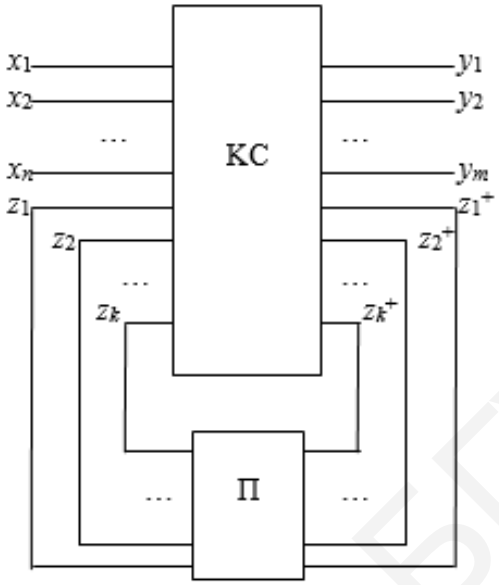


Рис. 11.2. Структура булева автомата

12. МИНИМИЗАЦИЯ ПОЛНЫХ АВТОМАТОВ

12.1. Эквивалентность состояний. Постановка задачи минимизации

Одно и то же поведение может быть задано автоматами с различным числом состояний. Отсюда возникает задача *минимизации числа состояний автомата*, или просто *минимизации автомата*.

Основным понятием, используемым в данной задаче, является эквивалентность состояний автомата. Состояние q_i автомата M_1 и состояние q_j автомата M_2 *эквивалентны*, если автомат M_1 при начальном состоянии q_i и автомат M_2 при начальном состоянии q_j под воздействием любой входной последовательности выдают одинаковые последовательности на выходе. При этом M_1 и M_2 могут представлять собой один и тот же автомат.

Отношение эквивалентности состояний обладает свойствами рефлексивности (каждое состояние эквивалентно самому себе), симметричности (если состояние q_i эквивалентно состоянию q_j, то q_j эквивалентно q_i) и транзитивности (если состояние q_i эквивалентно состоянию q_k, а q_k эквивалентно состоянию q_j, то q_i эквивалентно q_j). Как известно (см. подразд. 2.5), бинарное отношение на некотором множестве, обладающее такими свойствами, определяет разбиение данного множества на классы эквивалентности.

Автомат M_1 и автомат M_2 *эквивалентны*, если для каждого состояния одного из них имеется хотя бы одно эквивалентное ему состояние другого.

Задача минимизации полного автомата ставится следующим образом: для автомата M найти эквивалентный ему автомат с минимальным числом состояний.

Пусть задан автомат $M = (A, B, Q, \Psi, \Phi)$ и требуется найти эквивалентный ему автомат $M' = (A, B, Q', \Psi', \Phi')$, обладающий минимальным числом состояний. Допустим, установлено отношение эквивалентности на множестве состояний автомата M и определены классы эквивалентности $S_1, S_2, \ldots, S_m$. Обозначим символом $q^{(i)}$ любое состояние, принадлежащее классу S_i. Тогда минимальный автомат строится следующим образом.

Формируется множество состояний $Q' = \{q'_1, q'_2, \ldots, q'_m\}$, где каждому q'_i поставлен во взаимно однозначное соответствие класс эквивалентности S_i.

Если для некоторых $q^{(i)} \in S_i$ и $a \in A$ имеет место $\Phi(a, q^{(i)}) = b$, где $b \in B$, то $\Phi'(a, q'_i) = b$.

Если для некоторых $q^{(i)} \in S_i$ и $a \in A$ имеет место $\Psi(a, q^{(i)}) = q^{(j)}$, где $q^{(j)} \in S_j$, то $\Psi'(a, q'_i) = q'_j$.

12.2. Установление эквивалентности состояний

В следующих трех случаях эквивалентность или неэквивалентность состояний определяется непосредственно по таблицам переходов и выходов.

1. Найдется такой входной сигнал $a \in A$, что $\Phi(a, q_i) \neq \Phi(a, q_j)$. Тогда состояния q_i и q_j *явно неэквивалентны*. В таблице выходов это соответствует столбцу, где элементы в строках q_i и q_j различны.

2. Для всех $a \in A$ имеют место $\Phi(a, q_i) = \Phi(a, q_j)$ и $\Psi(a, q_i) = \Psi(a, q_j)$. Здесь состояния q_i и q_j *явно эквивалентны*. В таблице переходов и таблице выходов строки q_i и q_j совпадают.

3. Строки q_i и q_j таблицы выходов совпадают, а строки q_i и q_j таблицы переходов совпадут после замены каждого значения q_i на q_j или каждого значения q_j на q_i (если, конечно, такие значения присутствуют в этих строках). В данном случае состояния q_i и q_j также являются явно эквивалентными.

Если состояния не являются явно эквивалентными или явно неэквивалентными, то для решения вопроса об их эквивалентности приходится применять более глубокий анализ [7].

Пусть $q_i \neq q_j$. *Цепью, порождаемой парой состояний* $\langle q_i, q_j \rangle$ полного автомата M, назовем множество C, элементами которого являются следующие пары: 1) сама пара $\langle q_i, q_j \rangle$; 2) если $\langle q_k, q_l \rangle \in C$, то все пары вида $\langle \Psi(a, q_k), \Psi(a, q_l) \rangle$, где $\Psi(a, q_k)$ и $\Psi(a, q_l)$ различны. Другие пары не входят в C.

Таким образом, цепь C, порождаемая парой состояний $\langle q_i, q_j \rangle$, содержит все пары состояний, в которые автомат переходит из состояний q_i и q_j под воздействием всевозможных входных последовательностей (конечных и бесконеч-

ных). Само название «цепь» связано с характером образования этого множества: реализуется процесс, подобный цепной реакции (появление одного элемента в цепи вызывает вовлечение множества других элементов). Справедливо следующее утверждение.

Состояния q_i и q_j автомата M являются *эквивалентными*, если и только если в цепи, порождаемой парой состояний $\langle q_i, q_j \rangle$, нет ни одной пары явно неэквивалентных состояний. В этом случае все пары, принадлежащие данной цепи, являются парами эквивалентных состояний.

Действительно, пусть состояние q_i автомата M эквивалентно состоянию q_j, а цепь C, порождаемая парой $\langle q_i, q_j \rangle$, содержит пару явно неэквивалентных состояний, скажем, $\langle q_k, q_l \rangle$. Тогда существует входная последовательность, переводящая автомат M из состояний q_i и q_j в состояния q_k и q_l. Она вызывает различные выходные последовательности, что противоречит условию. С другой стороны, поскольку в C нет ни одной пары явно неэквивалентных состояний, выходные последовательности при одной и той же входной последовательности и при начальных состояниях q_i и q_j одинаковы.

Эквивалентность состояний можно представить *матрицей эквивалентности* – булевой матрицей, строки и столбцы которой соответствуют состояниям автомата. Элемент, расположенный на пересечении i-й строки и j-го столбца, имеет значение 1, если и только если состояния q_i и q_j эквивалентны.

Рассмотрим процесс минимизации полного автомата на примере табл. 12.1.

Таблица 12.1

Таблица переходов и выходов минимизируемого автомата

	a_1	a_2	a_3
1	2,1	2,0	5,0
2	1,0	4,1	4,1
3	2,1	2,0	5,0
4	3,0	2,1	2,1
5	6,1	4,0	3,0
6	8,0	9,1	6,1
7	6,1	2,0	8,0
8	4,1	4,0	7,0
9	7,0	9,1	7,1

Сначала найдем явно эквивалентные и явно неэквивалентные состояния. В силу рефлексивности и симметричности отношения эквивалентности достаточно иметь только часть матрицы эквивалентности, расположенную выше (или ниже) главной диагонали. Получим матрицу, пустые элементы в верхней части которой соответствуют парам. Непосредственно из таблицы переходов и выходов не видно, эквивалентны в этих парах состояния или нет, т. е. они не являются ни явно эквивалентными, ни явно неэквивалентными:

$$
\begin{array}{c}
\begin{array}{cccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left[\begin{array}{cccccccc}
0 & 1 & 0 & & 0 & & & 0 \\
& 0 & & 0 & & 0 & 0 & \\
& & 0 & & 0 & & & 0 \\
& & & 0 & & 0 & 0 & \\
& & & & 0 & & & 0 \\
& & & & & 0 & 0 & \\
& & & & & & & 0 \\
& & & & & & & 0
\end{array}\right]
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\end{array} \cdot
$$

Для пар состояний, которым соответствуют пустые места в этой матрице, необходимо строить порождаемые ими цепи, которые удобно представлять ориентированными графами. Вершинам такого графа соответствуют пары, принадлежащие формируемой цепи. Из вершины $\langle q_i, q_j \rangle$ в вершину $\langle q_k, q_l \rangle$ направлена дуга, если $\langle q_k, q_l \rangle = \langle \Psi(a, q_i), \Psi(a, q_j) \rangle$. Необязательно строить всю цепь, если порождающая ее пара не является эквивалентной. Можно прекратить процесс после появления первой пары, которой соответствует нулевое значение элемента матрицы эквивалентности. Тогда неэквивалентными объявляются все пары, лежащие на пути к данной паре от порождающей пары.

Часть цепи, порождаемой парой ⟨1, 5⟩, изображена на рис. 12.1.

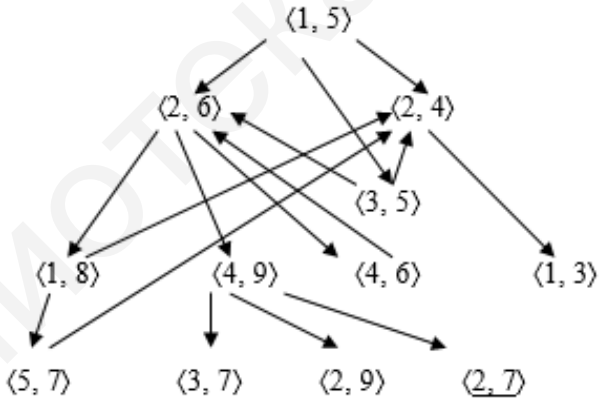


Рис. 12.1. Часть цепи, порождаемой парой ⟨1, 5⟩

Она содержит пару явно неэквивалентных состояний ⟨2, 7⟩. Дальше цепь можно не строить, так как ясно, что пара ⟨1, 5⟩ оказалась неэквивалентной и, кроме нее, неэквивалентными оказались пары ⟨4, 9⟩ и ⟨2, 6⟩, лежащие на пути из вершины ⟨1, 5⟩ к вершине ⟨2, 7⟩.

Таким образом, матрица дополняется нулями в виде значений элементов, соответствующих парам ⟨1, 5⟩, ⟨2, 6⟩ и ⟨4, 9⟩:

$$
\begin{array}{c}
\begin{array}{cccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left[\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 0 & & & 0 \\
 & 0 & & 0 & 0 & 0 & 0 & \\
 & & 0 & & 0 & & 0 & \\
 & & & 0 & & 0 & 0 & 0 \\
 & & & & 0 & & 0 & \\
 & & & & & 0 & 0 & \\
 & & & & & & 0 & \\
 & & & & & & 0 &
\end{array}\right]
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\end{array}.
$$

Состояния, принадлежащие паре ⟨1, 7⟩, также оказались неэквивалентными. Цепь, порождаемая парой ⟨1, 8⟩ (рис. 12.2), не содержит пар неэквивалентных состояний. Поэтому любая из пар, принадлежащих этому множеству, является парой эквивалентных состояний. Окончательно получим следующую матрицу эквивалентности:

$$
\begin{array}{c}
\begin{array}{cccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\
\left[\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & & 0 & 0 & 0 & 0 & 1 & 0 \\
 & & & 0 & 0 & 0 & 0 & 0 \\
 & & & & 0 & 1 & 0 & 0 \\
 & & & & & 0 & 0 & 0 \\
 & & & & & & 0 & 0 \\
 & & & & & & & 0
\end{array}\right]
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\end{array}.
$$

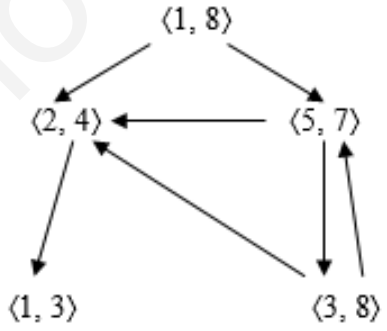


Рис. 12.2. Цепь, порождаемая парой ⟨1, 8⟩

Классы {1, 3, 8}, {2, 4}, {5, 7}, {6} и {9} определяются по строкам матрицы эквивалентности с учетом свойства транзитивности и симметричности отношения эквивалентности. Полученным классам эквивалентности ставим в соответствие состояния нового автомата 1, 2, 3, 4 и 5, в результате чего получим табл. 12.2, представляющую минимальный автомат, эквивалентный заданному.

Таблица 12.2

Таблица переходов и выходов минимального автомата

	a_1	a_2	a_3
1	2,1	2,0	3,0
2	1,0	2,1	2,1
3	4,1	2,0	1,0
4	1,0	5,1	4,1
5	3,0	5,1	3,1

Конечный автомат может служить моделью не только в задачах проектирования цифровых устройств. Можно назвать такие области, как распознавание образов, исследование поведения различных систем и т. п., где может использоваться теория автоматов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андерсон, Д. А. Дискретная математика и комбинаторика / Д. А. Андерсон. – М. : Изд. дом «Вильямс», 2003. – 960 с.

2. Ахо, А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – М. : Мир, 1979. – 536 с.

3. Глушков, В. М. Синтез цифровых автоматов / В. М. Глушков. – М. : ГИФМЛ, 1962. – 476 с.

4. Донской, В. И. Дискретная математика / В. И. Донской. – Симферополь : Сонат, 2000. – 354 с.

5. Закревский, А. Д. Основы логического проектирования. В 3 кн. Кн. 1 : Комбинаторные алгоритмы дискретной математики / А. Д. Закревский, Ю. В. Поттосин, Л. Д. Черемисинова. – Минск : ОИПИ НАН Беларуси, 2004. – 226 с.

6. Закревский, А. Д. Основы логического проектирования. В 3 кн. Кн. 2 : Оптимизация в булевом пространстве / А. Д. Закревский, Ю. В. Поттосин, Л. Д. Черемисинова. – Минск : ОИПИ НАН Беларуси, 2004. – 240 с.

7. Закревский, А. Д. Логические основы проектирования дискретных устройств / А. Д. Закревский, Ю. В. Поттосин, Л. Д. Черемисинова. – М. : ФИЗМАТЛИТ, 2007. – 592 с.

8. Зыков, А. А. Основы теории графов / А. А. Зыков.– М. : Наука, 1987. – 384 с.

9. Карпов, Ю. Г. Теория автоматов / Ю. Г. Карпов. – СПб. : Питер, 2002. – 206 с.

10. Кнут, Д. Э. Искусство программирования. Т. 4 : Комбинаторные алгоритмы. Ч. 1 / Д. Э. Кнут. – М. : ООО «И. Д. Вильямс», 2013. – 960 с.

11. Кофман, А. Введение в прикладную комбинаторику / А. Кофман. – М. : Наука, 1975. – 480 с.

12. Кристофидес, Н. Теория графов. Алгоритмический подход / Н. Кристофидес. – М. : Мир, 1978. – 432 с.

13. Кузнецов, О. П. Дискретная математика для инженеров / О. П. Кузнецов, Г. М. Адельсон-Вельский. – М. : Энергия, 1988. – 480 с.

14. Лекции по теории графов / В. А. Емеличев [и др]. – М. : Наука, 1990. – 384 с.

15. Москинова, Г. И. Дискретная математика / Г. И. Москинова. – М. : Логос, 2002. – 238 с.

16. Поттосина, С. А. Основы дискретной математики и теории алгоритмов : практикум для студ. спец. «Информационные системы и технологии в экономике» / С. А. Поттосина, Т. Г. Пинчук. – Минск : БГУИР, 2009. – 86 с.

17. Поттосин, Ю. В. Методы дискретной математики в проектировании цифровых устройств / Ю. В. Поттосин. – Saarbrücken : LAPLAMBERT Academic Publishing, 2017. – 176 с.

18. Реингольд, Э. Комбинаторные алгоритмы. Теория и практика / Э. Реингольд, Ю. Нивергельт, Н. Део. – М. : Мир, 1980. – 476 с.

19. Рыбников, К. А. Введение в комбинаторный анализ / К. А. Рыбников. – М. : Изд-во Московс. ун-та, 1985. – 308 с.

20. Харари, Ф. Теория графов / Ф. Харари. – М. : Мир, 1973. – 300 с.

21. Яблонский, С. В. Введение в дискретную математику / С. В. Яблонский. – М. : Наука, 1986. – 384 с.

Св. план 2020, поз. 51

Учебное издание

Поттосин Юрий Васильевич
Пинчук Татьяна Георгиевна
Поттосина Светлана Анатольевна

ОСНОВЫ ДИСКРЕТНОЙ МАТЕМАТИКИ И ТЕОРИИ АЛГОРИТМОВ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 03.03.2021. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 7,32. Уч.-изд. л. 8,0. Тираж 100 экз. Заказ 226.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск