УДК 004.421(075.8)
ББК 22.12
Г12

РЕЦЕНЗЕНТЫ:

Кафедра информатики и программного обеспечения информационных систем
Московского государственного института электронной техники (технического университета);

Ю.Н. Беляков,
доктор технических наук, **профессор**
(открытое акционерное общество <<ОТИк-гр|упп>>)

Гагарина Л.Г.

П2 Алгоритмы и структуры данных: учеб. пфоб,]ге/Л.Г. Гагарина, В.Д. Колдаев. - М: Финансы и статиста! ИНФРА-М, 2009. - 304 с : ил.

ISBN 978-5-279-03351-5 (Финансы и ста^исфжа)
ISBN 978-5-16-003682-3 (ИНФРА-М)

Приведены основные понятия алгоритмизации, сврйстра алгоритмов, общие принципы их построения, основные алгоритмические 1(онст|рукции, представлена эволюция языков программирования. Рассмотрен широкий методов обработки линейных и нелинейных структур данных. Описана техн|олог|ия функционирования и оценки функции сложности различных алгоритмов для работы с очередями, стеками, списками,'деревьями, таблицами и графами приложениях приведены системы счисления и методы измерения количества инфЪрмаши

Для студентов, аспирантов, преподавателей, специ) шис|тов — от инженера до системного аналитика в области численных методов и кфмпьротерного моделирования. Может быть использовано для самообразования

2404000000 - 026
010(01) - 2009 **172 - 2008**

УДК 004.421(075.8)
ББК 22.12

ISBN 978-5-279-03351-5
ISBN 978-5-16-003682-3

Оглавление

Предисловие

В последние годы программирование для персональных компьютеров сформировалось в некоторую дисциплину, владение которой стало ключевым моментом, определяющим успех многих инженерных проектов, а сама она превратилась в объект научного исследования. Из ремесла программирование перешло в разряд академических наук. Крупные специалисты в области программирования показали, что программы поддаются точному анализу, основанному на строгих математических рассуждениях. При этом можно избежать многих ошибок, традиционных для программистов, если они будут осмысленно пользоваться методами и приемами, которые раньше применялись ими интуитивно. Кроме того программисты уделяли гораздо больше внимания построению и анализу программы, чем выбору представления данных. Современная методология программирования предполагает, что оба аспекта программирования—запись алгоритма на языке программирования и выбор структур представления данных — заслуживают одинакового внимания. Всем, кто занят вопросами программирования для вычислительной техники, известно, что решение о том, как представлять данные, невозможно принимать без понимания того, какие алгоритмы будут к ним применяться, и Наоборот, выбор алгоритма часто очень сильно зависит от строения данных, к которым он применяется.

Без знания структур данных и алгоритмов невозможно создать серьезный программный продукт. Ни одна информационная система не обходится без программного обеспечения, она просто не может существовать без этой компоненты. В связи с этим задача данного пособия состоит в следующем:

• познакомить со всем разнообразием имеющихся структур данных, показать, как эти структуры реализованы в языках программирования;

• рассмотреть основные операции, которые выполняются над структурами данных;

• показать особенности структурного подхода к разработке алгоритмов, продемонстрировать порядок их разработки.

Тщательно подобранный материал, вошедший в книгу, включает в себя основные классы алгоритмов, которые в том или ином виде наиболее часто встречаются в практике программирования. Настоящее пособие предлагает массу рецептов усовершенствования программ и, что самое главное, учит самостоятельно находить эти рецепты. Ценность пособия состоит в том, что оно предназначено для обучения не столько технике программирования, сколько, если это возможно, «искусству» программирования.

Пособие состоит из двух частей. В части 1 приведены основы алгоритмизации. В части 2 рассмотрены структуры данных, их представление и алгоритмы обработки, без знания которых невозможно современное компьютерное моделирование. Описываются различные алгоритмы для работы с очередями, стеками, списками, деревьями, таблицами и графами. В конце каждой главы содержится большое количество контрольных вопросов и задач для самостоятельного решения.

В приложениях приведены сведения, необходимые для специалистов, занимающихся программированием и компьютерным моделированием: в приложении 1 — операции по переводу чисел из одной системы счисления в другую и двоичная арифметика; в приложении 2 — методики измерения количества информации. Завершает книгу словарь терминов, используемых при моделировании алгоритмов.

Пособие предназначено для студентов, обучающихся по направлению и специальностям «Вычислительные машины, системы, комплексы и сети», «Автоматизированные системы обработки информации и управления», «Программное обеспечение вычислительной техники и автоматизированных систем».

ЧАСТЬ 1

ОСНОВЫ АЛГОРИТМИЗАЦИИ

Глава 1. Структурная организация данных

Обработка информации на персональных электронных вычислительных машинах (ПЭВМ) требует, чтобы ее структура была определена и точно представлена в ПЭВМ. Информация, представленная в формализованном виде, пригодном для автоматизированной обработки, называется ***данными.*** Структура данных определяет их Семантику, а также способы организации данных и управления ими. При использовании компьютера для хранения и обработки данных необходимо точно определить тип и структуру данных, а также найти способ наиболее естественного их представления. Структуры данных и алгоритмы служат тем материалом, из которого строятся программы.

Без понимания структур данных и алгоритмов невозможно создать серьезный программный продукт — они служат базовыми элементами любой программы. В организации структур данных и процедур их обработки заложена возможность проверки правильности работы программы.

1.1. Основные понятия структур данных

Компьютер оперирует только с одним видом данных — с отдельными битами, или двоичными цифрами, и работает с этими данными только в соответствии с неизменным набором алгоритмов,

которые определяются системой команд центрального процессора. Задачи, которые решаются с помощью компьютера, редко выражаются на языке битов. Как правило, данные имеют форму чисел, литер, текстов, символов и более сложных структур типа последовательностей, списков и деревьев. Еще разнообразнее алгоритмы, применяемые для решения различных задач; фактически алгоритмов не меньше, чем вычислительных задач.

Структура данных, рассматриваемая без учета ее представления в машинной памяти, называется *абстрактной,* или *логической.* Понятие *«физическая структура данных»* отражает способ физического представления данных в машинной памяти. Вследствие различия между логической и соответствующей ей физической структурами в вычислительной системе существуют процедуры, осуществляющие отображение логической структуры в физическую, и наоборот.

Например, доступ к элементу двумерного массива на логическом уровне реализуется указанием номеров строки и столбца в прямоугольной таблице, на пересечении которых расположен соответствующий элемент. На физическом же уровне к элементу массива доступ осуществляется с помощью функции адресации, которая при известном начальном адресе массива в машинной памяти преобразует номера строки и столбца в адрес соответствующего элемента массива. Таким образом, каждую структуру данных можно характеризовать ее логическим (абстрактным) и физическим (конкретным) представлениями, а также совокупностью операций на этих двух уровнях представления структуры (рис. 1.1).

Очень часто, говоря о той или иной структуре данных, имеют в виду ее логическое представление, так как физическое представление обычно скрыто от программиста. Так как физическая структура данных реализуется в машинной памяти, имеющей ограниченный объем, то при изучении такой структуры должна учитываться проблема распределения и управления памятью.

Структуры данных, применяемые в алгоритмах, могут быть чрезвычайно сложными. В результате выбор правильного представлений данных часто служит ключом к удачному программированию и может в большей степени сказываться на производительности программы, чем детали используемого алгоритма.

Под *структурой данных* в общем случае понимают множество элементов данных и множество связей между ними. Такое определение охватывает все возможные подходы к структуризации данных, но

г

Операции над логической структурой

Логическая структура данных

С

Операции над физической структурой

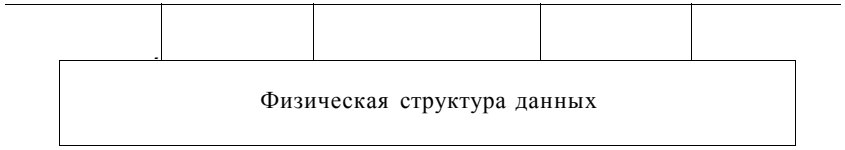


Рис. 1.1. **Уровни представления структуры данных**

в каждой конкретной задаче используются те или иные его аспекты. Поэтому вводится дополнительная классификация структур данных, направления которой соответствуют различным аспектам их рассмотрения. Прежде чем приступать к изучению конкретных структур данных, дадим их общую классификацию по нескольким признакам. Понятие «физическая структура данных» отражает способ физического представления данных в памяти ПЭВМ и называется еще структурой хранения, внутренней структурой или структурой памяти. Рассмотрение структуры данных без учета ее представления в памяти компьютера называется абстрактной, или логической, структурой. В общем случае между логической и соответствующей ей физической структурами существует различие, степень которого зависит от самой структуры и особенностей той среды, в которой она должна быть отражена. Вследствие этого различия существуют процедуры, осуществляющие отображение логической структуры в физическую и, наоборот, физической структуры в логическую. Эти процедуры обеспечивают, кроме того, доступ к физическим структурам и выполнение над ними различных операций, причем каждая операция рассматривается применительно к логической или физической структуре данных.

Различают простые (базовые, примитивные) структуры (типы) данных и интегрированные (структурированные, композитные, слож-

ные). ***Простыми*** называются такие структуры данных, которые не могут быть расчленены на составные части, большие, чем биты. Для физической структуры важным является то обстоятельство, что в данной машинной архитектуре и в данной системе программирования всегда можно заранее знать, каков будет размер выбранного простого типа и какова структура его размещения в памяти. С логической точки зрения простые данные являются неделимыми единицами.

Интегрированными называются такие структуры данных, составными частями которых являются другие структуры данных—простые или, в свою очередь, интегрированные. Интегрированные структуры данных конструируются программистом с использованием средств интеграции данных, предоставляемых языками программирования. В зависимости от отсутствия или наличия явно заданных связей между элементами данных следует различать ***несвязные структуры*** (векторы, массивы, строки, стеки, очереди) и ***связные структуры*** (связные списки).

1.2. Классификация структур данных по признаку изменчивости

Важный признак структуры данных — ее изменчивость, т.е. изменение числа элементов и (или) связей между элементами структуры. В определении изменчивости структуры не отражен факт изменения значений элементов данных, поскольку в этом случае все структуры данных имели бы свойство изменчивости.

По признаку изменчивости различают структуры ***базовые, статические, полустатические, динамические*** и ***файловые.*** Классификация структур данных (СД) по признаку изменчивости приведена на рис. 1.2. Базовые структуры данных, статические, полустатические и динамические характерны для оперативной памяти и часто называются оперативными структурами. Файловые структуры соответствуют структурам данных для внешней памяти.

Вектор (одномерный массив) — структура данных с фиксированным числом элементов одного и того же типа.

Структуры данных

1

Простые базовые	Статические	Полу-статические	[Динамические!	Файловые (файлы)
Числовые	Вектор	Стеки	Линейные связные списки	Последова-тельные
Символьные	Массивы	Очереди	Разветвленные связные списки	Прямого па
Логические	Множества	Деки	Графы	Комбиниро-ванного доступа
Перечисление	Записи	Строки	Деревья	Организо-ванные разделами
Интервал	Таблицы			
Указатели				

Рис. 1.2. Классификация структур данных по признаку изменчивости

Массив — последовательность элементов одного типа, называемого базовым.

Множество—такая структура, которая представляет собой набор неповторяющихся данных одного и того же типа.

Запись — конечное упорядоченное множество полей, характеризующихся различным типом данных.

Таблица — последовательность записей, которые имеют одну и ту же организацию.

Списком называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения. Список, отражающий отношения соседства между элементами, называется линейным.

1.3. Линейные и нелинейные структуры данных

Напомним, что структуры данных представляют собой отношения, заданные на множествах данных.

Пусть ***X*** — множество данных, У — множество значений данных:

X ***Y*** и при этом ***p{x)*** = {ж»: я» € ***X, Xi*** € ***Y}.***

Если на множестве *ц(х)* задано отношение *S(x) < ц(х),* **то** будем называть это отношение структурой данных, а *ц(х)* — **областью** определения структуры данных.

Структура данных называется *простой,* если между значениями данных связи отсутствуют, т.е. *S(x)* — 0 и *ц(х) ф* 0.

Рассмотрим различные классификации структур данных для того, чтобы определить вид структур данных, наиболее адекватно представляющих информацию, а также место, которое они занимают в общей системе классификации структур данных.

Важный признак структуры данных — характер упорядоченности ее элементов.

По этому признаку структуры можно делить *на. линейно-упорядоченные,* или *линейные,* и *нелинейные структуры* (рис. 1.3).

Структуры данных

Линейные	**Нелинейные**
Картезианские (прямоугольные)	**- Деревья (древовидные)**
Строчные (типа ряда)	**- Графы (графовые)**
Списковые	**I— Сплетения (плексы)**

Рис. 1.3. Линейные и нелинейные структуры данных

В зависимости от характера взаимного расположения элементов в памяти ПЭВМ линейные структуры можно разделить на структуры с последовательным распределением их элементов в памяти (векторы, строки, массивы, стеки, очереди) и структуры с произвольным связанным распределением элементов в памяти (односвязные, двухсвязные и прочие списки).

Линейные структуры данных. *Линейные структуры данных* (СД) — это структуры, в которых связи между элементами не зависят от выполнения какого-либо условия. Линейные структуры подразделяются на три типа: картезианские, строчные и списковые.

• картезианские, или прямоугольные, структуры названы так по способу записи данных в виде прямоугольных таблиц. Например:

$$A = \begin{pmatrix} 0 & 3 & 7 & 9 \\ 6 & 2 & 1 & 0 \\ 8 & 5 & 10 & 6 \end{pmatrix}$$ — матрица;

B = (9, 3, 6, 5) — вектор;

Z = {7, 6, 0, 2, 3} — множество элементов.

• Строчные структуры — одномерные, динамически изменяемые структуры данных, различающиеся способами включения и исключения элементов (рис. 1.4).

Строчные

Стек **Дек**

Очередь

Рис. 1.4. Строчные структуры данных

Стек—это последовательность, в которой включение и исключение элемента осуществляется с одной стороны последовательности (рис. 1.5).

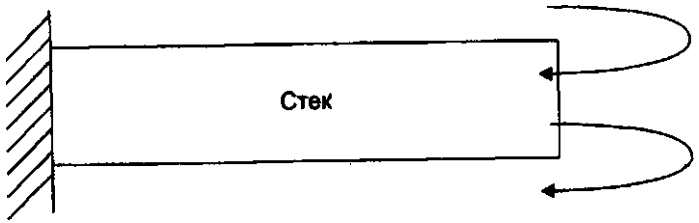


Рис. 1.5. Схема доступа к элементам стека

Пусть, например, дано множество элементов {2, 5, 7, 1,9,3} и задан алгоритм работы стека (X, X, X, Y, X, Y, Y). Тогда после работы алгоритма в стеке останется число 2.

Известные примеры стека — винтовочный патронный магазин, железнодорожный разъезд для сортировки вагонов (рис. 1.6).

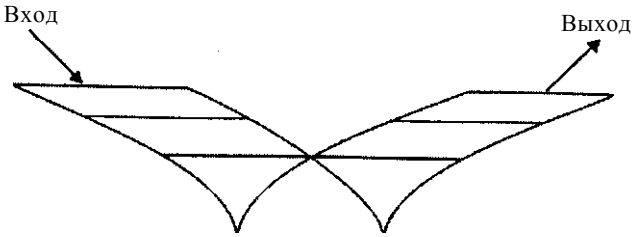


Рис. 1.6. Стек на железных дорогах

Очередь — последовательность, в которую включают элемент) с одной стороны, а исключают — с другой (рис. 1.7). Структур» функционирует по принципу FIFO (первым пришел — первым о^ служи вается).

Рис. 1.7. Схема доступа к элементам очереди

Дек — линейная структура (последовательность), в которой опе| рации включения и исключения элементов могут выполняться как одного, так и с другого конца последовательности (рис. 1.8).

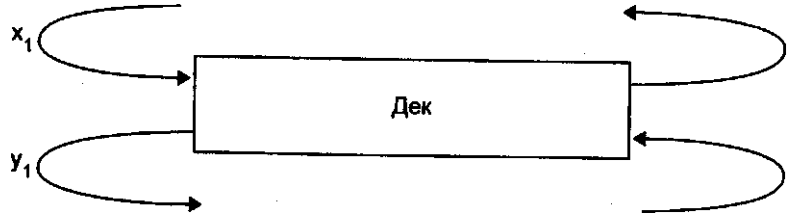


Рис. 1.8. Схема доступа к элементам дека

В *списковых структурах* логический порядок данных впредь ляется указателями. Любая списковая структура представляет собой набор элементов, каждый из которых состоит из двух полей: в одном

из них размещен элемент данных или указатель на него, а в другом — указатель на следующий элемент списка.

Нелинейные структуры данных. *Нелинейные структуры данных* — это СД, у которых связи между элементами зависят от выполнения определенного условия. Примеры нелинейных структур — деревья, графы, многосвязные списки.

•*Древовидные структуры* — это иерархические структуры, состоящие из набора вершин и ребер, каждая вершина содержит определенную информацию и ссылку на вершину нижнего уровня. *Дерево* — это совокупность элементов, называемых *узлами* (один из которых определен как *корень),* и отношений, образующих иерархическую структуру узлов (рис. 1.9).

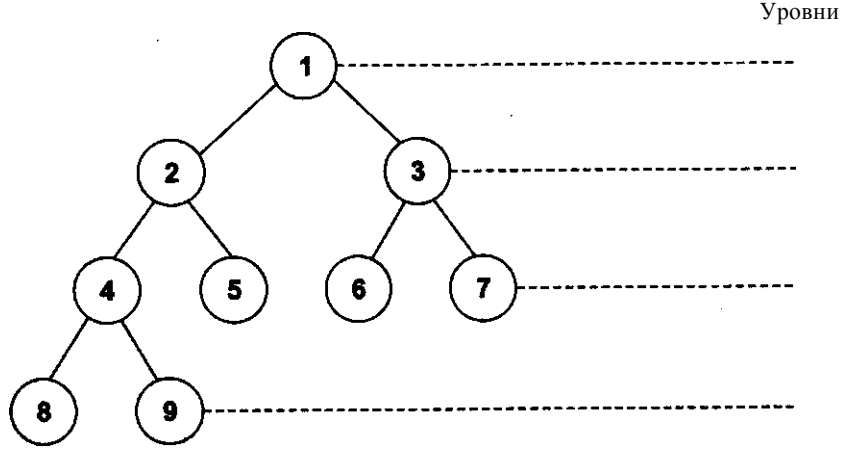


Рис. 1.9. Древовидная структура (дерево)

Вершина, располагающаяся в нулевом уровне, называется корнем дерева (нумерация уровней может начинаться с 1). В корень не входит ни одного ребра. Вершины, из которых не выходит ни одного ребра, называются *листьями* (вершины 8, 9, 5, 6, 7). Дерево, из каждой вершины которого выходит только по два ребра, называется *бинарным* (рис. 1.10).

• *Графы* представляют собой совокупность двух множеств: вершин и ребер. Граф — это с/южная нелинейная многосвязная динамическая структура, отображающая свойства и связи сложного объекта j (рис. 1.11).

Рис. **1.10.** Бинарное дерево

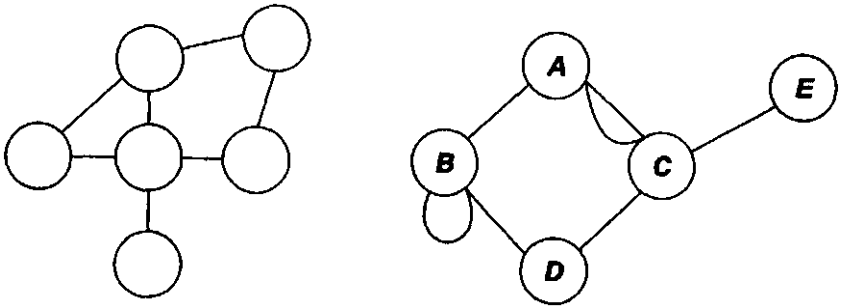


Рис. **1.11.** Примеры графовых структур

Многосвязная структура обладает следующими свойствами:

1) на каждый элемент (узел, вершину) может быть произвольное количество ссылок;

2) каждый элемент может иметь связь с любым количеством других элементов;

3) каждая связка (ребро, дуга) может иметь направление и вес.

Типичными графами являются схемы авиалиний и схемы метро, а на географических картах — изображение железных или автомобильных дорог. Выбранные точки графа называются его *вершинами,* а соединяющие их линии — *ребрами.*

• *Сплетение* (многосвязный список, плекс) — это нелинейная структура данных, объединяющая такие понятия, как дерево, граф и списковая структура.

Основное свойство сплетений, отличное от других типов структур, — наличие у каждого элемента сплетения нескольких полей с указателями на другие элементы того же сплетения (рис. 1.12).

г

г1

0x263A

72

OxOOd

z3

OxO3c6

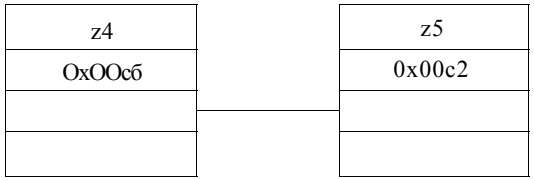


Рис. **1.12.** Многосвязный список (сплетение)

Сплетение — связь элементов, основанная на сплетении указателей. Каждый элемент сплетения может содержать информацию о количестве полей с указателями и формате поля данных. Плексы (сплетения) используются для представления различных семейств связей между индивидуумами и владельцами, отражают производственные, отраслевые связи и т.п.

Контрольные вопросы

1. Что называется структурой данных?
2. Что такое логическая и физическая структура данных?
3. Чем различаются простые и интегрированные структуры данных?
4. Назовите основные особенности статических, полустатических и динамических структур.
5. Перечислите основные типы связных списков.
6. Дайте определение линейных и нелинейных структур.
7. Чем различаются стек, очередь и дек?
8. В чем заключается основная особенность древовидных структур?
9. Приведите примеры графовых структур.

Глава 2
Модели объектов и процессов

При изучении информатики часто приходится заниматься объединением каких-либо сходных по свойствам сущностей *{объектов)* в различные группы (классы) в зависимости от того, обладают они или нет какими-то признаками или свойствами (например, выделяя типы сложных данных или методов описания алгоритмов). Такой прием называется *классификацией,* т.е. распределением однотипных объектов в соответствии с выделенными свойствами (признаками, категориями, классами).

Может оказаться, что у объектов несколько общих свойств, не зависимых друг от друга, тогда распределять их можно по разным классификационным признакам, т.е. проявляется множественность классификаций для одной и той же группы объектов.

Объект — простейшая составляющая сложного объединения, обладающая следующими качествами:

• в рамках данной задачи он не имеет внутреннего устройства и рассматривается как единое целое;

• у него имеется набор свойств (атрибутов), которые изменяются в результате внешних воздействий;

• он идентифицирован, т.е. имеет имя (название).

Например, отдельное предприятие в рамках экономики государства может считаться простым элементом, т.е. объектом, с некоторым набором параметров, для государства существенных: характер и объем выпускаемой продукции, местонахождение, потребности в ресурсном обеспечении, количество работников и т.д. При этом не включаются в рассмотрение структура производства, количество производственных помещений, личности руководителей, цвет зданий и пр.

В другой задаче, где требуется найти оптимальную схему производства конкретного предприятия, рассматривать его как простой элемент, безусловно, нельзя. Проникновение вглубь устройства чего-либо, вообще говоря, безгранично, поэтому всегда приходится останавливаться на некотором «уровне простоты», который приемлем для данной задачи. Таким образом, отнесение каких-то составляющих сложного объединения к объектам есть не что иное, как упрощение реальной ситуации, т.е. моделирование. Объект — модельное

дставление. Выделение объектов из составляющих сложного обънения производится на этапе построения модели при решении гической задачи.

Класс — это множество объектов, обладающих одним или нелькими одинаковыми атрибутами; эти атрибуты называются *полем ств класса.*

,-Среди атрибутов объекта всегда имеются те, которые определяют актер его связей (взаимодействия) с другими объектами, следо-1ьно, оказываются существенными для объединения объектов, ЯЗОборот, часть атрибутов объекта для объединения может быть ' щественной. Например, учебная группа объединяет людей, посту-шйх в одно время в данное учебное заведение; несущественными Ываются рост, цвет глаз и волос и прочие индивидуальные ка-.гва.

'Система — совокупность взаимодействующих компонентов, каиз которых в отдельности не обладает свойствами системы в ***А,*** но является ее неотъемлемой частью.

Человек издавна использует моделирование для исследования объ-:ов, явлений и процессов в различных областях. Моделирование •огает принимать обоснованные и продуманные решения, предгь последствия своей деятельности. Понятие «компьютерное чирование» введено для того, чтобы отразить использование в процессе мощного современного средства переработки инфор-•*м* — компьютера.

Модель — упрощенное представление о реальном объекте, про-Евили явлении. Моделирование — построение моделей для иссле-1ния и изучения объектов, процессов или явлений.

Для чего создавать модель, а не исследовать сам оригинал?

Во-первых, можно моделировать оригинал (прототип), которого не существует или его нет в действительности.

.Во-вторых, оригинал может иметь много свойств и взаимосвязей, (изучения какого-либо свойства иногда полезно отказаться от менее ственных, вовсе не учитывая их.

чюбая классификация начинается с выделения общих свойств йзнаков) и, возможно, определения их значений, по которым объек-.фудут распределяться в различные группы (классы). Представлена ификация может быть в графической форме (в виде графа) и в эвой форме (в виде таблиц или списков).

2.1. Модели структурные и функциональные

Состояние прототипа — это совокупность свойств его составных частей, а также его собственных. Состояние — «моментальная» фотография прототипа для выбранного момента времени. С течением времени состояние может изменяться, тогда говорят о существовании *процесса.* В соответствии со сказанным возможно построение *модели состояния* и *модели процессов.* Модели первого типа называются *структурными,* второго типа—*функциональными моделями.*

Примерами структурных моделей являются чертеж какого-либо устройства, схема компьютера, блок-схема алгоритма и пр. Примером функциональных моделей является макет, демонстрирующий работу чего-либо. Важнейшим классом функциональных моделей являются *имитационные модели.*

Имитационное моделирование — метод исследования, основанный на том, что изучаемый прототип заменяется его имитатором (натурной или информационной моделью), с которым и проводятся эксперименты с целью получения информации об особенностях прототипа.

Примером натурной имитационной установки может служить аэродинамическая труба, позволяющая исследовать воздушные потоки, обтекающие транспортное средство (самолет, автомобиль, тепловоз и пр.) при движении. В качестве имитаторов могут выступать и математические модели, реализованные на компьютере. В настоящее время именно имитационное моделирование оказывается важнейшим методом исследования и прогнозирования в науке, экономике и других отраслях знаний. Примерами являются моделирование последствий ядерной войны, прогноз погоды, экономические прогнозы и пр.

2.2. Модели натурные и информационные

Модели можно отнести к одной из двух групп: натурные риальные) и информационные (нематериальные) — рис. 2.1.

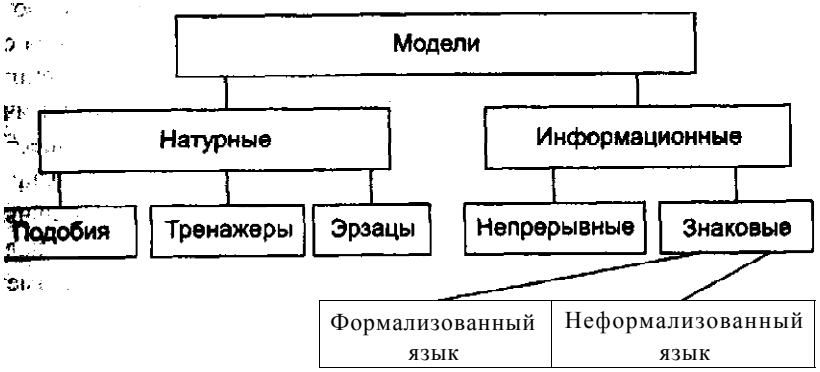


Рис. 2.1. Классификация моделей

Примерами *натурных моделей* подобия являются: игрушка, мафотография и т.п. К натурным моделям-тренажерам следует различные устройства, применяемые при подготовке летчиводителей и других, которые имитируют некоторую ситуацию, 'Ющую принятия решений и действий, и позволяют отрабатывать ii ее разрешения.

!одели-эрзацы — это в первую очередь протезы, заменяющие и 1ЧНО выполняющие функции настоящих органов.

'Материальные модели иначе можно назвать предметными, физикими. Они воспроизводят геометрические и физические свойства 1ала и всегда имеют реальное воплощение. *Информационная ь* -»- совокупность информации, характеризующая свойства и яиие объекта, процесса, явления, а также взаимосвязь с внешним ром. Информационные модели в свою очередь подразделяются на *Шювые* и *непрерывные.*

Примером непрерывной информационной модели могут служить рематическая функция и ее график.

Знаковые информационные модели можно было бы назвать такскретными, подчеркивая то обстоятельство, что информация в

них представлена в дискретной форме, т.е. посредством некот<|рс|го алфавита и языка. Язык может быть обычным разговорным — с *кеформализованным синтаксисом.* Примерами мэоделей, построенных посредством такого языка, являются различные описания, харак' еристики, мнения, толкования и пр.

Формализованные языки имеют фиксирс **вашный набор лекрических** единиц (слов) и жесткий синтаксис фраз- Именно этим **ip6«:с**печивается однозначность понимания смысла, фараз и исполнение **со**держащихся в них указаний. Примером моделей, представленных посредством формализованных языков, мож«гг служить матема]'ическое описание существующего в природе или **чегловеческом обществе** явления, процесса; запись шахматной партии; **ИСУтная запись услы ланных** звуков и пр. Построение знаковой моделр **якляется обязател! ным** этапом решения практической задачи с помощью компьютера; в **iiarb**нейшем, говоря о моделях в информатике, будеш подразумевать и|**мен**но информационные знаковые модели.

Нельзя называть моделями монтажную схезиу нового компь|кУТе ра, блок-схему разрабатываемой программы илти план создаваемого литературного или научного произведения — Правильнее было 6i.1 использовать какой-то иной термин, например ***проеект.*** Если же пр<|дукт творчества обладает прототипом (например, это портрет или пейз aaf **с** натуры), то он может считаться моделью, причегм это могут быт|, как натурные, так и информационные модели.

Рассмотрим графическую форму модели (ршс. 2.2), соответствуй щей следующему словесному описанию: «А у*чится **в** одной группе с ***B*** и ***C,*** но не с ***D*** и ***E,*** которые учатся в другой группе». Э|дерь ***M = {A,B,C,D},*** а отношением ***R*** будет «учичгься в одной группе».

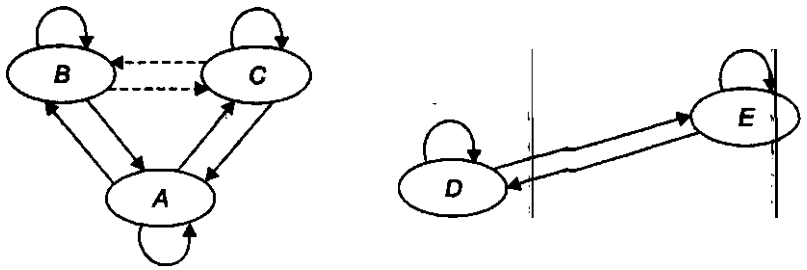


Рис. 2.2. Графическая форма моде: л и

инами графа являются элементы несущего множества, а его ду- — отношения.

ййрчевидно, рассматриваемое отношение транзитивно, что отраже- ^арными пунктирными дугами, связывающими ***B*** и ***C.***

2.3. Классификация моделей

Рассмотрим наиболее распространенные признаки, по ым классифицируются модели:

- область использования;
- учет в модели временного фактора (динамики);
- отрасль знаний;
- способ представления моделей.

Классификация по области использования:

- *учебные модели* — наглядные пособия, различные тренажеры, дающие программы;
- *научно-технические модели* создают для исследования процес- и явлений;
- *игровые модели* — это военные, экономические, спортивные и е игры;
- *имитационные модели* не просто отражают реальность, а ими- ее. Эксперимент либо многократно повторяется, либо про- я одновременно со многими другими похожими объектами, но енными в разные условия.

Классификация с учетом фактора времени. Модели можно раз- fa на *статические* (это как бы одномоментный срез информации кту) и *динамические.* Динамическая модель позволяет увидеть ния объекта во времени.

смотрим схему классификации моделей по способу представ- **шие.** 2.3).

'бальная модель — информационная модель в мысленной или «ой форме. К таким моделям можно отнести и идею, возник- иэобретателя, и музыкальную тему, и рифму, прозвучавшую *Ще в сознании у автора.

"Чековая модель — информационная модель, выраженная спеыми знаками, т.е. средствами любого формального языка (ри-

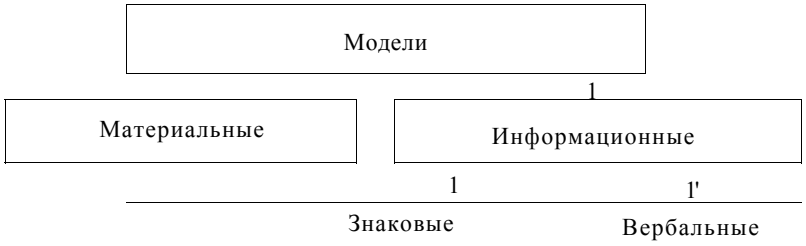


Рис. 2.3. Классификация моделей по способу представления

сунки, тексты, графики, схемы). По форме представления можно выделить следующие виды информационных моделей:

• *геометрические* — графические формы и объемные конструкции;

• *словесные* — устные и письменные описания с использованием иллюстраций;

• *математические* — математические формулы, отображающие связь различных параметров объекта или процесса;

• *структурные* — схемы, графики, таблицы и т.п.;

• *логические* — модели, в которых представлены различные варианты выбора действий на основе умозаключений и анализа условий;

• *специальные* — ноты, химические формулы и т.п.;

• *компьютерные* и *некомпьютерные модели.*

2.4. Этапы моделирования

Моделирование является одним из ключевых видов деятельности человека. Оно всегда в той или иной форме предшествует любому делу. Моделирование занимает центральное место в исследовании объекта. Оно позволяет обоснованно принимать решение. Решение любой задачи разбивается на несколько этапов. Моделирование — творческий процесс и заключить его в формальные рамки очень трудно. В общем виде его можно представить поэтапно (рис. 2.4).

Все этапы определяются поставленной задачей и целями моделирования. Рассмотрим основные этапы моделирования подробнее.

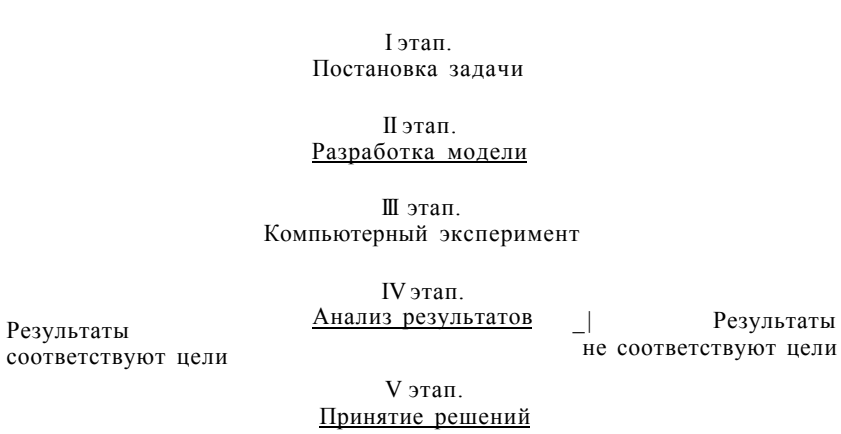


Рис. 2.4. Этапы моделирования

Этап I. *Постановка задачи.* Описание задачи. Определение цели моделирования. Анализ объекта моделирования.

Этап II. *Разработка модели.* На этом этапе выясняются свойства, состояния, действия и другие характеристики элементарных объектов. Формируется представление об элементарных объектах. Выбор наиболее существенной информации при создании информационной модели и ее сложность обусловлены целью моделирования.

Этап III.*Компьютерный эксперимент.* Тестирование — процесс проверки правильности модели.

Этап IV. *Анализ результатов моделирования.* Принятие решения, которое должно быть выработано на основе всестороннего анализа полученных результатов.

Этап V. *Конечная цель моделирования.*

2.5. Свойства алгоритма

Алгоритмом называется система формальных правил, четко и однозначно определяющая процесс решения поставленной задачи в виде конечной последовательности действий или операций.

Свойства, которыми должен обладать алгоритм:

1. *Конечность (финитностъ) алгоритма.* Алгоритм должен приводить к решению задачи обязательно за конечное время. Последова-

тельность правил, приведшая к бесконечному циклу, алгоритмом не является.

2. *Определенность,* или *детерминированность, алгоритма.* 'Это свойство означает, что неоднозначность толкования записи алгоритма недопустима.

3. *Результативность алгоритма.* Под результативностью понимается доступность результата решения задачи для пользователя, иными словами, алгоритм должен обеспечить выдачу результата решения задачи на печать, на экран монитора или в файл.

4. *Массовость алгоритма.* Это означает, если правильный результат по алгоритму получен для одних исходных данных, то правильн ый результат по этому же алгоритму должен быть получен и для других исходных данных, допустимых в данной задаче.

5. *Эффективность алгоритма.* Под эффективностью алгоритма будем понимать такое его свойство (качество), которое позволяет решить задачу за приемлемое для разработчика время. К параметру, характеризующему эффективность алгоритма, следует отнести также объем памяти компьютера, необходимый для решения задачи.

2.6. Виды алгоритмов и их реализация

В зависимости от цели, начальных условий задачи, путей ее решения, определения действий разработчика алгоритмы подразде ляются на механические, или детерминированные (жесткие), и гибкие, или стохастические (вероятностные и эвристические).

Механический алгоритм задает определенные действия, обозначая их в единственной последовательности, обеспечивающей однозначный требуемый (искомый) результат в том случае, если выполняются условия процесса, для которых разработан алгоритм. К таким алгоритмам относятся алгоритмы работы машин, станков, двигателей и т.п.

Вероятностный {стохастический) алгоритм предлагает програм му решения задачи несколькими путями или способами, приводящими к достижению результата.

Эвристический алгоритм (от греческого слова «эврика»)—это **тк**кой алгоритм, в котором достижение конечного результата однозначно

не определено, так же как не обозначена вся последовательность действий. В этих алгоритмах используются универсальные логические процедуры и способы принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения похожих задач. При реализации эвристических алгоритмов большую роль играет интуиция разработчика.

В программировании алгоритмы подразделяются на три типа:

• *линейный* — набор команд (указаний), выполняемых последовательно друг за другом;

• *разветвляющийся* — алгоритм, содержащий хотя бы одну проверку условия, в результате которой обеспечивается переход на один из возможных вариантов решения;

• *циклический* — алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений и перебора вариантов.

Вспомогательный (подчиненный) алгоритм—это алгоритм, ранее разработанный и целиком используемый при алгоритмизации конкретной задачи.

В зависимости от степени детализации, поставленных целей, методов и технических средств решения задачи используются различные формы представления алгоритмов. Все варианты представления алгоритмов могут быть объединены в единую классификационную схему, изображенную на рис. 2.5.

На практике наиболее распространены следующие способы:

• словесный;
• формульно-словесный;
• блок-схемный;
• псевдокод;
• структурные диаграммы;
• языки программирования.

Словесный способ — содержание этапов вычислений задается на естественном языке в произвольной форме с требуемой детализацией.

Например, пусть задан массив чисел. Требуется проверить, все ли числа принадлежат заданному отрезку, который задается границами *A а B.*

Ш а г 1. Выбираем первый элемент массива — к шагу 2.

Способы представления алгоритмов

Естественное представление Формальное представление

Графическая Строчная (языковая) Алгоритмические системы

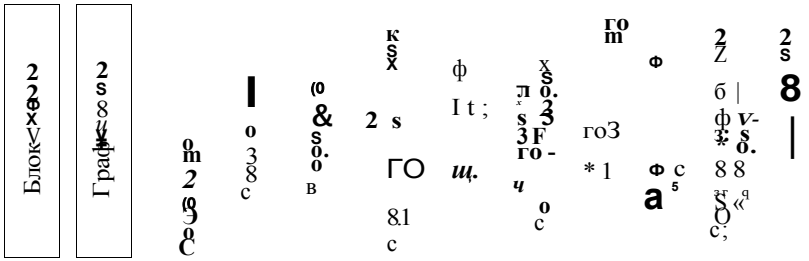


Рис. 2.5. Классификация способов представления алгоритмов

Ш а г 2. Сравниваем: выбранный элемент массива принадлежу**нт** интервалу — если «да», то к шагу 3, если «нет» — к шагу 6.

Ш а г 3. Все элементы массива просмотрены? Если «да» —тс| шагу 5, если «нет» — то к шагу 4.

Ш а г 4. Выбираем следующий элемент — к шагу 2.

Ш а г 5. Печать сообщения: все элементы принадлежат интер||а лу — на шаг 7.

Ш а г 6. Печать сообщения: не все элементы принадлежат интф валу — на шаг 7.

Ш а г 7. Конец.

При этом способе записи алгоритма отсутствует наглядность й| числительного процесса, так как нет достаточной формализации.

Формульно-словесный способ — задание инструкций с использованием математических символов и выражений в сочетании to словесными пояснениями.

Например, вычислить площадь треугольника по трем сторонам' *Ь, с.* Данный алгоритм может быть представлен следующим образоА

Ш а г 1. Вычислить полупериметр треугольника ***p = (a*** + ***b*** + с)j

Ш а г 2. Вычислить ***S = y/p{p — a){p*** *Ь)(р-с).*

Ш а г 3. Напечатать результат ***S*** и прекратить вычисления.

При использовании этого способа может быть достигнута любая степень детализации более наглядно, но не строго формализованно.

Блок-схемный способ—это графическое изображение алгоритма, в котором каждый этап процесса обработки данных представляется в виде геометрических фигур (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций. В табл. 2.1 приведены наиболее часто употребляемые блоки.

Таблица 2.1

Геометрические фигуры блок-схем

Название символа	Обозначение	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение	o	Блок проверки условия, имеющий один вход и ряд альтернативных выходов, один из которых может быть активизирован после выполнения условий
Модификация	< >	Модификация команды или группы команд с целью воздействия на некоторую последующую функцию
Предопределенный процесс		**Процесс, состоящий из одной или нескольких операций (шагов) подпрограммы или модуля**
Ввод-вывод	/ /	Ввод-вывод информации, при котором отсутствует необходимость в описании фактического носителя данных
Пуск-останов	( )	Начало или конец алгоритма, вход (выход) подпрограммы
Линии потока данных	——●	Отображает поток данных или управления (при необходимости могут быть добавлены стрелки-указатели)
Соединитель	O	Используется для обрыва линии и продолжения ее в другом месте блок-схемы
Комментарий	- С	Символ используют для добавления комментариев или пояснительных записей

Рассмотрим пример блок-схемы задачи, для которой ранее представлен словесный алгоритм (рис. 2.6).

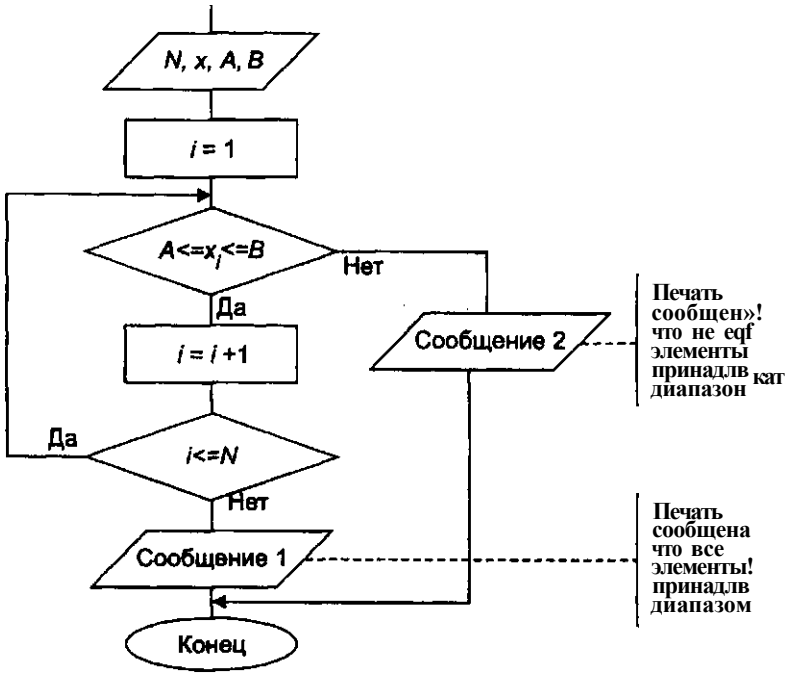


Рис. 2.6. Блок-схема алгоритма

Псевдокод представляет собой совокупность операторов тыка программирования и естественного языка. Запись алгоритма в|виде псевдокода представлена ниже:

Выбираем первый элемент (г = 1)
IF ***A*** > *Xi* или ***xi>*** *B* THEN печать сообщения и выход на коней!
ELSE переход к следующему элементу
IF массив не кончился THEN переход на проверку интервала
ELSE печать сообщения, что все элементы входят в интефал
Конец

При записи алгоритма на псевдокоде каждое отдельное ***пщ*** Пложение может начинаться со звездочки (*). Алгоритм строится хим образом, что разбиение продолжается до тех пор, пока каждый шаг алгоритма не станет достаточно понятным.

Пример. Вычислить при заданном ***x***

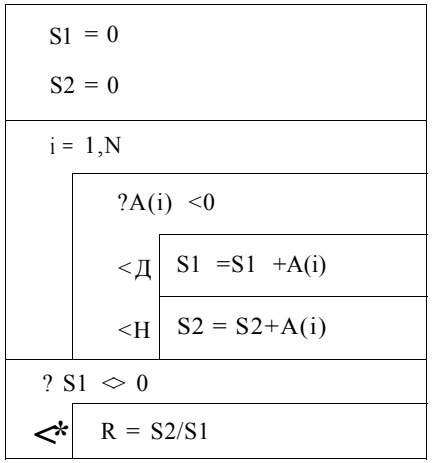
$$y = \begin{cases} & \text{если } x < 0; \\ x + 1, & \text{если } x < 0. \end{cases}$$

Решение.

Ввод (x)
Если *x* < 0 то
* *y* := *x* * *x*
иначе
* *y* := *x* + 1
конец проверки условия
вывод *(y)*
конец алгоритма

Структурные диаграммы могут использоваться в качестве структурных блок-схем, для показа межмодульных связей, для отображения структур данных и систем обработки данных. Существуют следующие структурные диаграммы: диаграммы Насси — Шнейдермана, Варнье, Джексона, МЭСИД и др.

Пример. Задан одномерный массив из положительных и отрицательных чисел. Требуется определить частное от деления суммы положительных элементов на сумму отрицательных элементов этого массива. Справа от диаграммы (рис. 2.7) приводятся соответствующие операторы языка Turbo Pascal.




```
... S1 :=0;
    S2 := 0;
    FOR I := 1 TO N DO
        IF A[I] < 0 THEN
            51 :=S1 +A[I]
        ELSE
            52 := S2+A[I];
    IFS1 <> 0 THEN
        R := S2/S1; ...
```

Рис. 2.7. Диаграмма МЭСИД

2.7. Базовые канонические структуры алгоритмов

Для реализации алгоритмов на ПЭВМ используете i алгоритмический язык — набор символов и правил образования и истолкования конструкций из этих символов для записи алгоритп ов

Любую программу можно написать, используя комбинации frpex базовых структур:

- следования, или последовательности операторов;
- ветвления, или условного оператора;
- повторения, или оператора цикла.

Программа, составленная из канонических структур, называется *регулярной программой,* т.е. имеющей один вход и один выход

Поскольку алгоритм определяет порядок обработки информации он должен содержать, с одной стороны, действия по обработке, а с другой — порядок их следования, называемый *потоком управления.*

Рассмотренные выше блоки, связанные с обработкой данных], делятся на *простые* и *условные.* Особенность простого действия в том, что оно имеет один вход и один выход, в отличие от условного, обладающего двумя выходами в зависимости от того, истинным ли окажется условие. Простое действие не означает, что оно единственное, - это может быть некоторая последовательность действий.

Часть алгоритма, организованная как простое действие, те имеющая один вход (выполнение начинается всегда с одного и топЬ же действия) и один выход (т.е. после завершения данного блока всегда начинает выполняться одно и то же действие), называется *функциональным блоком.*

Из этого определения, в частности, следует, что каждое пра|стое действие является функциональным-блоком, а условное — нет.

Согласно положениям структурного программирования можнс| выделить всего три различных варианта организации потока управления действиями алгоритма. Поток управления может обладать следук щими свойствами:

- каждый блок выполняется не более одного раза;
- выполняется каждый блок.

Поток управления, в котором выполняются оба эти свойства, называется *линейным;* в нем несколько функциональных блоков выполняются последовательно. Линейному потоку на языке блок-схем соответствует структура, показанная на рис. 2.8.

Рис. 2.8. Линейный поток управления

Очевидно несколько блоков, связанных линейным потоком управления, могут быть объединены в один функциональный блок (рис. 2.9).

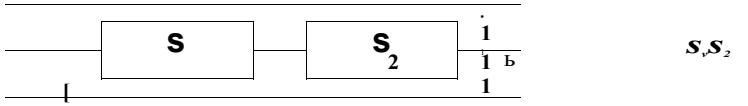


Рис. 2.9. Функциональный блок

Второй тип потока управления называется *ветвлением* — он организует выполнение одного из двух функциональных блоков в зависимости от значения проверяемого логического условия. Проверка *Р* представляется *предикатом,* т.е. функцией, задающей логическое выражение или условие, значением которого может быть *истина* или *ложь.*

Если структура содержит два функциональных блока *(Si* и 5г), ветвление называется *полным* (рис. 2.10); возможно существование *неполного* ветвления — при этом один блок отсутствует (рис. 2.11).

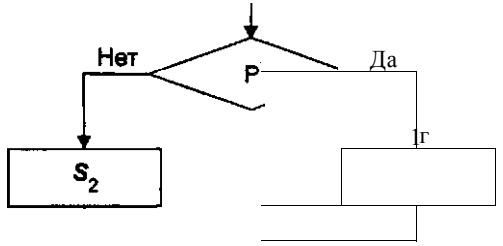


Рис. 2.10. Полное ветвление

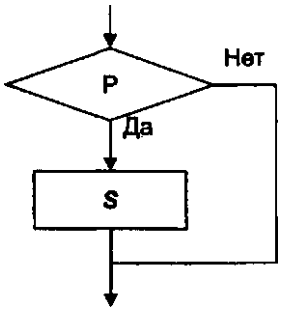


Рис. 2.11. Неполное ветвление

Переключатель — выбор одного варианта из множества возможных альтернатив. В зависимости от значения *P* выполняется одно из действий *A,B,..., Z* (рис. 2.12). После выбора варианта происходит переход к выполнению следующей управляющей структуры.

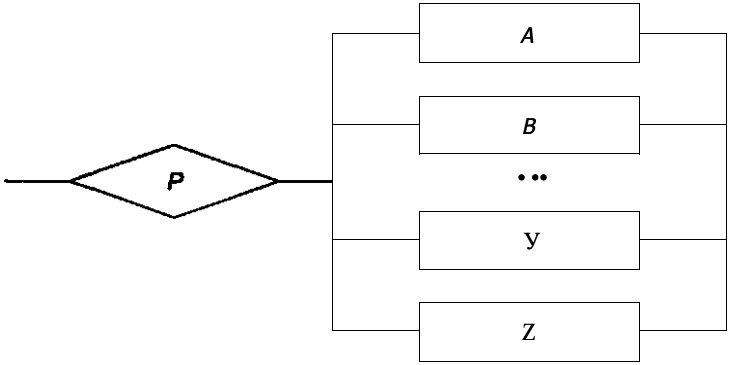


Рис. 2.12. Схема переключателя

Повторение — многократное выполнение фрагментов алгоритма (программы). Такой тип потока управления называется *циклическим* — он организует многократное повторение функционального блока, пока логическое условие его выполнения является истинным.

Действие *A* будет повторяться до тех пор, пока значение предиката *P* будет оставаться истинным (рис. 2.13). Поэтому в действии *A* должны изменяться значения переменных, от которых зависит *P,* в противном случае произойдет зацикливание.

Ложь

Рис. 2.13. Схема цикла с предусловием

Вычисление предиката *P* может производиться после выполнения действия *A,* в этом случае действие *A* будет выполняться хотя бы один раз (рис. 2.14).

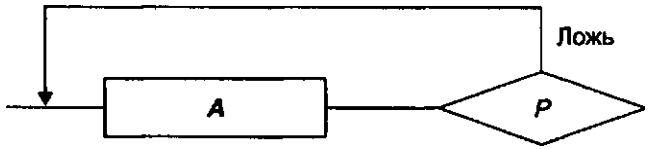


Рис. 2.14. Схема цикла с постусловием

Алгоритм называется *структурным,* если он представляет собой комбинацию из трех рассмотренных выше структур (они называются *базовыми алгоритмическими структурами).* Безусловно, не все алгоритмы являются структурными. Однако именно структурные алгоритмы обладают рядом замечательных преимуществ по сравнению с неструктурными:

• *понятность* и *простота* восприятия алгоритма (поскольку невелико число исходных структур, которыми он образован);

• *проверяемость* (для проверки любой из основных структур достаточно убедиться в правильности входящих в нее функциональных блоков);

• *модифицируемость* (она состоит в простоте изменения структуры алгоритма, поскольку составляющие блоки относительно независимы).

После введенных определений можно сформулировать структурную теорему Бома — Джакопини: *любой алгоритм может быть \рведен к структурному.* Иными словами, *любому неструктурному ^алгоритму может быть построен эквивалентный ему структурный {алгоритм.*

2.8.
Полное построение алгоритма

В последние годы большое распространение по [учила концепция структурного программирования. Понятие струю' рного программирования включает определенные принципы проек! фования, кодирования, тестирования и документирования про] мм в соответствии с заранее определенной жесткой дисциплиной

Полное построение алгоритма предусматривает последова |льное выполнение следующих этапов:

1) постановка задачи;

2) построение модели;

3) разработка алгоритма;

4) проверка правильности алгоритма;

5) реализация, т.е. программирование алгоритма;

6) анализ алгоритма и его сложности;

7) проверка (отладка) программы;

8) составление документации.

Не все эти этапы четко различимы между собой, особенно жа раз личимость делается мало заметной при программировании пшостых задач (рис. 2.15). При программировании простых задач нем орые этапы могут вообще не выполняться — настолько очевидны ^ хре- зультаты.

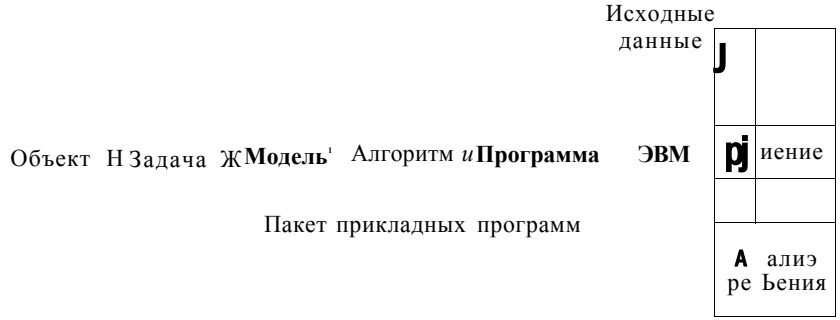


Рис. 2.15. Этапы решения задач на ЭВМ

При программировании сложных, объемных задач некоторые из вышеперечисленных этапов приходится выполнять не в том порядке, как выше указано, или выполнять их не один раз.

Рассмотрим более подробно каждый из этапов построения алгоритма.

Постановка задачи. Прежде чем понять задачу, ее нужно точно сформулировать. Это условие само по себе не является достаточным для понимания задачи, но оно абсолютно необходимо.

Обычно процесс точной формулировки задачи сводится к постановке правильных вопросов. Перечислим некоторые полезные вопросы для плохо сформулированных задач:

- Понятна ли терминология, используемая в предварительной формулировке?
- Что дано? Что нужно найти?
- Как определить решение?
- Каких данных не хватает или, наоборот, все ли перечисленные в формулировке задачи данные используются?
- Какие сделаны допущения?

Возможны и другие вопросы, возникающие в зависимости от конкретной задачи.

Построение модели. Задача четко поставлена, теперь нужно сформулировать для нее математическую модель. Выбор модели , существенно влияет на остальные этапы в процессе решения.

Выбор модели — в большей степени искусство, чем наука. Правда, если вы будете встречаться с типовой задачей, то опыт, приобретенный ранее, не потребует от вас особого творческого подхода, и в этом случае как раз будет удобно и целесообразно воспользоваться ранее наработанными правилами. Поэтому изучение удачных моделей — • это наилучший способ приобрести опыт в моделировании.

Приступая к разработке модели, следует задать, по крайней мере, два основных вопроса:

1. Какие математические структуры больше всего подходят для аадачи?

2. Существуют ли решенные аналогичные задачи?

Второй вопрос, возможно, самый полезный во всей математике. В ^юнтексте моделирования он часто дает ответ на первый вопрос. Дей- таительно, большинство решаемых в математике задач, как правило, :вляются модификациями ранее решенных.

Сначала нужно рассмотреть первый вопрос. Мы должны опи математически, что мы знаем и что хотим найти. На выбор соот ствующей структуры будут оказывать влияние следующие фактор!

• ограниченность наших знаний относительно небольшим кс чеством структур;

• удобство представления;

• простота вычислений;

• полезность различных операций, связанных с рассматривав! структурой или структурами.

Сделав пробный выбор математической структуры, задачу с л £ у ет переформулировать в терминах соответствующих математич< объектов. Это будет одна из возможных моделей, если мы м утвердительно ответить на вопросы:

• Вся ли важная информация задачи хорошо описана матема скими объектами?

• Существует ли математическая величина, ассоциируемая комым результатом?

• Выявили ли мы какие-нибудь полезные отношения между ектами модели?

• Можем ли мы работать с моделью? Удобно ли с ней работ;

Разработка **алгоритма.** Выбор метода разработки алгоритм частую сильно зависит от выбора модели и может в значите: степени повлиять на эффективность алгоритма решения. Два ра ных алгоритма могут быть правильными, но очень сильно отлич по эффективности.

Правильность алгоритма. Доказательство правильности ритма — это один из наиболее трудных, а иногда и особеннс мительных этапов создания алгоритма. Вероятно, наиболее ра страненный прием доказательства правильности программы - прогон ее на разных тестах. Если выданные программой ответы і быть подтверждены известными или вычисленными вручную д! ми, возникает искушение сделать вьтод, что программа «рабе правильно. Однако этот метод редко исключает все сомнения; » существовать случай, в котором программа не работает.

Рассмотрим следующую общую методику доказательства вильности алгоритма. Предположим, что алгоритм описан в последовательности шагов, допустим, от шага 0 до шага ***m.*** Г раемся предложить некое обоснование правомерности для ка:

шага. В частности, может потребоваться формулировка утверждения об условиях, действующих до и после пройденного шага. Затем постараемся предложить доказательство конечности алгоритма, при этом будут проверены *все* подходящие входные данные и получены *все* подходящие выходные данные.

Другой метод доказательства правильности алгоритма, который не имеет специального названия, состоит в следующем. Для каждого цикла, который имеется в программе (алгоритме), вручную (например, на калькуляторе) подсчитываются две контрольные точки. Если контрольные точки совпадают со значениями, выданными программой, можно быть уверенным, что все циклы в программе работают правильно. Почему речь идет о двух контрольных точках? Дело в том, что первое контрольное значение в программе может быть вычислено правильно, а затем в этом цикле будут произведены некоторые некорректные действия, которые приведут к искажению всех последующих результатов. Совпадение второго контрольного значения как раз и подтверждает, что в данном цикле некорректности нет. Таким образом, два контрольных вычисления должны быть сделаны для каждого цикла программы.

Следует подчеркнуть и тот факт, что правильность алгоритма еще ничего не говорит о его эффективности. В этом смысле исчерпывающие алгоритмы, или, как их еще называют, алгоритмы полного перебора, редко бывают хорошими во всех отношениях.

Реализация алгоритма. Как только алгоритм выражен, допустим, в виде последовательности шагов и мы убедились в его правильности, настает черед реализации алгоритма, т.е. написания программы для компьютера.

При этом возникают следующие проблемы.

• Очень часто отдельно взятый шаг алгоритма может быть выражен в форме, которую трудно перевести непосредственно в конструкции языка программирования. Например, один из шагов алгоритма может быть записан в виде, требующем целой подпрограммы для своей реализации.

• Реализация может оказаться трудным процессом потому, что перед тем, как написать программу, необходимо построить целую систему структур данных для представления важных аспектов используемой модели.

Чтобы сделать это, необходимо ответить, например, на такие вопросы:

Каковы основные переменные?

Каких они типов?

Сколько нужно массивов и какой размерности?

Имеет ли смысл пользоваться связными списками?

Какие нужны подпрограммы (возможно, уже записанные в памяти)?

Каким языком программирования пользоваться?

Конкретная реализация может существенно влиять на требования к памяти и на скорость алгоритма.

Заметим, что одно дело — доказать правильность конкретного алгоритма, описанного в словесной форме, другое — доказать, данная машинная программа, предположительно являющаяся реализацией этого алгоритма, также правильна. Поэтому необходимо очень тщательно следить, чтобы процесс преобразования правильного ритма (в словесной форме или форме схемы алгоритма) в программу, написанную на алгоритмическом языке, заслуживал доверия.

2.9. Главные принципы создания эффективных алгоритмов

Каждый, кто занимается разработкой алгоритмов, должен овладеть некоторыми основными методами и понятиями. Перед тем, кто когда-то столкнулся с трудной задачей, вставал вопрос: «С чего начать?». Один из возможных путей — просмотреть свой запас общих алгоритмических методов для того, чтобы проверить, нельзя ли с помощью одного из них сформулировать решение новой задачи. Ну, а если такого запаса нет, то как все-таки разработать хороший алгоритм? Рассмотрим три общих метода решения задач, полезных для разработки алгоритмов.

Первый метод связан со сведением трудной задачи к последовательности более простых задач. Конечно, мы надеемся на то, что более простые задачи легче поддаются обработке, чем первоначальная задача, а также на то, что решение первоначальной задачи может быть получено из решений этих более простых задач. Такая процедура называется *методом частных целей.*

Этот метод выглядит очень разумно. Но, как и большинство общих методов решения задач или разработки алгоритмов, его не всегда легко перенести на конкретную задачу. Осмысленный выбор более простых задач — скорее, искусство или интуиция, чем наука. Не существует общего набора правил для определения класса задач, которые можно решать с помощью такого подхода. Размышление над любой конкретной задачей начинается с постановки вопросов. Частные цели могут быть установлены, когда получены ответы на следующие вопросы:

1. Можем ли мы решить часть задачи? Можно ли, игнорируя некоторые условия, решить оставшуюся часть задачи?

2. Можем ли мы решить задачу для частных случаев? Можно ли разработать алгоритм, который дает решение, удовлетворяющее всем условиям задачи, но входные данные которого ограничены некоторым подмножеством всех входных данных?

3. Есть ли что-то, относящееся к задаче, что мы недостаточно хорошо поняли? Если попытаться глубже вникнуть в некоторые особенности задачи, сможем ли мы что-то узнать, что поможет нам подойти к решению?

4. Встречались ли мы с похожей задачей, решение которой известно? Можно ли видоизменить ее решение для решения нашей задачи? Возможно ли, что эта задача эквивалентна известной нерешенной задаче?

Второй метод разработки алгоритмов известен как *метод подъема.* Алгоритм подъема начинается с принятия начального предположения или вычисления начального решения задачи. Затем начинается насколько возможно быстрое движение «вверх» от начального решения по направлению к лучшим решениям. Когда алгоритм достигнет такой точки, из которой больше невозможно двигаться наверх, алгоритм останавливается. К сожалению, мы не можем всегда гарантировать, что окончательное решение, полученное с помощью алгоритма подъема, будет оптимальным. Эта ситуация часто ограничивает применение метода подъема.

Вообще методы подъема являются «грубыми». Они запоминают некоторую цель и стараются сделать все, что могут и где могут, чтобы подойти ближе к цели. Это делает их несколько недальновидными. Недальновидность метода подъема хорошо иллюстрируется следующим примером. Пусть требуется найти максимум функции $y = /(ж)$, представленной графиком (рис. 2.16).

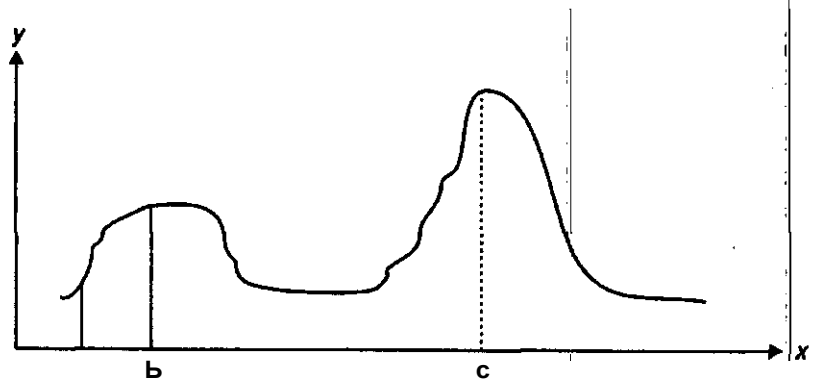


Рис. 2.16. Иллюстрация метода подъема

Если начальное значение аргумента x равна a, то метод подъема даст устремление к ближайшей цели, т.е. к значению функции в то)чке $x = b$, тогда как подлинный максимум этой функции находится $x = c$. В данном случае метод подъема находит локальный максифм, но не глобальный. В этом и заключается «грубость» метода **подъема.**

Третий метод известен как *отрабатывание назад,* т.е. i бота этого, алгоритма начинается с цели или решения задачи и затем осуществляетсядвижение к начальной постановке задачи. Затем, е|;ли эти действия обратимы, производится движение обратно от постановки задачи к решению.

Контрольные вопросы

1. Дайте определение объекта, класса, системы, Цодели.
2. Назовите основные типы моделей.
3. Что такое имитационное моделирование?
4. Какие классификации моделей существуют?
5. Укажите основные этапы моделирования.
6. Что такое алгоритм?
7. Перечислите свойства алгоритма.
8. Какие этапы выполняются при полном построении алгоритма?
9. Что такое блок-схема алгоритма?
10. Дайте определение функционального блока.
11. Какой алгоритм называется структурным?
12. Назовите главные принципы, лежащие в основе создания эффективных алгоритмов.

Глава 3
Эволюция языков программирования

Для преобразования текста в последовательность машинных команд необходима промежуточная программа, называемая *компилятором.* На этапе компиляции производится также распределение данных в оперативном запоминающем устройстве (ОЗУ) ПЭВМ; при этом вместо имен переменных подставляются относительные адреса ячеек, в которых располагаются данные.

Абсолютные адреса данным присваивает операционная система при размещении программы в ОЗУ компьютера перед ее использованием.

Для всех языков высокого уровня общим является то, что они ориентированы не на систему команд той или иной машины, а на систему операторов, характерных для записи определенного класса **Алгоритмов.**

3.1. Классификация языков программирования по функциональному назначению

По функциональному назначению языки программирования высокого уровня делят на *проблемно-ориентированные чуниверсальные.* Первые предназначены для решения специфических задач некоторой отрасли знаний. Примерами являются: *FORTRAN* — ~ык решения сложных научных и инженерных задач; *COBOL* — язык решения экономических и коммерческих задач; *LISP* — язык, спользуемый в решении задач искусственного интеллекта. К универсальным языкам относятся *PASCAL, BASIC, C* (C++), *Java,* а ке современные среды визуального программирования *DELPHI, 'SUAL BASIC* и др. Эти языки позволяют решить любую задачу, хотрудоемкость решения конкретной задачи в разных языках будет 1ичаться.

3.2. Классификация языков программирования по парадигме (концепции) и методологии программирования

Другой признак, в соответствии с которым возможна классификация языков программирования, является *парадигма* (концепция) программирования (табл. 3.1), т.е. совокупность основополагающих идей и подходов, определяющих модель представления данных и их обработки, а также методологии программирования.

Таблица 3.1

Основные различия в парадигмах

Парадигма программирования	Представление программ и данных	Исполнение программы	Связь частей программы между собой
Процедурное	Программа и данные представляют собой отдельные, не связанные друг с другом элементы	Последовательное выполнение операторов	Возможна только через совместно обрабатываемые данные
Объектно-ориентированное	Данные и методы их обработки инкапсулированы в рамках единого объекта	Последовательность событий и реакций объектов на эти события	Отдельные части программы могут наследовать методы и элементы данных друг у друга
Логическое	Данные и правила их обработки объединены в рамках единого логического структурного образования	Преобразование логического образования в соответствии с логическими правилами	Разбиение программы на отдельные независимые части затруднительно

Известно, что информационные технологии являются одной из наиболее быстро развивающихся областей современной жизни. Новые технологии, проекты, названия и аббревиатуры появляются едва ли не каждый день. В настоящее время в распоряжении программиста имеется довольно обширный спектр языков-инструментов, из которых для любой конкретной задачи можно выбрать тот, что позволит решить ее оптимальным путем.

Язык программирования — набор ключевых слов (словарь) и система правил (грамматических и синтаксических) для конструирования операторов, состоящих из групп или строк чисел, букв, знаков

препинания и других символов, с помощью которых люди могут сообщать компьютеру набор команд. Хронология создания языков программирования представлена в табл. 3.2.

Таблица 3.2

Хронология создания языков программирования

Язык	Год создания	Вид	Автор	География создания
Фортран *(Fortran)*	1954	*A*	Джон Бэкус	Америка
Лисп *(LISP)*	1958	*F*	Джон Маккарти	Америка
Алгол-60 *(Algol 60)*	1960	*A*	Питер Наур+	Международный
Кобол *(Cobol)*	1960	*A*	Группа авторов	Международный
Симула *(Simula)*	1962	*B*	Кристен Нигаард+	Европа
Бейсик *(Basic)*	1963	*A*	Джон Кемени+	Америка
ПЛЛ *(PL/I)*	1964	*A*	Джордж Радин	Америка
Алгол-68	1968	*A*	Адван Вайнгартен+	Международный
Паскаль *(Pascal)*	1971	*C*	Никлаус Вирт	Европа
Форт *(FORTH)*	1970	*A**	Чарльз Мур	Америка
lСи (С)	1972	*C**	Деннис Ритчи	Америка
Smalltalk	1972	*B*	Алан Кей	Америка
Пролог *(Prolog)*	1973	*E*	Алан Кольмеро+	Европа
Ада *(Ada)*	1980	*H**	Джин Ишбиа+	Америка
Си-н-	1984	*н**	Бьорн Страуструп	Америка
Java	1995	*н*	Джеймс Гослинг	Америка
АПЛ *(APL)*	1957	*I*	Кеннет Айверсон	Америка
Снобол *(Snobol)*	1962	*I*	Ральф Грисуолд	Америка
Сетл *(SETL)*	1969	*I*	Джек Шварц	Америка
Scheme	1975	*F*	Гай Стил+	Америка
Icon	1977	*I*	Ральф Грисуолд	Америка
Модула-2 *(Modula-2)*	1979	*D**	Никлаус Вирт	Европа
Оккам *(Occam)*	1982	*G**	Дэвид Мэй+	Европа
Common *Lisp*	1984	*F*	Гай Стил+	Америка
Objective C	1986	*H**	Брэд Кокс	Америка
Оберон *(Oberon)*	1988	*D**	Никлаус Вирт	Европа
Модула-3 *(Modula-3)*	1988	*H**	Билл Калсов+	Америка
Limbo	1996	*D**	Деннис Ритчи	Америка
; ic#	2000	*H**	Андерс Хейльсберг+	Америка

Условные обозначения:
A — процедурное программирование;
B — объектно-ориентированное программирование;
C — структурное программирование;
D — модульное (компонентное) программирование;
E — логическое (реляционное) программирование;
F — функциональное программирование;
G — параллельное программирование;
H — смесь парадигм: *B + C + D + G;*
I — специализированные языки;
* — поддержка системного программирования;
\+ — язык программирования создан несколькими авторами.

3.3. Классификация языков программирования по типам задач

Языки программирования можно классифицировать по типам задач следующим образом:

• *Задачи искусственного интеллекта — Lisp, Prolog, Multilisp, Commonlisp, Рефал, Planner, QA4, FRL, KRL, Qlisp;*

• *Параллельные вычисления — Fun, Apl, Alfl, PARAlfl, ML, SML, PPL/1, Hope, Miranda, Occam, PFOR, Glypnir, Actus,* параллельный *Cobol,* ОВС-ЛЯПИС, ОВС-Мнемокод, ОВС-Алгол, ОВС-Фортран, РА(1), РА(G);

• *Задачи вычислительной математики и физики — Occam, PFOR, Glypnir, Actus,* параллельный *Cobol,* ОВС-ЛЯПИС, ОВС-Мнемокод, ОВС-Алгол, ОВС-Фортран, РА(1), РА(G);

• *Разработка интерфейса — Forth,* С, С++, Ассемблер, Макроассемблер, *Simula-67,* ОАК, *Smalltalk, Java,* РПГ;

• *Разработка программ-оболочек, разработка систем — Forth, C,* С++, Ассемблер, Макроассемблер, Simula-67, *OAK, Smalltalk, Java,* РПГ;

• *Задачи вычислительного характера — Algol, Fortran, Cobol, Ada, PL/1,* Фокал, *Basic, Pascal;*

• *Оформление документов, обработка больших текстовых файлов, организация виртуальных трехмерных интерфейсов в Интернете, разработка баз данных —* HTML, *Perl, Tcl/Tk, VRML, SQL, PL/SCL, Informix 4GL, Natural, DDL, DSDL, SEQUEL, QBE, ISBL.*

* * *

В настоящее время существуют программно-аппаратные комплексы, позволяющие организовать параллельное выполнение различных частей одного и того же вычислительного процесса.

В основе функционального программирования лежит представление программы в виде математических функций. Существуют языки с ленивой и с энергичной семантикой: в языках с энергичной семантикой вычисления производятся в том же месте, где они описаны, а в случае ленивой семантики вычисление производится только тогда, когда оно необходимо. Программы на языках логического программирования выражены как формулы математической логики, а компилятор получает следствия из них.

В последнее время в связи с развитием интернет-технологий получили распространение скриптовые языки. Характерными особенностями данных языков являются их интерпретируемость, простота синтаксиса и легкая расширяемость. Они идеально подходят для использования в часто изменяемых программах или в случаях, когда для выполнения операторов языка затрачивается время, несопоставимое с временем их разбора.

Контрольные вопросы

1. Дайте определение алгоритмического языка.
2. Что такое компилятор?
3. Классифицируйте языки программирования.
4. Что такое парадигма?
5. Назовите первый алгоритмический язык.
6. Перечислите парадигмы языков программирования.
7. В чем заключена основная особенность объектно-ориентированных языков?
8. Назовите языки для обработки данных.
9. Назовите Pascal- и С-подобные языки.
10. Какие алгоритмические языки используются для решения задач математики и физики? Для решения задач вычислительного характера? Для решения задач искусственного интеллекта?
11. Чем различаются языки с ленивой и энергичной семантикой?
12. Назовите характерные особенности скриптовых языков.

Глава 4 Функция сложности алгоритма

Существует ряд важных причин для анализа алгоритмов. Одной из них является необходимость получения оценок или границ для объема памяти или времени работы, которое потребуется алгоритму для успешной обработки конкретных данных.

Для оценки качества алгоритма вводится понятие *сложность алгоритма,* или обратное понятие—*эффективность алгоритма.* Чем

большее время и объем памяти требуются для реализации алгоритма, тем больше его сложность и соответственно ниже эффективность. Сложность алгоритма делится на временную и емкостную. Временная сложность — это критерий, характеризующий временные затраты на реализацию алгоритма. Емкостная сложность—критерий, характеризующий затраты памяти на те же цели. В зависимости от конкретной формы этих критериев сложность алгоритма в свою очередь подрааделяется на *практическую* и *теоретическую.* Практическая временная сложность обычно оценивается во временных единицах (секунды, милисекунды, количество временных тактов процессора, количество выполнения циклов и т.п.). Практическая емкостная сложность выражается, как правило, в битах, байтах, словах и т.п. Способы представления теоретической сложности алгоритма будут рассмотрены далее.

Перечислим основные факторы, от которых может зависеть сложность алгоритма:

• быстродействие компьютера и его емкостные ресурсы (в первую очередь—объем оперативной памяти). В самом деле, чем ниже тактовая частота процессора и меньше объем оперативного запоминающего устройства, тем медленнее выполняются арифметические и логические операции, тем чаще (для больших задач) приходится обращаться к медленно действующей внешней памяти, и, следовательно, больше времени уходит на реализацию алгоритма;

• выбранный язык программирования. Задача, запрограммированная, например, на языке Ассемблера, в общем случае решится быстрее, чем по тому же самому алгоритму, но запрограммированному на языке более высокого уровня, например на Паскале;

• выбранный математический метод формулирования задачи;

• искусство и опыт программиста. В общем случае по одному и тому же алгоритму опытный программист напишет более эффективно работающую программу, чем его начинающий коллега.

Для решения многих проблем существует множество различных алгоритмов. Какой из них выбрать для решения конкретной задачи? Этот вопрос очень тщательно прорабатывается в программировании. Эффективность программы является очень важной ее характеристикой. Эффективность программы имеет две составляющие: память (или пространство) и время. Пространственная эффективность измеряется количеством памяти, требуемой для выполнения программы.

Компьютеры обладают ограниченным объемом памяти. Если две программы реализуют идентичные функции, то та, которая использует меньший объем памяти, характеризуется большей пространственной эффективностью. Иногда память становится доминирующим фактором в оценке эффективности программ. Однако в последние годы в связи с быстрым ее удешевлением эта составляющая эффективности постепенно теряет свое значение. Временная эффективность программы определяется временем, необходимым для ее выполнения.

Лучший способ сравнения эффективностей алгоритмов состоит в сопоставлении их порядков сложности. Этот метод применим как к временной, так и пространственной сложности. Порядок сложности алгоритма выражает его эффективность обычно через количество обрабатываемых данных. Например, некоторый алгоритм может существенно зависеть от размера обрабатываемого массива. Если, например, время обработки удваивается с удвоением размера массива, то порядок временной сложности алгоритма определяется как размер массива. Порядок алгоритма — это функция, доминирующая над точным выражением временной сложности. Функция *f(n)* имеет порядок *0(д(n)),* если имеется константа *к* и счетчик **по,** такие, что **/(п) <** *кд(n)* для **п > по.** Действительное время есть функция, зависящая от длины массива.

Например, известно, что точное время обработки массива определяется из уравнения

$$\text{Действительное время (Длина массива)} = \text{Длина массива}^2 + 5 \cdot \text{Длина массива} + 100.$$

Грубое значение определяется вспомогательной функцией:

$$\text{Оценка времени (Длина массива)} = 1{,}1 \cdot \text{Длина массива}^2.$$

Функция сложности *О* выражает относительную скорость алгоритма в зависимости от некоторой переменной (или переменных). Существуют три важных правила для определения функции сложности:

\.*0(kf) = 0(f).*

2. *O(fg) = 0(f) 0(g)* или *0(f/g) = 0(f)/0(g).*

3. *0(f + g)* равнд доминанте *O(f)* и *O(g).*

Здесь А; обозначает константу, а / и д — функции.

Первое правило декларирует, что постоянные множители не име!эт значения для определения порядка сложности, например:

$$0\{1,5N) = 0\{N).$$

Из второго правила следует, что порядок сложности произведения двух функций равен произведению их сложностей, например::

$$0(17N \cdot N) = 0(17JV)\ 0\{N) = O(N)\ O(N) = 0(N \cdot N) = 0(N^2).$$

Из третьего правила следует, что порядок сложности суммы функций определяется как порядок доминанты первого и второго слагаемых, т.е. выбирается наибольший порядок, например:

$$0\{N^3 + N^2) = 0(N^3).$$

4.1. Виды функции сложности алгоритмов

Функция сложности **0(1).** В алгоритмах константно!i сложности большинство операций в программе выполняется один или несколько раз. Любой алгоритм, всегда требующий независимо от размера данных одного и того же времени, имеет константную сложность.

Функция сложности O(N). Время работы программы линейно, обычно когда каждый элемент входных данных требуется обработал лишь линейное число раз.

Функция сложности $0(N^2)$, $0(N^3)$, $0(N^e)$ — полиномиальная фуНКЦИЯ СЛОЖНОСТИ. ;

Функция сложности $0(\log_2 iV)$, $0(N\log_2 JV)$. Такое врем)| работы имеют те алгоритмы, которые делят большую проблему на множество небольших, а затем, решив их, объединяют решения.

Функция сложности $0(2^N)$. Экспоненциальная сложность. Такие алгоритмы чаще всего возникают в результате подхода, именуемого «метод грубой силы».

4.2. Временная функция сложности

Программист должен уметь проводить анализ алгоритмов и определять их сложность. Временная сложность алгоритма может быть посчитана исходя из анализа его управляющих структур.

Алгоритмы без циклов и рекурсивных вызовов имеют константную сложность. Если нет рекурсии и циклов, то все управляющие структуры могут быть сведены к структурам константной сложности. Следовательно, и весь алгоритм также характеризуется константной сложностью. Определение сложности алгоритма в основном сводится к анализу циклов и рекурсивных вызовов.

Например, рассмотрим алгоритм обработки элементов массива:

```
For i:=l to N do
Begin


End;
```

Сложность этого алгоритма ***O(N),*** так как тело цикла выполняется ***N*** раз, и сложность тела цикла равна 0(1). Если один цикл вложен в другой и оба цикла зависят от размера одной и той же переменной, то вся конструкция характеризуется квадратичной сложностью:

```
For i:=l to N do
Forj:=ltoNdo
Begin "J
   ... > тело цикла
End; J
```

Сложность этой программы $0(N^2)$.

4.3. Анализ функции сложности по программе

Существуют два способа анализа сложности алгоритма: восходящий (от внутренних управляющих структур к внешним) и нисходящий (от внешних структур к внутренним).

Как правило, около 90% времени работы программы требует выполнение повторений и только 10% составляют непосредственно

вычисления. Анализ сложности программ показывает, на какие фра| менты выпадают эти 90% — это циклы наибольшей глубины вл» женности. Повторения могут быть организованы в виде вложенньи циклов или вложенной рекурсии. Эта информация может использ! ваться программистом для построения более эффективной программ следующим образом. Прежде всего можно попытаться сократи1 глубину вложенности повторений. Затем следует рассмотреть во: можность сокращения количества операторов в циклах с наибольшем глубиной вложенности. Если 90% времени выполнения составляв выполнение внутренних циклов, то тридцатипроцентное сокращени этих небольших секций приводит к 27-процентному снижению врем1 ни выполнения всей программы.

Оценим сложность программы «Тройки Пифафра»

```
Г     Writelnf Введите ограничение');
A-    Readln(N);
      small:=1;

   CWhile small < N do
     begin
    B£next:=small;
      ^While next <= N do
        begin
       c£last:=next;
         Г While last <=N do
           begin
            if (last<=small*2) and (next<=small*2) and
              (last*last=small*small+next*napd)then
              begin

^GI \IA      writeln(small);
             writeln(next);
             writeln(last);
           ^   end;

         J £ inc(last);
        V.  end;

       K-{jnc(next);

      ^   end;

   L £inc(small);
   ^   end;
```


O(Я)=O(1)+0(1)+O(1)=O(1);

0(I)=0{N)-{0(F)+0{J))=0{N)-0(aoMHHambiycnoBM) = 0(N);

0(G)=0(NHO(C)+0{I)+0(K))=0(N)-(0(l)+0(N)+0(l))=0(N²)-,

0(E)=0(N)-(0(B)+0(G)+0(L))=0{N)0(N²)=0(N³);

0{D)=0(A)+0(E)=0(l)+0(N³)=0(N³).

Сложность данного алгоритма *0{N³).*

4.4. Оценка алгоритма бинарного поиска

Произведем оценку алгоритма бинарного поиска в масси- ве по приведенной программе:

```
function search(low, high, key: integer): integer;
var
    mid, data: integer;
begin
    while low <= high do
    begin
        mid:=(low+high) div 2;
        data:=a[mid];
        if key=datathen
            search:=mid
        else
            if key < data then
                high := mid-1
            else
                low:=mid+l;
    end;
    search:=-l;
end;
```

Первая итерация цикла имеет дело со всем списком. Каждая по- следующая итерация делит пополам размер массива. Так, размерами ! списка для алгоритма являются

п п п п 71

2'"' 2^' 2^' 2' '"' 2 ш*

В конце концов будет такое целое *m,* что ^ < 2 или *n* < 2^{m+1}. пак как *m* — это первое целое, для которого ^ < 2, то должно быть ве|)йо > 2 или 2^{m} < п. Из этого следует, что 2^{m} < *n* < $2^{т+1}$. Возьмш логарифм каждой части неравенства и получим *m* < $\log_2 n$ < $m_т$ l - Отсюда 0($\log_2$ n).

4.5. Теоретическая и практическая функции сложности

Для оценки эффективности алгоритмов используеп;я функция сложности алгоритма, которая обозначается заглавной буквой «О», в круглых скобках записывается аргумент. Например, фу| ция сложности *0(n²)* читается как функция сложности порядка Функция сложности алгоритма — это функция, которая определяет количество сравнений, перестановок, а также временные и ресурсное затраты на реализацию алгоритма (рис. 4.1).

люд,л *n!*

Рис. 4.1. **Ряд** значений функции сложности

Функция /(п) в ряде случаев может иметь достаточно слож ную аналитическую форму. Поскольку для временной теоретической сложности большее значение имеет не столько вид функции, скол >ко порядок ее роста, то во многих математических дисциплинах, в тем числе и в теории алгоритмов, функцию /(п) определяют как *0(д(г))* и говорят, что она порядка *д(n)* для больших п, если

$$\lim_{П\text{-}УОО} \frac{/(n)}{д(n)} = \text{const} \neq 0,$$

где /(п) и *д(n)* — экспериментальная и теоретическая функции сложности. При этом если /(n) = *0(д(n)),* то предел отношения равен константе и тогда функция /(п) имеет порядок *0(д(м))* («О большое»).

Пример **1.** Полином *f(n)* = $2n^5$ + б$п^4$ + 6$п^2$ +18 имеет *0(n^5),* так как

$$\lim \frac{2n^5 + б п^4 + 6 п^2 + 18}{} = 2.$$

Пример 2. Определить функцию сложности алгоритма по результатам эксперимента:

N	Количество сравнений
54	

Решение. Вначале найдем экспериментальную функцию сложности $O_э$.

Экспериментальная функция сложности алгоритма принимает следующий ряд значений:

$$an,\quad an\log_2 n,\quad an^2,\quad an^3,\quad ae^n,\quad an!$$

Для правильно подобранной экспериментальной функции а = 1, однако на практике допускается 1 < а < 2;

а) допустим, $O_э$ = *an,* тогда an = 54, 6а = 54, а = $\frac{54}{о}$ (не удовлетворяет условию 1 < а < 2).

При *а* = 1 значение экспериментальной функции совпадает со значением теоретической функции сложности;

б) допустим, $O_э$ — $an\log_2 n$, тогда $an\log_2 n$ = 54, $a6\log_2 6$ = 54, а = $\frac{54}{б\log_2 б}$ = $\frac{9}{\log_2 6}$ (а > 2, не удовлетворяет условию);

в) допустим, $O_э$ = *an',* тогда an^2 = 54, а • 36 = 54, а = $\frac{54}{}$ = 1,5 (а < 2 — удовлетворяет условию).

Таким образом, экспериментальная функция сложности имеет вид $O_э(1,5п^2)$.

Найдем теоретическую функцию сложности:

$$\lim_{n\text{-}voo} \frac{/(n)}{д(n)} = \text{const} \neq 0, \qquad \lim_{n\to\infty} \frac{/(n)}{X} = \text{const} \neq 0,$$

$$n\to\infty \quad = \text{const} \neq 0.$$

Отсюда теоретическая функция сложности — 0($п^2$).

Контрольные вопросы

1. Дайте определение функции сложности.
2. Укажите виды функций сложности алгоритмов.
3. Что включает понятие сложности алгоритма?
4. Укажите правила для определения функции сложности.
5. Какие виды функции сложности существуют?
6. Каким образом определяется временная функция сложности?
7. Назовите способы анализа функции сложности по программе.
8. Укажите уравнение, по которому определяется функция сложности.
9. Какие значения принимает экспериментальная функция сложности?
10. Каков смысл коэффициента ***a*** в экспериментальной функции сложности?

ЧАСТЬ 2

АЛГОРИТМЫ ОБРАБОТКИ СТРУКТУР ДАННЫХ

Глава 5
Методы сортировки

Упорядочение элементов множества в возрастающем или убывающем порядке называется *сортировкой.*

С упорядоченными элементами проще работать, чем с произвольно расположенными: легче найти необходимые элементы, исключить, вставить новые. Сортировка применяется при трансляции программ, при организации наборов данных на внешних носителях, при создании библиотек, каталогов, баз данных и т.д.

Алгоритмы сортировки можно разбить на следующие группы (рис. 5.1).

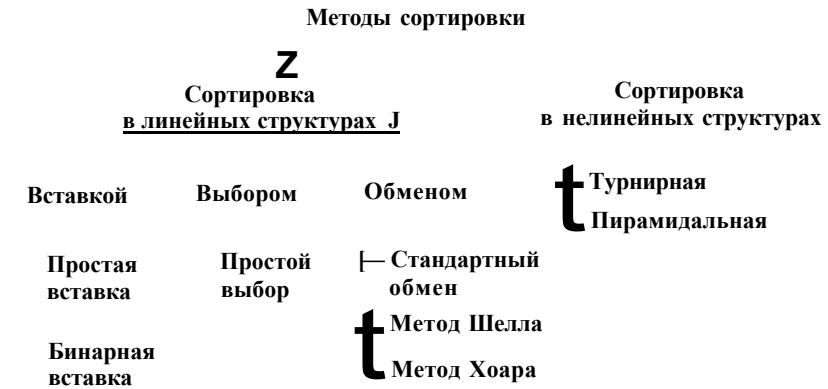


Рис. 5.1. Алгоритмы сортировки

Обычно сортируемые элементы множества называют *записями* и обозначают через $k_1, k_2, \ldots, k_n$.

5.1. Сортировка выбором

Сортировка выбором состоит в том, что сначала в неупорядоченном списке выбирается и отделяется от остальных наименьший элемент. После этого исходный список оказывается измененным. Измененный список принимается за исходный и процесс продолжается до тех пор, пока все элементы не будут выбраны. Очевидно, что выбранные элементы образуют упорядоченный список.

Например, требуется найти минимальный элемент списка

{5, 11, 6, 4, 9, 2, 15, 7}.

Процесс выбора показан на рис. 5.2, где в каждой строчке выписаны сравниваемые пары. Выбираемые элементы с меньшим весом обведены кружком. Нетрудно видеть, что число сравнений соответствует на рисунке числу строк, а число перемещений — количеству изменений выбранного элемента.

Рис. 5.2. Сортировка выбором

Выбранный в исходном списке минимальный элемент размещается на предназначенном ему месте несколькими способами.

• Минимальный элемент после г'-го просмотра перемещается на г-е место нового списка (г = 1,2,..., п), а в исходном списке на место выбранного элемента записывается какое-то очень большое число, превосходящее по величине любой элемент списка. При этом длина заданного списка остается постоянной. Измененный таким образом список можно принимать за исходный.

• Минимальный элемент записывается на г-е место исходного списка (г = 1, 2,..., *n),* а элемент с г-го места—на место выбранного. При этом очевидно, что уже упорядоченные элементы (а они будут расположены, начиная с первого места) исключаются из дальнейшей сортировки, поэтому длина каждого последующего списка (списка, участвующего в каждом последующем просмотре) должна быть на один элемент меньше предыдущего.

• Выбранный минимальный элемент, как и в предыдущем случае, перемещается на г-е место заданного списка, а чтобы это г-е место освободилось для записи очередного минимального элемента, левая от выбранного элемента часть списка перемещается вправо на одну позицию так, чтобы заполнилось место, занимаемое до этого выбранным элементом.

Сложность метода сортировки выбором порядка $0(n^2)$.

5.2. Сортировка вставкой и сортировка слиянием

В этом методе из неупорядоченной последовательности элементов выбирается поочередно каждый элемент, сравнивается с предыдущим, уже упорядоченным, и помещается на соответствующее место.

Сортировка вставкой. Такую сортировку рассмотрим на примере заданной неупорядоченной последовательности элементов:

{40, 11, 83, 57, 32, 21, 75, 64}.

Процедура сортировки отражена на рис. 5.3, где кружком на каждом этапе обведен анализируемый элемент, стрелкой сверху отмечено место перемещения анализируемого элемента, в рамку заключены упорядоченные части последовательности.

На 1-м этапе сравниваются два начальных элемента. Поскольку второй элемент меньше первого, он перемещается на место первого элемента, который сдвигается вправо на одну позицию. Остальная часть последовательности остается без изменения.

На 2-м этапе из неупорядоченной последовательности выбирается элемент и сравнивается с двумя упорядоченными ранее элементами. Так как он больше предыдущих, то остается на месте. Затем

анализируются четвертый, пятый и последующие элементы до тех пор, пока весь список не будет упорядоченным, что имеет место на последнем 7-м этапе.

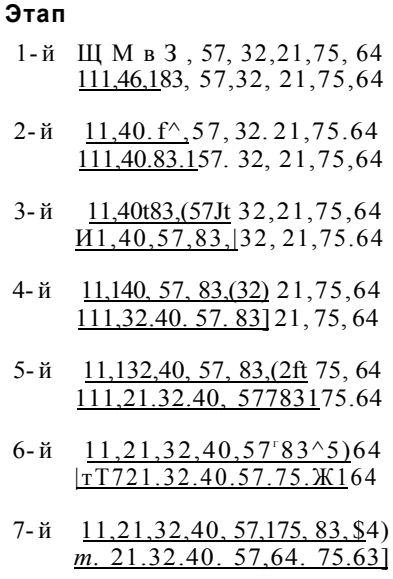


Рис. 5.3. Сортировка вставкой

Сортировка слиянием. Разновидностью сортировки вставкой является метод фон Неймана, или сортировка слиянием. Идея метода состоит в следующем: сначала анализируются первые элементы обоих массивов. Меньший элемент переписывается в новый массив. Оставшийся элемент последовательно сравнивается с элементами из другого массива. В новый массив после каждого сравнения попадает меньший элемент. Процесс продолжается до исчерпания элементов одного из массивов. Затем остаток другого массива дописывается в новый массив. Полученный новый массив упорядочен таким же образом, как исходные.

Пусть имеются два отсортированных в порядке возрастания массива р[1], *p[2],p[n]* и *q[l], q[2], q[n]* и имеется пустой массив *г[1], г[2],..., г[2n],* который мы хотим заполнить значениями массивов *ри q в* порядке возрастания. Для слияния выполняются следующие действия: сравниваются *p[1]* и *q[l]* и меньшее из значений записывается в г[1]. Предположим, что это значение*р[1].* Тогда*р[2]* сравнивается

с д[1], и меньшее из значений заносится в *г[2].* Предположим, что это значение *q[l].* Тогда на следующем шаге сравниваются значения *p[2)* и *q[2]* и т.д., пока мы не достигнем границ одного из массивов. Тогда остаток другого массива просто дописывается в «хвост» массива г. Пример слияния двух массивов показан на рис. 5.4.

Сложность метода сортировки вставкой порядка $0(n^2)$.

5.3. Сортировка обменом и шейкерная сортировка

Сортировка обменом. Это метод, в котором элементы списка последовательно сравниваются между собой и меняются местами в том случае, если предшествующий элемент больше последующего. Требуется, например, провести сортировку списка методом стандартного обмена, или методом «пузырька»:

{40, 11, 83, 57, 32, 21, 75, 64}.

Обозначим квадратными скобками со стрелками *i j* обмениваемые элементы, а LJ — сравниваемые элементы. Первый этап сортировки показан на рис. 5.5, а второй этап — на рис. 5.6.

Нетрудно видеть, что после каждого просмотра списка все элементы, начиная с последнего, занимают свои окончательные позиции, поэтому их не следует проверять при следующих просмотрах. Каждый последующий просмотр исключает очередную позицию с найденным максимальным элементом, тем самым укорачивая список.

После первого просмотра в последней позиции оказался больший элемент, равный 83 (исключаем его из дальнейшего рассмотрения). Второй просмотр выявляет максимальный элемент, равный 75 (см. рис. 5.6).

Процесс сортировки продолжается до тех пор, пока не будут формированы все элементы конечного списка либо не выполнится словие Айверсона.

Условие Айверсона: если в ходе сортировки при сравнении ^тементов не было сделано ни одной перестановки, то множество читается упорядоченным (условие Айверсона выполняется только и шаге *d* = 1).

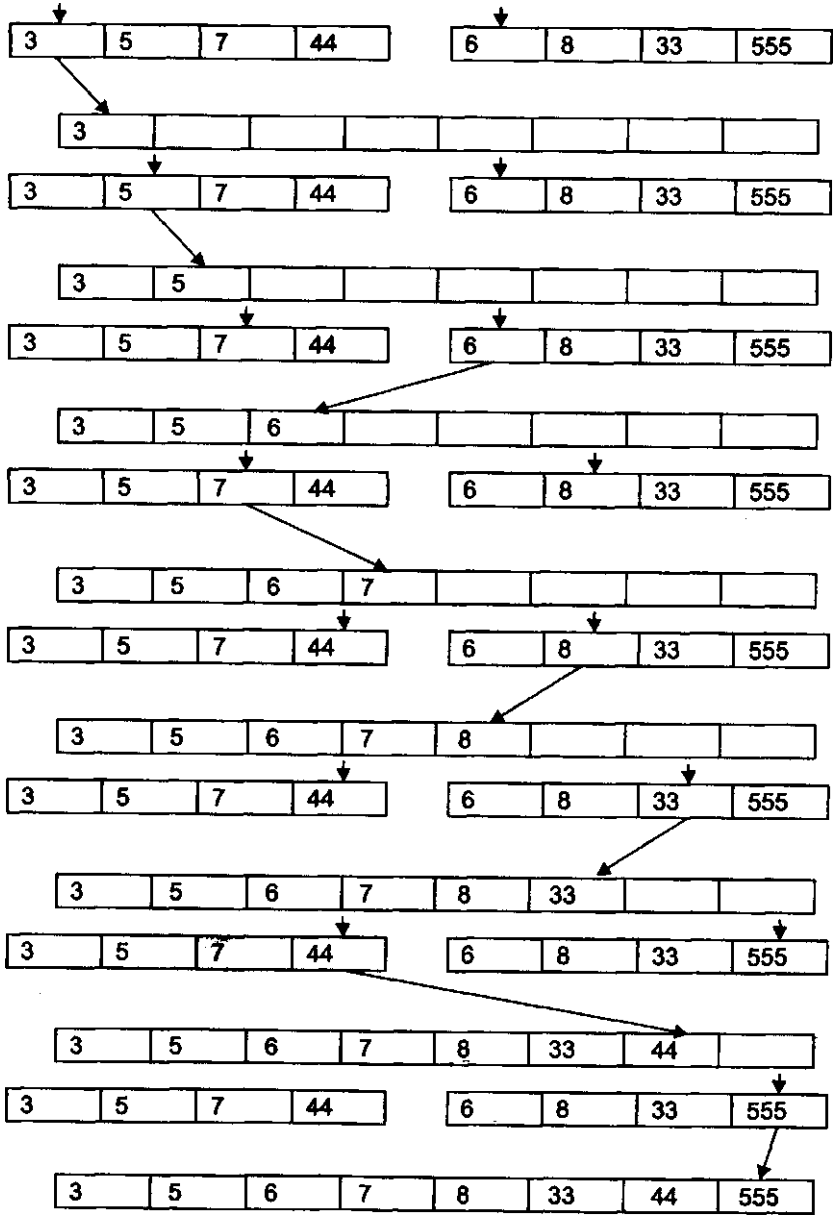


Рис. 5.4. Сортировка слиянием

Исходный список	40	11	63	57	32	21	75	64
Первый просмотр	t 11	t 40 40	83 57	t 83 32	t 83 t 21	t 83 t 75	t 83 64	t 83
Полученный список	11	40	57	32	21	75	64	

Рис. 5.5. Сортировка обменом (первый просмотр)

Исходный список	11	40	57	32	21	75	64
Второй просмотр	I 11	I 40 40"	~57 t 32	t 57 t 21	t 57	I 75 t 64	t 75
Полученный список	11	40	32	21	57	64	

Рис. 5.6. Сортировка обменом (второй просмотр)

Сложность метода стандартного обмена $0(n^2)$.

Шейкерная сортировка. Очевидный прием улучшения алгорит-і стандартного обмена — запоминать, были или не были переста-овки в процессе некоторого прохода. Если в последнем проходе грестановок не было, то алгоритм можно заканчивать. Это улуч-;ние, однако, можно опять же улучшить, если запоминать не только ам факт, что обмен имел место, но и положение (индекс) последнего 5мена. Ясно, что все пары соседних элементов выше этого индекса *к*

L-2	3	3	4	4
R=8	8	7	7	4
dir	t	i	T	***I***	T
	44	06	06	06	06
	55	44	44	12	12
	12	55	12	44	18
	42	12	42	18	42
	94	42	55	42	44
	18	94	18	55	55
	06	18	67	67	67
	67	67	94	94	94

Рис. 5.7. Схема шейкерной сортировки

уже упорядочены. Поэтому просмотры можно заканчивать на этом индексе, а не идти до заранее определенного нижнего предела для *г*. Например, массив {12, 18, 42, 44, 55, 67, 94, 06} с помощью усовершенствованной «пузырьковой» сортировки можно упорядочить за один просмотр, а для сортировки массива {94, 06, 12, 18, 42, 44, 55, 67} требуется семь просмотров. Это приводит к мысли: чередовать направление последовательных просмотров. Модификацией сортировки стандартным обменом является *шейкерная,* или *челночная, сортировка.*

На рис. 5.7 приведена схема шейкерной сортировки восьми ключей.

5.4. Сортировка Шелла

В методе Шелла сравниваются не соседние элементы, а элементы, расположенные на расстоянии *d* (где *d* — шаг между сравниваемыми элементами).

Если *d* = [п/2], то после каждого просмотра шаг *d* уменьшается вдвое. На последнем просмотре он сокращается до *d* = 1.

Например, пусть дан список, в котором число элементов четно:

{40, 11, 83, 57, 32, 21, 75, 64}.

Список длины *n* разбивается на п/2 частей, т.е. *d = [n/2]* = 4, где [ ] — целая часть числа.

При первом просмотре сравниваются элементы, отстоящие друг от друга на *d* = 4 (рис. 5.8), т.е. *ki* и *кь, къ* и *кв* и т.д. Если *ki* > *ki+i,* то происходит обмен между позициями г и (г + *d).* Перед вторым просмотром выбирается шаг *d = [d/2] = 2* (рис. 5.9). Затем выбираем шаг *d* = [d/2] = 1 (рис. 5.10), т.е. имеем аналогию с методом стандартного обмена.

Сложность метода Шелла $O(0{,}3n(\log_2 n)^2)$.

Исходный массив	40	11	83	57	32	21	75	64
UJard = 4	32	11	75	57	40	21	83	64
Полученный массив	32	11	75	57	40	21	83	64

Рис. 5.8. Метод Шелла (шаг *d* = 4)

Исходный массив	32	11	75	57	40	21	83	64
UJard = 2								
Полученный массив	**32**	**11**	**40**	**21**	75	57	83	64

Рис. 5.9. Метод Шелла (шаг *d* = 2)

Исходный список	32	11	40	21	75	57	83	64
mard = 1								
Полученный список	11	32	21	40	57	75	64	(ёз)

Рис. 5.10. Метод Шелла (шаг *d* = 1)

5.5. Быстрая сортировка (сортировка Хоара)

В методе быстрой сортировки фиксируется какой-либо ключ (базовый), относительно которого все элементы с большим весом перемещаются вправо, а с меньшим — влево. При этом весь список элементов делится относительно базового ключа на две части. Для каждой части процесс повторяется. Поясним метод на примере.

На рис. 5.11 представлены этапы быстрой сортировки Хоара. В первой строке указана исходная последовательность.

Номер шага	/——>						<——j		Примечание
	40	11	83	57	32	21	75	64	Исходный список
1	t40j-							<§4)	‘o*j
2	|40г-						•<75)		‘ o ‘ ‘ ;
3	[4Qh @*					*140(			Обмен; ‘o*j
4		(TiV				-[40j			‘ , ‘ ‘ o
5			# - |40К		<	щ			Обмен; ‘ , ’ ‘ c
6			ЦоК.						Обмен; ‘ o ’ ‘ ,
7				m	/Ш				Обмен; ‘ , ’ ‘ o
	21	11	32	N	57	83	75	64	Полученный список

Рис. 5.11. Метод Хоара

Примем первый элемент последовательности за базовый ключ, выделим его квадратом и обозначим fco = 40. Установим два указателя г и *j,* из которых г начинает отсчет слева (г = 1), *aj* — справа *(J — n).*

Сравниваем базовый ключ fco и текущий ключ *kj.* Если *ко* < *kj,* то устанавливаем *j* = *j* - 1 и проводим следующее сравнение fco и *kj.* Продолжаем уменьшать *j* до тех пор, пока не достигнем условия fco > *kj.* После этого меняем местами ключи fc) и *kj* (см. шаг 3 на рис. 5.11).

Теперь начинаем изменять индекс г = г + 1 и сравнивать элементы *ki* и fco. Продолжаем увеличение г до тех пор, пока не получим условие fcj > fco, после чего следует обмен fc, и *fo* (см. шаг 5). Снова возвращаемся к индексу j, уменьшаем его. Чергдуя уменьшение *j* и увеличение г, продолжаем этот процесс с обоих концов к середине до тех пор, пока не получим г = *j* (см. шаг 7).

В отличие от предыдущих рассмотренных сортировок уже на первом этапе имеют место два факта: во-первых, базовый ключ fco=40 занял свое постоянное место в сортируемой последовательности; во-вторых, все элементы слева от fco будут меньше него, а справа — больше него. Таким образом, по окончании перзого этапа имеем:

21,11,32 U p l 57, 83,75,64
Левая часть Правая часть

Указанная процедура сортировки применяемся независимо к левой и правой частям.

Сложность метода Хоара: *0(n* log₂ *n).*

5.6. Турнирная сортировка

Элементы исходного множества представляются листьями дерева. Их попарное сравнение позволяет определить минимальный (максимальный) элемент. Метод турниркой сортировки основан на повторяющихс» поисках наименьшего ключа среди п элементов, среди оставшихся га-1 элементов и т.д. Например, сделав п/2 сравнений, можно определить в каждой паре ключей меньший. С помощью п/4 сравнений — меньший из пары уже выбранных меньших и т.д.

Проделав n — 1 сравнений, можно построить дерево выбора (рис. 5.12) и идентифицировать его корень как наименьший ключ.

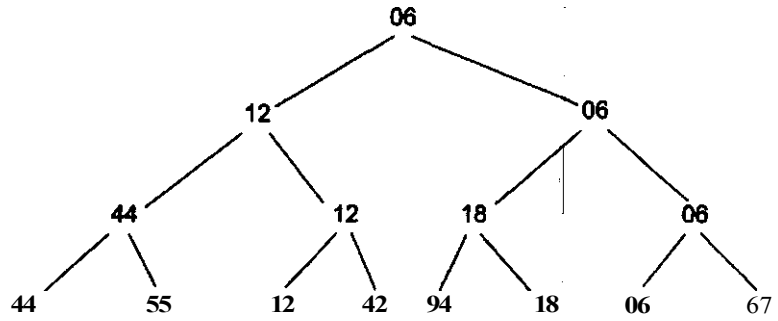


Рис. **5.12.** Повторяющиеся выборы среди ключей

Следующий этап сортировки — спуск вдоль пути, отмеченного наименьшим элементом, и исключение его из дерева путем замены либо на пустой элемент (дырку) в самом **нн3у,j** либо на элемент из соседней ветви в промежуточных вершинах (рис^ .5.13).

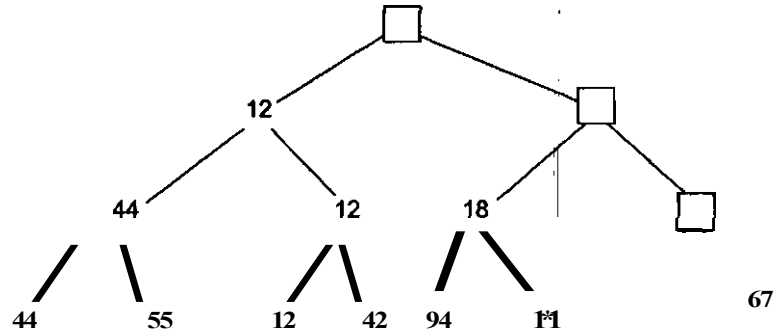


Рис. **5.13.** Исключение наименьшего ключа

Элемент, передвинувшийся в корень дерева, рновь будет наименьшим (теперь уже вторым) ключом (рис. 5.14), и можно исключить. После n таких шагов дерево станет пустым (т.е |в нем останутся одни дырки), и процесс сортировки заканчивается.

Свое название турнирная сортировка получила, потому что она используется при проведении соревнований, турниров и олимпиад.

Пример 1. Дано исходное множество {а!, *а2,* оЗ, а4, а5, об, а7, а8}. Осуществить турнирную сортировку.

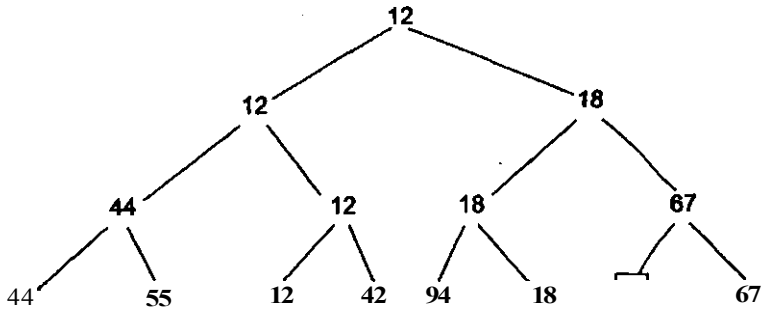


Рис. **5.14.** Заполнение дырки

Производится попарное сравнение вершин дерева (рис. 5.15). Найденный минимальный элемент заменяется специальным символом M и помещается в результирующее множество (рис. 5.16).

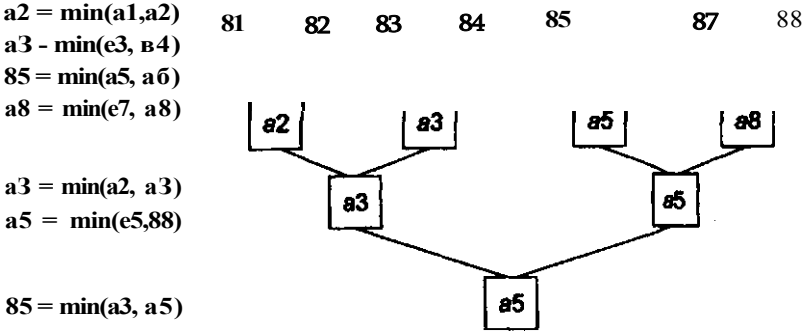


Рис. **5.15.** Первый этап турнирной сортировки

На последующих этапах найденные минимальные элементы помещаются в результирующее множество (рис. 5.17).

5.7. Пирамидальная сортировка

Данный тип сортировки заключается в построение пирамидального дерева. Пирамидальное дерево — это бинарное дерево, обладающее тремя свойствами:

a1 a2 I I a3 **s4** **a6** **a7** **86**

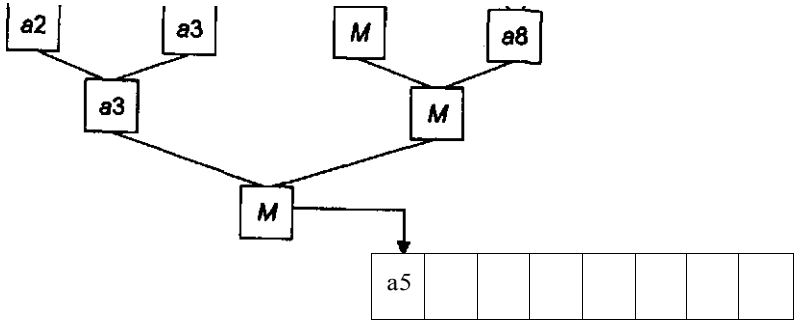


Рис. 5.16. Удаление минимального элемента

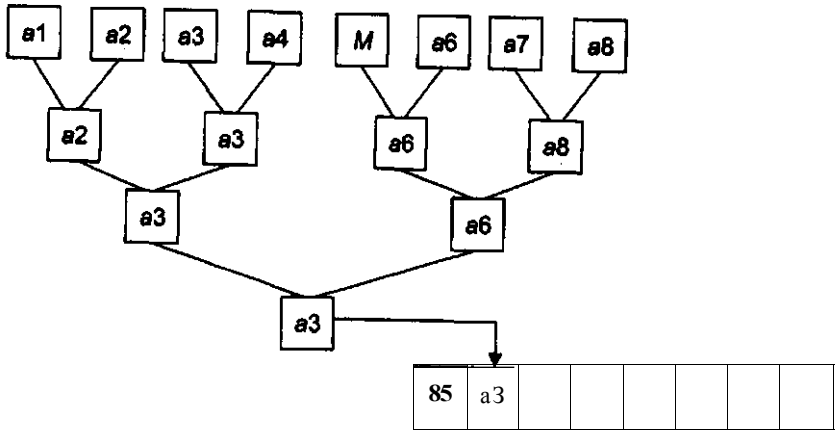


Рис. 5.17. Этапы турнирной сортировки

• в вершине каждой триады располагается элемент с большим весом;

• листья бинарного дерева располагаются либо в одном уровне, либо в двух соседних (рис. 5.18);

• листья нижнего уровня располагаются левее листьев более высокого уровня.

В ходе преобразования элементы триад сравниваются дважды (рис. 5.19), при этом элемент с большим весом перейдет вверх, а с меньшим — вниз.

а ***б***

Рис. 5.18. Бинарное дерево:
а — листья на одном уровне;
б — листья на соседних уровнях

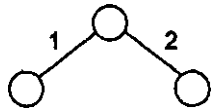


Рис. 5.19. Сравнение элементов триад:
/ — первое сравнение; *2* — второе сравнение

П р и м е р . Дано исходное множество {2, 4, б, 3, 5, 7}. В ходе сортировки последовательно сравниваются элементы триад. При этом элемент с большим весом перемещается в корень дерева.

После завершения этапа сортировки элемент, находящийся в корне дерева, помещается на место последнего элемента и в дальнейшем не рассматривается. Метод пирамидальной сортировки показан на рис. 5.20.

В результате будет получено упорядоченное множество {2, 3, 4, 5, 6, 7}.

Контрольные вопросы

1. Что понимается под сортировкой?
2. Каковы особенности сортировки: вставкой, выбором, обменом?
3. Каковы особенности сортировки Шелла и Хоара?
4. Каковы особенности сортировки турнирной и пирамидальной?
5. Какова основная идея шейкерной сортировки?
6. К какой группе методов относится сортировка фон Неймана?
7. В чем состоит методика анализа сложности алгоритмов сортировки?

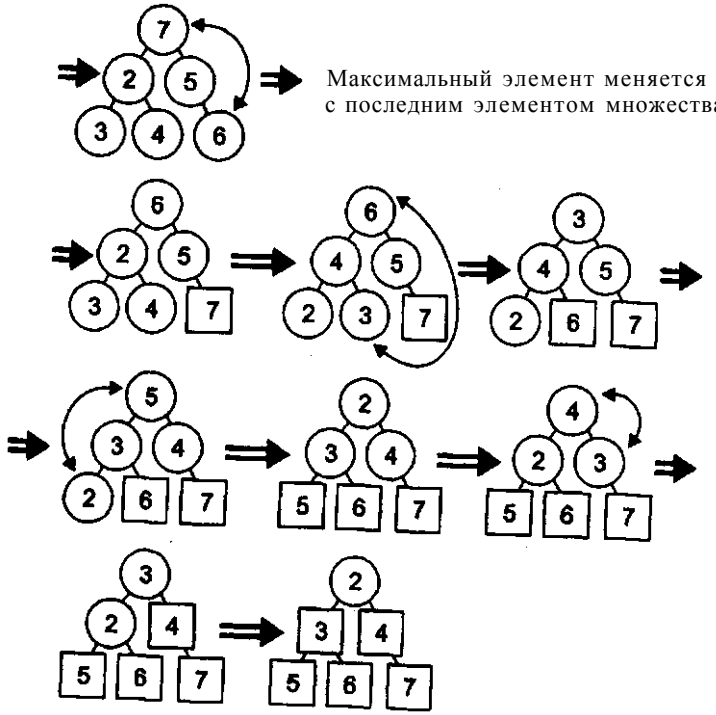


Рис. 5.20. Пирамидальная сортировка

Глава 6
Методы поиска

Предметы (объекты), составляющие множество, называются его *элементами.* Элемент множества будет называться *ключом* и обозначаться латинской буквой *ki* с индексом, указывающим номер элемента.

Алгоритмы поиска можно разбить на следующие группы (рис. 6.1).

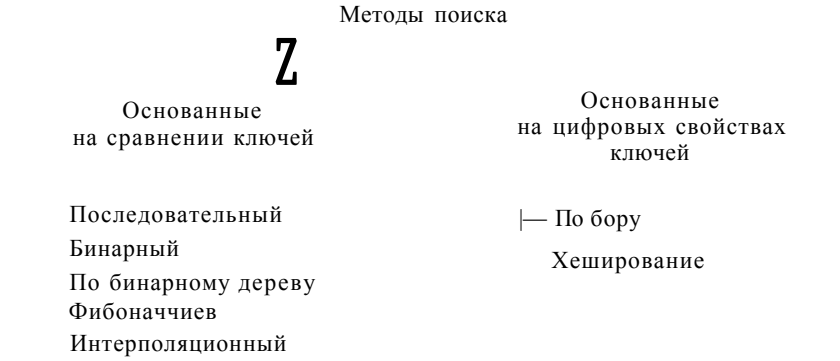


Рис. 6.1. Методы поиска

Задача поиска. Пусть задано множество ключей {fci, fo, • •• > k_n}. Необходимо отыскать во множестве ключ ***ki.*** Поиск может быть завершен в двух случаях:

- ключ во множестве отсутствует;
- ключ найден во множестве.

6.1. Последовательный поиск

В последовательном поиске исходное множество не упорядочено, т.е. имеется произвольный набор ключей *{к\, кг, кз,* fc_n}. Метод заключается в том, что отыскиваемый ключ *ki* последовательно сравнивается со всеми элементами множества. При этом поиск заканчивается досрочно, если ключ найден.

Алгоритм поиска сводится к последовательности шагов:

Шаг Si. [Начальная установка.] Установить г := 1.

Шаг 5г. [Сравнение.] Если *к — ки* алгоритм заканчивается удачно.

Шаг *S3.* [Продвижение.] Увеличить г на 1.

Шаг 5**4.** [Конец файла?] Если *i < N,* то вернуться к шагу S2. В противном случае алгоритм заканчивается неудачно.

6.2. Бинарный поиск

В бинарном поиске исходящее множество должно быть упорядочено по возрастанию, иными словами, каждый последующий ключ больше предыдущего, т.е. *к\ <к2 <кз < ЫІ < ... < к„-\ < к„.*

Отыскиваемый ключ сравнивается с центральным элементом множества, если он меньше центрального, то поиск продолжается в левом подмножестве, в противном случае — в правом.

Медианой последовательности из п элементов считается элемент, значение которого меньше (или равно) половины *n* элементов и больше (или равно) другой половины. Задачу поиска медианы принято связывать с сортировкой, так как медиану всегда можно найти следующим способом: отсортировать *n* элементов и затем выбрать средний элемент.

В алгоритме К. Хоара для нахождения медианы используется операция разделения, применяемая при быстрой сортировке, с *L* = 1, й = п и с *a[к],* выбранными в качестве разделяющего значения *x*. Получаются значения индексов г и *j:*

1) разделяющее значение *x* было слишком мало; в результате граница между двумя частями ниже искомого значения *к*. Процесс разделения следует повторить для элементов *a[i],..., a[R]* (рис. 6.2);

Рис. 6.2. Смещение границы влево

2) выбранная граница *x* была слишком велика. Операцию разбиения следует повторить на подмассиве *a[L],..., a[j]* (рис. 6.3);

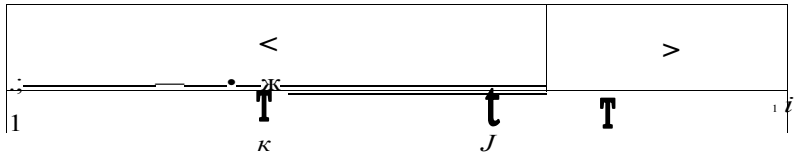


Рис. 6.3. Смещение границы вправо

3) значение *к* лежит в интервале *j* < *к* < г: элемент *a[к]* разделяет массив в заданной пропорции и, следовательно, является искомым (рис. 6.4).

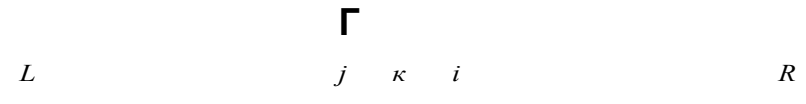


Рис. 6.4. Граница определена точно

Процесс разбиения повторяется до появления последнего случая.

Центральный элемент находится по формуле *N^.-n — [n/2]* + 1, где квадратные скобки обозначают, что от деления берется только целая часть, дробная часть отбрасывается. В методе бинарного поиска анализируются только центральные элементы подмножеств.

Пример 1. Во множестве элементов отыскать ключ, равный 653. В квадратных скобках выделены множества анализируемых элементов. Центральные элементы подмножеств подчеркнуты.

Поиск ключа *К* = 653 осуществляется за четыре шага:

[061 087 154 170 275 426 503 §09 512 612 653 677 703 765 897 908]
061 087 154 170 275 426 503 509 **[512 612 653 677 703 765 897 908]**
061 087 154 170 275 426 503 509**[512 612 653]**677 703 765 897 908
061 087 154 170 275 426 503 509 512 612 **[653]** 677 703 765 897 908

Пример 2. Дано упорядоченное множество элементов

{7, 8, 12, 16, 18, 20, 30, 38, 49, 50, 54, 60, 61, 69,
75, 79, 80, 81, 95, 101, 123, 198).

Найти во множестве ключ *К* = 61.

Шаг 1. Л^эл-та = [п/2] + 1 = [22/2] + 1 = 12;

{7, 8, 12, 16, 18, 20, 30, 38, 49, 50, 54, 60, 61, 69,
75, 79, 80, 81, 95, 101, 123, 198};

К V & 12 (значок «V» обозначает сравнение элементов: чисел, значений). Поскольку 61 > 60, дальнейший поиск выполняем в правом подмножестве.

Шаг 2. $N_{м.п}$ = *[n/2]* +1 = [10/2] + 1 = 6;

{61, 69, 75, 79, 80, 81, 95, 101, 123, 198};

K V *k\s.* Поскольку 61 < 81, дальнейший поиск выполняем в левом подмножестве.

Ш а г 3. iV$_{з,,,\cdot т_3}$ = [п/2] + 1 = [5/2] + 1 = 3;

{61,69,75,79,80};

KVki5. Поскольку 61 < 75, дальнейший поиск выполняем в левом подмножестве.

Ш а г 4. TV*$_{,}$™ = [п/2] + 1 = [2/2] + 1 = 2;

{61, 69};

K V **&14.** Поскольку 61 < 69, дальнейший поиск выполняем в левом подмножестве.

Ш а г 5. {61}. *K* V *kiz.* Так как 61 = 61, делаем вывод: искомый ключ найден под номером 13.

6.3. Фибоначчиев поиск

6 этом поиске анализируются элементы, находящиеся в позициях, равных числам Фибоначчи. Числа Фибоначчи получаются по следующему правилу: каждое последующее число равно сумме двух предыдущих чисел, например:

{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...}.

Поиск продолжается до тех пор, пока не будет найден интервал между двумя ключами, где может располагаться отыскиваемый ключ.

Фибоначчиев поиск предназначается для поиска аргумента *K* среди расположенных в порядке возрастания ключей *K\ < Ki* < ... ...<K_n.

Для удобства описания предполагается, что *n* + 1 есть число Фибоначчи *Fk.i.* Подходящей начальной установкой данный метод можно сделать пригодным для любого п.

F\. [Начальная установка.] Установить *г := F^, p* := -Ffc-i, *q* := := -£fc$_{-2}$. (В алгоритме *p* и *q* обозначают последовательные числа Фибоначчи.)

F2. [Сравнение.] Если *K < Ki,* то перейти на F$_3$; если *K > Ki, mo* перейти на **F 4;** если *K = Ki,* алгоритм заканчивается удачно.

F 3. [Уменьшение!.] Если *q* = 0, алгоритм заканчивается неудачно. Если *q ф* 0, то установить *г := i — q,* заменить *(p,q)* на *(q,p — q)* и вернуться на ***F2.***

F4. [Увеличение г.] Если *p* = 1, алгоритм заканчивается неудачно. Если *рф1,* установить *г := г + q, p := p — q, q:= q — p* и вернуться на^г.

Пример. Дано исходное множество ключей

{3, 5, 8, 9, 11, 14, 15, 19, 21, 22, 28, 33, 35, 37, 42, 45, 48, 52}.
Пусть отыскиваемый ключ равен 42 *(K* = 42).

Последовательное сравнение отыскиваемого ключа будет проводиться с элементами исходного множества, расположенными в позициях, равных числам Фибоначчи: 1,2,3, 5,8,13,21,...

Ш а г 1. *K ~ ki,* 42 > 3, отыскиваемый ключ сравнивается с ключом, стоящим в позиции, равной числу Фибоначчи.

Ш а г 2. *K ~* fo, 42 > 5, сравнение продолжается с ключом, стоящим в позиции, равной следующему числу Фибоначчи.

Ш а г 3. *K ~ кз,* 42 > 8, сравнение продолжается.

Ш а г 4. *K ~ к$,* 42 > 11, сравнение продолжается.

Ш а г 5. *K ~* Are, 42 > 19, сравнение продолжается.

Ш а г 6. *K ~ к\з,* 42 > 35, сравнение продолжается.

Ш а г 7. *K ~ kis,* 42 < 52, найден интервал, в котором находится отыскиваемый ключ, т.е. отыскиваемый ключ может находиться в исходном множестве между позициями 13 и 18, т.е. {35, 37, 42, 45, 48, 52}.

В найденном интервале поиск вновь ведется в позициях, равных числам Фибоначчи.

6.4. Интерполяционный поиск

Исходное множество должно быть упорядочено по возрастанию весов. Первоначальное сравнение осуществляется на расстоянии шага d, который определяется по формуле:

Kj-Ki

где t — номер первого рассматриваемого элемента;
3 — номер последнего рассматриваемого элемента;
К — отыскиваемый ключ;
Ki, Kj — значения ключей в позициях г и j;
[ ] — целая часть числа.

Идея метода заключается в следующем: шаг *d* меняется nocje каждого этапа по формуле, приведенной выше (рис. 6.5).

Рис. 6.5. Схема интерполяционного поиска

Алгоритм заканчивает работу при *d* = 0, при этом анализируютс' соседние элементы, после чего принимается окончательно решение о результатах поиска.

Этот метод прекрасно работает, если исходное множество пред ставляет собой арифметическую прогрессию или множество, прибли женное к ней.

Пример 1. Дано множество ключей:

{2, 9, 10, 12, 20, 24, 28, 30, 37, 40, 45, 50, 51, 60, 65, 70, 74, 76}

Пусть искомый ключ ***К*** = 70.

Определим шаг *d* для исходного множества ключей (г = 1; *j* = 18)

$$d = \frac{(18-1)(70-2)}{76-2} = 15.$$

Сравниваем ключ, стоящий под шестнадцатым порядковым номером. данном множестве с искомым ключом: A;i₀ V ***К;*** 70 = 70; ключ найден

Пример 2. Дано множество ключей:

{4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95, 98, 102}.

Требуется отыскать ключ ***К*** = 90.

Шаг 1. Определим шаг *d* для исходного множества ключей (г = 1; *j* = 17):

$$d = \frac{'(17-1)(90-4)"}{102-4} \quad 14.$$

Сравниваем ключ, стоящий под пятнадцатым порядковым номером, с отыскиваемым ключом: fcis V ***К,*** 95 > 90, следовательно, сужается область поиска:

{4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95}.

Шаг 2. Определим шаг d для множества ключей (г = 1; *j* = 15):

$$\frac{'(15-1)(90-4)"}{95-4} = 13.$$

Сравниваем ключ, стоящий под четырнадцатым порядковым номером, с отыскиваемым ключом: ***кц*** V ***К; ки = К;90*** = 90; ключ найден.

6.5. Поиск по бинарному дереву

Дерево двоичного поиска для множества чисел *S* — это размеченное *бинарное дерево,* каждой *вершине* которого сопоставлено число из множества *S,* причем все пометки удовлетворяют следующему простому правилу: «если больше — направо, если меньше — налево».

Например, для набора чисел {7, 3, 5, 2, 8, 1, 6, 10, 9, 4, 11} получится бинарное дерево (рис. 6.6). Для того чтобы правильно учесть повторения чисел, можно ввести дополнительное поле, которое будет хранить количество вхождений для каждого числа.

Бинарное дерево, соответствующее бинарному поиску среди *п* записей, можно построить следующим образом: при п = 0 дерево сводится к узлу 0. В противном случае корневым узлом является [п/2], левое поддерево соответствует бинарному дереву с [п/2] — 1 узлами, а правое — дереву с [п/2] узлами и числами в узлах, увеличенными на [п/2] (рис. 6.7).

Число узлов, порожденных отдельным узлом (число поддеревьев данного корня), называется его *степенью.* Узел с нулевой степенью

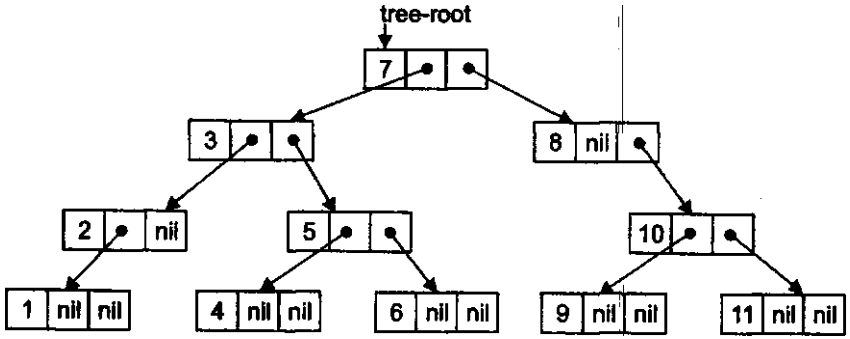


Рис. 6.6. Дерево двоичного поиска

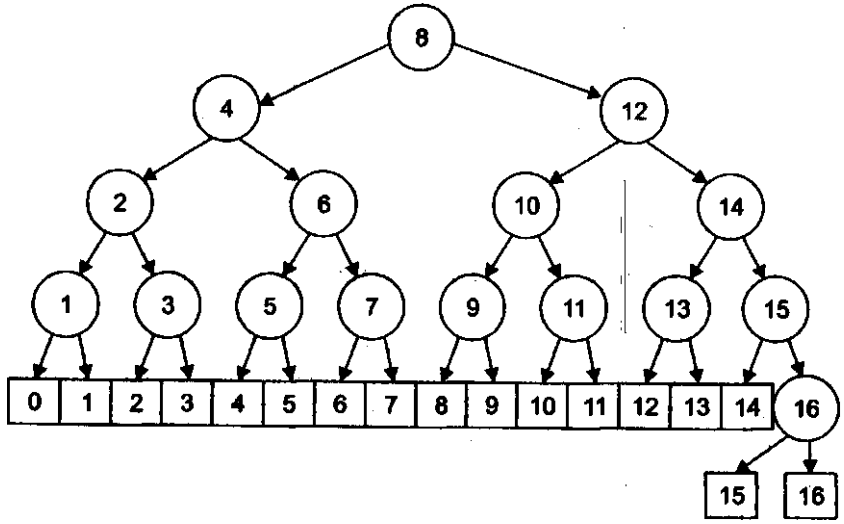


Рис. 6.7. Бинарное дерево, соответствующее бинарному поиску (п = 16)

называют ***листом,*** или ***концевым узлом*** Максимальное значение степени всех узлов данного дерева называется ***степенью дерева.***

Алгоритм поиска по бинарному дереву: вначале аргумент поиска сравнивается с ключом, находящимся в кор»ф Если аргумент совпадает с ключом, поиск закончен, если не совпадает, то в случае, когда аргумент оказывается меньше ключа, поиск п родолжается в левом поддереве, а в случае, когда больше ключа, — в правом поддереве. Увеличив уровень на 1, повторяют сравнение, счи)гая текущий узел корнем.

Использование структуры бинарного дерева позволяет быстро вставлять и удалять записи и проводить эффективный поиск по таблице. Такая гибкость достигается добавлением в каждую запись двух полей для хранения ссылок.

Пусть дано бинарное дерево поиска (рис. 6.8). Требуется по бинарному дереву отыскать ключ SAG.

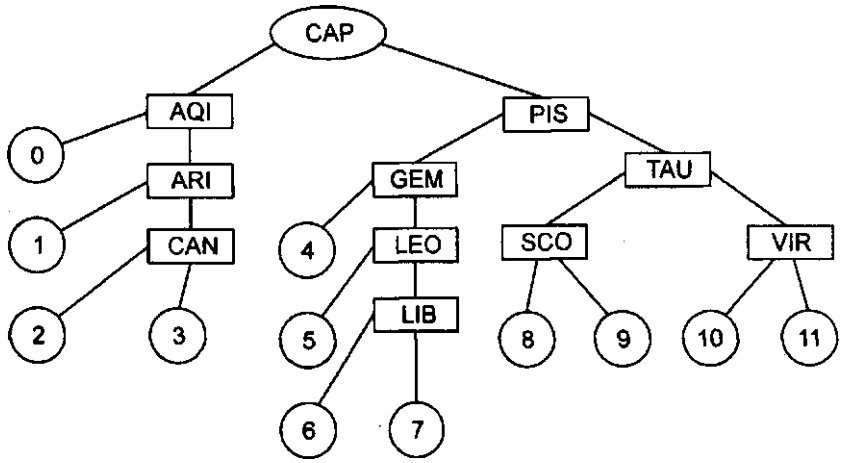


Рис. 6.8. Бинарное дерево

При просмотре от корня дерева видно, что по первой букве латинского алфавита название SAG больше чем CAP. Следовательно, дальнейший поиск будем осуществлять в правой ветви. Это слово больше, чем PIS, значит, снова идем вправо; оно меньше, чем TAU, — идем влево; оно меньше, чем SCO, попадаем в узел 8. Таким образом, название SAG должно находиться в узле 8. При этом узлы дерева имеют структуру, представленную в табл. 6.1.

Таблица 6.1

Структура узлов дерева

Ключ	Информационная часть	Указатель на левое поддерево	Указатель на правое поддерево
	(может отсутствовать) KEY	LLINK	RLINK

Пример. Пусть дано исходное множество ключей

{2, 4, 5, 6, 7, 9, 12, 14, 18, 21, 24, 25, 27, 30, 32, 33, 34, 37, 39}.

Исходное множество ключей должно быть упорядочено по возрастанию.

От линейного списка переходим к построению бинарного дерева поиска (рис. 6.9). В данном случае корнем дерева является центральный элемент множества $N^{\wedge}.^{\wedge} = [n/2] + 1$, где n — количество элементов множества. Вершиной по левой ветке является центральный элемент левого подмножества, а правой — правого подмножества, и т.д.

Отыскиваемый ключ $K = 24$.

Исходное множество имеет 19 элементов. $Л^{\wedge}л._{,,} = [19/2] + 1 = 10$.

Поиск ключа $K = 24$ по бинарному дереву от корня до листьев показан на рис. 6.9.

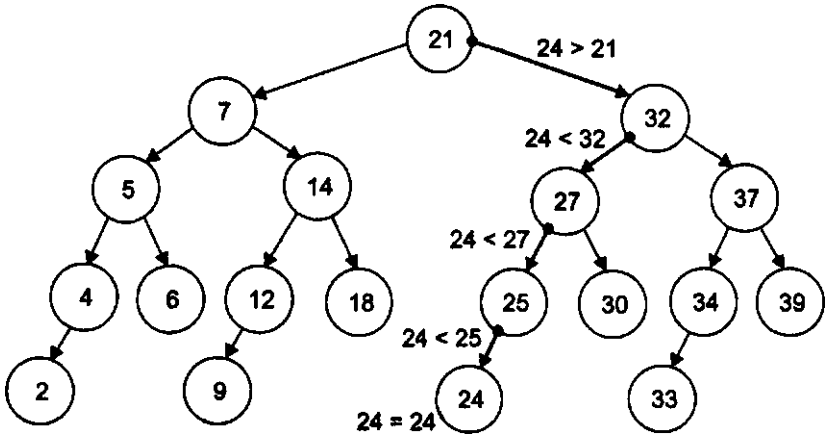


Рис. 6.9. Бинарное дерево поиска

Преобразование произвольного дерева в бинарное. Упорядоченные деревья степени 2 называются *бинарными деревьями.* Бинарное дерево состоит из конечного множества элементов (узлов), каждый из которых либо пуст, либо состоит из корня (узла), связанного с двумя различными бинарными деревьями, называемыми *левым* и *правым поддеревом корня.* Деревья, у которых d > 2, называются *d-арными.*

Преобразование произвольного дерева с упорядоченными узлами в бинарное дерево сводится к следующим действиям. Сначала в каждом узле исходного дерева вычеркиваем все ветви, кроме самых левых ветвей. После этого в получившемся графе соединяем горизонтальными ветвями те узлы одного уровня, которые являются «братьями» в исходном дереве. (Поясним, что если несколько узлов имеют общего предка, то такие узлы называются «братьями»). В получившемся таким образом дереве левым потомком каждого узла *x* считается непосредственно находящийся под ним узел (если он есть), а в качестве правого потомка — соседний справа «брат» для *x,* если таковой имеется. На рис. 6.10 показаны этапы преобразования таким способом некоторого d-арного дерева в бинарное.

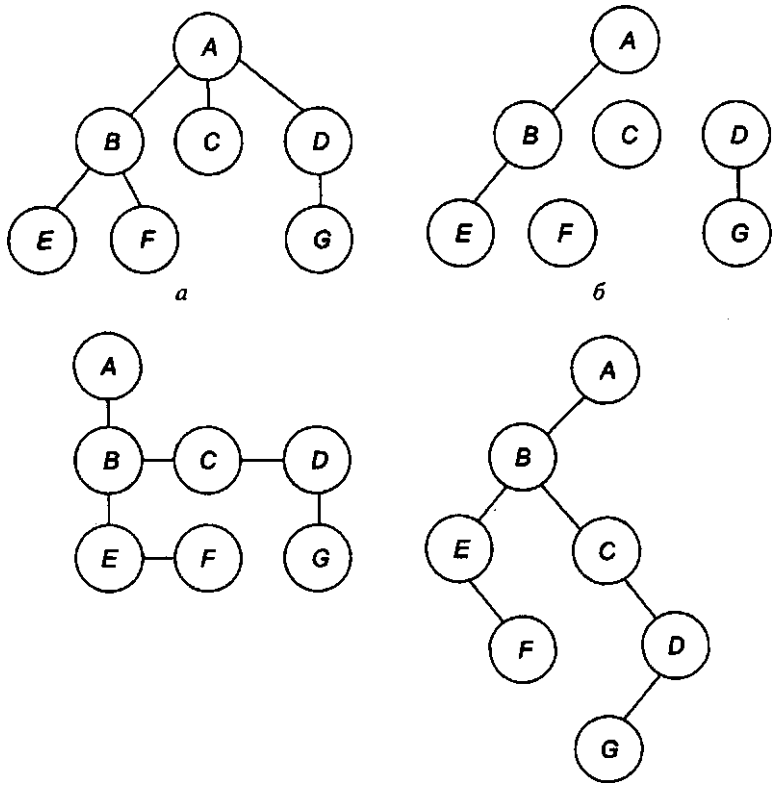


Рис. 6.10. Этапы преобразования d-арного дерева в бинарное ***(а-г)***

Переход от произвольного дерева к его бинарному эквиваленту не только облегчает анализ логической структуры, но также упрощает машинное представление, т.е. физическую структуру дерева.

Сбалансированное бинарное дерево. Бинарное дерево называется сбалансированным *(B-balanced),* если высота левого поддерева каждого узла отличается от высоты правого не более чем на 1 (рис. 6.11).

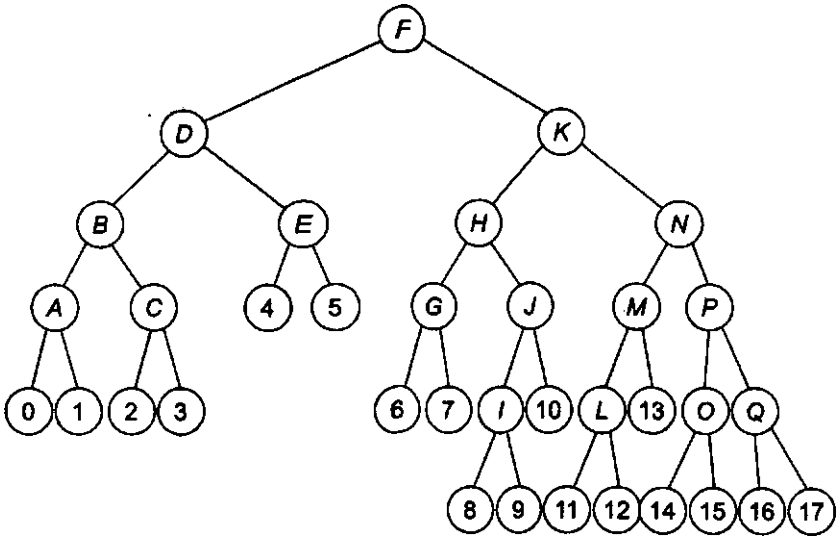


Рис. 6.11. Пример сбалансированного дерева

Сбалансированные бинарные деревья занимают промежуточное положение между оптимальными бинарными деревьями (все внешние узлы которых расположены на двух смежных уровнях) и произвольными бинарными деревьями.

Рассмотрим следующую структуру узлов сбалансированного бинарного дерева (табл. 6.2).

Таблица 6.2

Структура узлов сбалансированного дерева

Ключ	Указатель на левое поддерево	Указатель на правое поддерево	Показатель сбалансированности узла
KEY	LLINK	RLINK	B

B — показатель сбалансированности узла, т.е. разность высот правого и левого поддерева (B = +1; 0; —1).

При восстановлении баланса дерева по высоте учитывается показатель *B* (рис. 6.12).

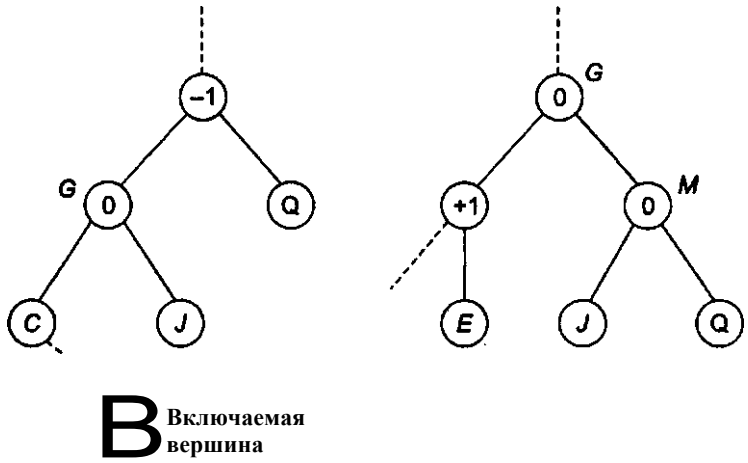


Рис. 6.12. Деревья с показателями сбалансированности узлов

Символы +1, 0, —1 на рис. 6.12 указывают, что левое поддерево выше правого, поддеревья равны по высоте, правое поддерево выше левого.

6.6. Поиск по бору

Особую группу методов поиска образует представление ключей в виде последовательности цифр и букв. Рассмотрим, например, имеющиеся во многих словарях буквенные высечки. Тогда по первой букве данного слова можно отыскать страницы, содержащие все слова, начинающиеся с этой буквы. Развивая идею побуквенных высечек, получим схему поиска, основанную на индексации в структуре бора (термин использует часть слова выборка).

Бор представляет собой тп-арное дерево. Каждый узел уровня *h* — это множество всех ключей, начинающихся с определенной последовательности из *h* литер. Узел определяет m-путевое разветвление в

зависимости от (л+1)-й литеры. Структуру бора можно представить в виде табл. 6.3.

Таблица 6.3

Структура бора

Символы	Узлы					
	1	2	3			*N*
Пробел (-)						

Пробел („) — обязательный символ таблицы.

В первом узле записывается первая буква ил\ цифра ключа. Во втором узле к ней добавляется еще один символ и т.д. Если слово, начинающееся с определенной буквы (цифры), ед инственное, то оно сразу записывается в первом узле.

Пример **1.** Дано множество

{A, AA, AB, ABC, ABCD, АБСА, ABCC, OR,
C, CC, CCC, CCCD, CCCB, CCCA}.

От исходного множества перейдем к построени! о бора. Исходный алфавит — *{A, B, C, D).*

BOR — единственное слово на букву B, и оно побуквенно не разбивается.

Узлы бора представляют собой векторы, кажд* w компонента которых представляет собой либо ключ, либо ссылку (в<>зможно пустую)— табл. 6.4.

Таблица 6.4

Пример поиска по бору

Символы	Узлы						
	1	2	3	4	5	6	*1*
A B C D	2 BOR 5	A_ AA 3	AB- 4	ABC- АБСА ABCC ABCD	C- 6	CC- 7	CCC CCCA CCCB CCCD

Узел 1 — корень, и первую букву следует искать здесь. Если первой оказалась, например, буква *B,* то из таблицы видно, что ему соответствует слово BOR. Если же первая буква *A,* то первый узел передает управление к узлу 2, где аналогичным образом отыскивается вторая буква. Узел 2 указывает, что вторыми буквами будут „ (пробел), *A, B* и т.д.

Пример 2. Пусть задано множество слов: воск, сок, оса, сто, сев, век, ост, сотка. Построим m-арное дерево для этого списка (рис. 6.13).

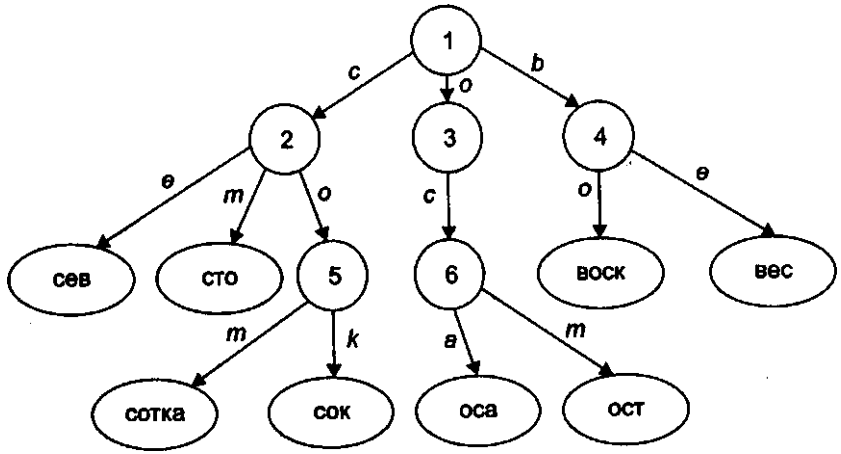


Рис. 6.13. т-арное дерево

На основе анализа этого дерева может быть построена табл. 6.5.

Таблица 6.5

Пример поиска по бору

Символы	Узлы					
	1	2	3	4	5	6
С О Т К А В Е	(2) (3) (4)	 (5) сто сев	(6)	 воск вес	 сотка сок	 ост оса

Следует отметить, что при построении бора порядок букв в первом столбце таблицы может быть произвольным. Естественно, что для каждого конкретного порядка букв заполнение таблицы будет разным. Допустим, для нашего примера в списке слов нужно найти слово «ост». Слово начинается на букву «о». Смотрим пересечение первого столбца со строкой, обозначенной буквой «о». Там записана ссылка (3); следовательно, обращаемся к третьему столбцу рора. Вторая буква слова «ост» — буква «с». На пересечении третьего столбца и строки с буквой «с» записана ссылка (6), которая отправляет нас на просмотр шестого столбца бора. В этом столбце видим два слова: «ост» и «оса». Но поскольку третьей буквой искомого слова является буква «т», то на строке, обозначенной этой буквой, находим слово «ост».

6.7. Поиск хешированием

В основе поиска лежит переход от исходного множества к множеству хеш-функций *h(k).* Хеш-функция имеет следующий вид: *h(k)* = *к* mod m, где *к* — ключ; *m* — целое число; mod • остаток от целочисленного деления.

Например, пусть дано множество {9, 1, 4, 10| 8, 5}. Определим для него хеш-функцию *h{k)* = *к* mod *m.*

• Пусть m = 1, тогда *h(k)* = {О, 0, 0, 0, 0, 0^ Множество хеш-функций состоит из нулей.

• Пусть тп = 20, тогда *h(k)* = {9, 1, 4, 10, 8, S}. Множество хеш-функций повторяет исходное множество.

• Пусть m равно половине максимального ключа m = $[K_{max}/2]$, тогда *m* = [10/2] = 5; *h(k)* = {4, 1, 4, 0, 3, 0}.

Хеш-функция указывает адрес, по которому следует отыскивать ключ. Для разных ключей хеш-функция может принимать одинаковые значения, такая ситуация называется *коллизией.* Таким образом, поиск хешированием заключается в устранении коллизий методом цепочек (рис. 6.14).

Пример 1. Дано множество ключей {7, 13, 6, 3, 9, 4, 8, 5}. Найти ключ *К* = 27.

Хеш-функция равна *h(k)* = *k*mod *m;m* = [13/2] — 6 (так как 13 • максимальный ключ); *h(k)* = {1, 1, 0, 3, 3, 4, 2,|5}. Для устранения коллизий построим табл. 6.6.

Рис. 6.14. Устранение коллизий методом цепочек

Таблица 6.6

Разрешение коллизий

h(k)	Цепочки ключей	/и»	Цепочки ключей
0	6	3	3,9
1	7,13	4	4
2	8	5	5

Попарным сравнением множества хеш-функций и множества исходных ключей заполняем таблицу.

При этом хеш-функция указывает адрес, по которому следует отыскивать ключ. Например, если отыскивается ключ *К* = 27, тогда

$$/*(£) = 27 \bmod 6 = 3.$$

Это значит, что ключ *К = 27* может находиться только в строке, где *h(k)* = 3. Так как его там нет, то данный ключ отсутствует в исходном множестве.

Пример 2. Дано множество ключей

{7, 1, 8, 5, 14, 9, 16, 3, 4}.

Найти ключ *К* = 14.

Хеш-функция равна *h(k)* = *к* modm, где m = [16/2] = 8 (так как 16 — максимальный ключ). Следовательно,

h(k) = {7, 1, 0, 5, 6, 1, 0, 3, 4}.

Для разрешения коллизий, которые присутствуют во множестве хеш-функций, построим табл. 6.7.

Таблица 6.7

Разрешение коллизий

h(k)	Цепочки ключей	Л(*0		Цепочки ключей
0	8,16	5		5
1	1,9	6		14
3	3	7		7
4	4			

При заполнении таблицы установим соответствие между исходным множеством и множеством хеш-функций. Поиск осуществляется по таблице: $K = 14$; $h(k) = 14 \bmod 8 = 6$. Это значит, что ключ $K = 14$ может быть только в строке со значением $h(k) = 6$|

6.8. Алгоритмы поиска словесной информации

В настоящее время наличие сверхпроизводительных микропроцессоров и дешевизна электронных компонентов позволяют делать значительные успехи в алгоритмическом моделировании. Рассмотрим несколько алгоритмов обработки слов.

Алгоритм Кнута — Морриса — Пратта (К1ЮП). Данный алгоритм получает на вход слово $X = x[1]x[2]... x[n]$ и просматривает его слева направо, буква за буквой, заполняя при этом массив натуральных чисел ... *1[n],* где *l[i]* — длина слова *1(x[1]... x[i\).* Таким образом, *l[i]* есть длина наибольшего начала слова *x[1]...x[г],* одновременно являющегося его концом.

Пример. Используя алгоритм КМП, определить, является ли слово *A* полсловом слова *B.*

Решение. Применим алгоритм КМП к слов^ *A#B,* где # — специальная буква, не встречающаяся ни в *A,* ни в В . Слово *A* является полсловом слова *B* тогда и только тогда, когда среди чисел в массиве *I* будет число, равное длине слова *A.*

Предположим, что первые *i* значений /[1].. уже найдены, Читается очередная буква слова, т.е. *x[i* + 1], и вычисляется *1[г* +1].

Другими словами, нужно определить начала *Z* слова x[1]...*x[i* + 1], одновременно являющиеся его концами. Из них следует выбрать самое длинное (рис. 6.15).

1 /+ 1

Прочитанная часть X

Рис. 6.15. Анализ слова

Получаем следующий способ отыскания слова *Z.* Рассмотрим все начала слова ж[1].. .х[г], являющиеся одновременно его концами. Из них выберем подходящие — те, за которыми следует буква *x[г* + 1]. Из подходящих выберем самое длинное. Приписав в его конец *x[г* + 1], получим искомое слово *Z.* Теперь пора воспользоваться сделанными приготовлениями и вспомнить, что все слова, являющиеся одновременно началами и концами данного слова, можно получить повторными применениями к нему функции *I.*

Получим следующий фрагмент программы.

```
i:=1; l[1]:=0;
{таблица l[l]..l[i] заполнена правильно}
while i O n do begin
    len:= l[i];
    {len — длина начала слова x[1].. .x[i], которое является
        его концом; все более длинные начала
        оказались неподходящими}
    while (x[len+I]Ox[i+l]) and (len>0) do begin
        {начало не подходит, применяем к нему функцию 1}
        len:=l[len];
    end;
    {нашли подходящее слово или убедились в его отсутствии}
    if x[len+l]=x[i+l] do begin
        {x[l]..x[len] — самое длинное подходящее начало}
        l[i+l]:=len+l;
    end else begin
        {подходящих нет}
        l[i+l]:=0;
    end;
    i:=i+l;
```

Представим алгоритм, который позволяет проверить, является ли слово *X = x[1]. ..x[n]* полсловом слова *Y = y[1]. ..y[m].*

Решение. Вычисляем таблицу *1[1]... 1[n],* как раньше.

```
j:=0; 1еп:=0;
 {len — длина максимального начала слова X, одновременно
          являющегося концом слова y[1]...y[j]}
while (lenOn) and (j<>m) do begin
    while (x[len+l]Oy[j+l]) and (len>0) do begin
        {начало не подходит, применяем к нему функцию 1}
        len: = l[1еп];
    end;
    {нашли подходящее слово или убедились в его отсутствии}
    ifx[len+l]=y[j+l] do begin
        { x[1].. .x[len] — самое длинное подходящее начало}
        1еп:=1еп+1;
    end else begin
        {подходящих нет}
        1еп:=0;
    end;
    j:=j+i;
end;
{если len = п, слово X встретилось; иначе мы дошли до конца
          слова Y, так и не встретив X]
```

Алгоритм Бойера — Мура (БМ). Этот алгоритм делает то, что на первый взгляд кажется невозможным: в типичной ситуации он читает лишь небольшую часть всех букв слова, в котором ищется заданный образец. Пусть, например, отыскивается образец ***abed.*** Посмотрим на четвертую букву слова: если, к примеру, это буква е, то нет никакой необходимости читать первые три буквы. (В самом деле, в образце буквы е нет, поэтому он может начаться-не раньше пятой буквы.)

Приведем самый простой вариант этого алгоритма, который не гарантирует быстрой работы во всех случаях. Пусть *x[1]...x[n]* — образец, который надо искать. Для каждого символа *s* найдем самое правое его вхождение в слово *X,* т.е. наибольшее *к,* при котором *x[к] = s.* Эти сведения будем хранить в массиве pos[s]; если символ *s* вовсе не встречается, то будет удобно предположить pos[s] = 0.

Решение.

```
Принять все pos[s] равными 0
for i:=l to n do begin
    pos[x[i]]:=i;
end;
```

В процессе поиска будем хранить в переменной last номер буквы в слове, против которой стоит последняя буква образца. Вначале last = п (длина образца), затем last постепенно увеличивается.

```
last:=n;
{все предыдущие положения образца уже проверены}
while last <= m do begin {слово не кончилось}
    if x[m] <> y[last] then begin {последние буквы разные}
        last := last + (n - pos[y[last]]);
        {n - pos[y[last]] — минимальный сдвиг образца, при котором
            напротив y[last] встанет такая же буква в образце.
            Если такой буквы нет вообще, то сдвигаем
            на всю длину образца}
    end else begin
        {если нынешнее положение подходит, т.е. если
            x[i]... x[n] = y[last - п + 1]... y[last],
            то сообщить о совпадении}
        last:=last+l;
    end;
end;
```

Алгоритм Рабина. Этот алгоритм основан на простой идее. Представим себе, что в слове длиной *m* ищется образец длиной *n.* Вырежем окошко размером *n* и будем двигать его по входному слову. При этом проверяем, не совпадает ли слово в окошке с заданным образцом. Сравнивать по буквам долго. Вместо этого фиксируем некоторую функцию, определенную на словах длиной *n.* Если значения этой функции на слове в окошке и на образце различны, то совпадения нет. Только если значения одинаковы, нужно проверять совпадение по буквам.

Выигрыш при таком подходе состоит в следующем. Чтобы вычислить значение функции на слове в окошке, нужно прочесть все буквы этого слова. Так уж лучше их сразу сравнить с образцом. Тем не менее выигрыш возможен, так как при сдвиге окошка слово не

меняется полностью, а лишь добавляется буква в конце и убирается в начале. Заменим все буквы в слове и образце их номерами, представляющими собой целые числа. Тогда удобной функцией является сумма цифр. (При сдвиге окошка нужно добавить новое число и вычесть пропавшее.) Выберем некоторое число *p* (желательно простое) и некоторый вычет *x* по модулю *p*. Каждое слово длиной п представим как последовательность целых чисел (заменив буквы кодами). Эти числа будем рассматривать как коэффициенты многочлена степени ***n*** — 1 и вычислим значение этого многочлена по модулю *p в* точке *x*. Это и будет одна из функций семейства (для каждой пары *p* и *x* получается, таким образом, своя функция). Сдвиг окошка на 1 соответствует вычитанию старшего члена *(x*~[1] следует вычислить заранее), умножению на *x* и добавлению свободного члена. Следующее соображение говорит в пользу того, что совпадения не слишком вероятны.

Пусть число *p* фиксировано и к тому же простое, а *X* и *Y* — два различных слова длиной ***n.*** Тогда им соответствуют различные многочлены (предполагаем, что коды всех букв различны — это возможно, если *p* больше числа букв алфавита). Совпадение значений функции означает, что в точке *x* эти два различных многочлена совпадают, т.е. их разность обращается в 0. Разность есть многочлен степени ***n*** — 1 и имеет не более ***n*** - 1 корней. Таким образом, если ***n*** много меньше *p,* то у случайного *x* мало шансов попасть в неудачную точку.

Контрольные вопросы

1. Что понимается под поиском?
2. Каковы особенности последовательного и бинарного поиска?
3. Каковы особенности интерполяционного и фибоначчиевого поиска?
4. Каковы особенности поиска по бинарному дереву?
5. Каковы особенности поиска по бору и хешированием?
6. В чем состоит методика анализа сложности алгоритмов поиска?
7. В чем заключается особенность алгоритмовпоиска словесной информации?
8. Что такое коллизия?
9. Какой метод используется для разрешения коллизий?
10. Что определяет показатель сбалансированности узла дерева?
11. Назовите алгоритмы поиска словесной информации.

Глава 7
Итеративные и рекурсивные алгоритмы

Эффективным средством программирования для некоторого класса задач является рекурсия. С ее помощью можно решать сложные задачи численного анализа, комбинаторики, алгоритмов трансляции, операций над списковыми структурами и т.д. Программы в этом случае имеют небольшие объемы по сравнению с итерацией и требуют Меньше времени на отладку.

Т1од,рекурсией понимают способ задания функции через саму себя, например способ задания факториала в виде

$$n! = (n - 1)! \cdot \boldsymbol{n}.$$

В программировании под рекурсивной процедурой (функцией) понимают способ обращения процедуры (функции) к самой себе.

Под *итерацией* понимают результат многократно повторяемой какой-либо операции, например представление факториала в виде

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot \boldsymbol{n}.$$

Среди широкого класса задач удобно представлять с использованием рекурсивных процедур (функций) те задачи, которые сводятся на подзадачи того же типа, но меньшей размерности.

Обща* методика анализа рекурсии содержит три этапа:

1 • Параметризация задачи, заключающаяся в выделении различных элементов, от которых зависит решение, в частности размерности решаемой задачи. После каждого рекурсивного вызова размерность должна убывать.

2. Поиск тривиального случая и его решение. Как правило, это ключевойэтап в рекурсии, размерность задачи при этом часто равна 0 или 1.

3- А*омпозиция общего случая, имеющая целью привести задачу к одной т[и] нескольким задачам того же типа, но меньшей размерности.

Рассиотрим понятие итеративного и рекурсивного алгоритмов на примере счисления факториала.

7.1. Итеративный алгоритм

Наиболее простой и естественной формой представления итеративного алгоритма при реализации на компьютере является его описание с использованием цикла.

Рассмотрим алгоритм итеративного алгоритма вычисления факториала **n!**

В соответствии с определением функции **n!** имеем

$$n! = \begin{cases} 1, & \text{если } n = 0; \\ 1 \cdot 2 \cdot \ldots \cdot n, & \text{если } n \neq 0. \end{cases}$$

Вначале в зависимости от текущего значения происходит выбор способа вычисления *n*\. Если *n* = 0, то переменная *fctrl* принимает значение 1. Если *n* ф 0, в цикле вычисляется произведение 1 • 2 -... - **n.** Переменная г — это параметр цикла, который последовательно принимает значения 2, 3, 4 и т.д. до *n* включительно. Для каждого значения параметра цикла выполняется тело цикла:

fctrl := fctrl i.*

Последовательность итераций цикла для *n* = 6 показана в табл. 7.1.

Под итерацией цикла будем понимать выполнение тела цикла для конкретного значения параметра цикла.

Программа состоит из процедуры-функции FACTORIAL и основной программы. В основной программе происходит ввод значения *N,* вызов процедуры-функции и печать результата.

```
{Итеративное вычисление факториала}
PROGRAM FI;
VAR
FAC: LONGINT;
N: INTEGER;
{Функция вычисления факториала}
FUNCTION FACTORIAL (N: INTEGER): LONGINT;
VAR F: LONGINT;
I: INTEGER;
```


Таблица 7.1

Пошаговое вычисление факториала

		Состояние		
		i	*fctrl*	n
1-я итерация г < *п — •*	Гг:=2 \/сЬI:=*fctrl *i*	2	1	6
		i_ 2	2	6
2-я итерация г < *п —>*	fi:=t + 1 *[fctrl :=fctrl *i*	3	2	6
		3	6	6
3-я итерация г < тг -+	*(i:=i + l* *{fctrl := fctrl *i*	4	6	6
		4	24	6
4-я итерация г < п —•	ft:=» + 1 *\fctrl:= fctrl *i*	5	24	6
		5	120	6
5-я итерация г < п —>	ft:=t + 1 *{fctrl := fctrl *i*	6	120	6
		6	720	6

```
BEGIN
   IF (N=0) OR (N=1) THEN FACTORIAL:=1 ELSE
   BEGIN
      F:=1;
      FOR I: = 2 TO N DO
         F: = F * I;
      FACTORIAL:=F;
   END;
END;
{Основная программа}
BEGIN
   WRITELN ('ВВЕДИТЕ ЗНАЧЕНИЕ N')
   READLN (N);
   FAC:= FACTORIALS); {Вызов функции FACTORIAL}
   WRITELN(' ФАКТОРИАЛ =', FAC);
   READLN;
END.
```

Рекурсивный алгоритм

Рекуррентные соотношения довольно часто встречаются в математических выражениях. Рекурсия в определении состоит в том, что определяемое понятие определяется через само это понятие. Рекурсия в вычислениях выступает в форме рекуррентных соотношений, которые показывают, как вычислить очередное значение, используя предыдущие.

Например, рекуррентное соотношение **X» = Xi_2 + X i _ i ,** где **XI** = 1, **X2** = 2, задает правило вычисления так называемых чисел Фибоначчи.

Другим примером рекуррентных соотношений могут служить правила вычисления членов арифметической прогрессии

$$\mathbf{On+i} = a_{,,} + d,$$

где *d* — разность прогрессии, либо геометрической профессии

$$\mathbf{п+1} = Q<ln,$$

где *q* — коэффициент прогрессии.

Рекурсивное представление факториала имеет вид:

$$п! = (п— 1)! \cdot п.$$

Проведем анализ рекурсивного вычисления факториала.

1. Параметризация. В данном случае имеется всего один параметр п — целое число.

2. Поиск тривиального случая. При ***n*** = 0 или ***n*** = 1 значение факториала равно 1, что соответствует выходу из рекурсии.

3. Декомпозиция общего случая: п! вычисляется через меньшую размерность этой же задачи (п — 1)!.

Покажем алгоритм вычисления факториала для ***n*** = 4 (рис. 7.1).

Сначала образуется так называемый рекурсивный фрейм 1 при п = 4. *Фрейм* — структура, содержащая некоторую информацию. Для этого фрейма отводится память и в нем фиксируются все значения переменных тела функции при ***n*** = 4. Отметим, что в рекурсивном фрейме фиксируются значения всех переменных функции, кроме глобальных.

Затем происходит вызов Factorial(n) при ***n*** = 3. Образуется фрейм 2, где фиксируются значения переменных тела функции при п = 3.

Фрейм 1

If *n>0* **then** Factorials Factqrial(3)*4
else Factorial:^

Фрейм 2

n=3
if *n>0* **then** Factorial- Factqrial(2)*3
else Factorials

Фрейм 3

n=2 ' ;
If n>0 **then** Factorials Factqrial(1)*2
else Factorial^

Фрейм 4

If л>0 **then** Factorials Factqrial(0)*1
else Factorials!

Фрейм S

/7=0
Ifn>0then i
else Factorialsl

Рнс. 7.**1.** Рекурсивное вычисление факториала

При этом фрейм 1 также хранится в памяти. Из фрейма 2 происходит обращение к Factorial(n) при п = 2. В результате этого обращения образуется фрейм 3, где фиксируются значения переменных тела "функции при п = 2 и т.д. до тех пор, пока при очередном обращении к функции Factorial условие ***n*** > 0 не примет значение false.

Это произойдет в фрейме 5. В этом фрейме получим значение , Factorial = 1 и передадим это значение в фрейм 4.

После этого фрейм 5 будет уничтожен, так как обращение ,Factorial(n) при *n* = 0 будет выполнено.

В фрейме 4 вычислим значение Factorial(n) для ***n*** = 1. После этого передадим это значение во фрейм 3, а фрейм 4 будет закрыт, так как обращение к Factorial(n) при ***n*** = 1 будет закончено.

Так будет сворачиваться цепочка фреймов в последовательности, обратной той, в которой их порождали, пока не свернем фрейм 1. После этого вычисление функции будет окончено.

Программа рекурсивного алгоритма вычисления факториала будет иметь вид:

```
{Рекурсивное вычисление факториала}
Program FR;
VAR
fac: longint;
n: integer;
{Функция вычисления факториала}
Function factorial (n: integer): longint;
Begin
    If (n=0) or (n=l) then factorial := I
    Else factorial := factorial (n-l)*n;
End;
{Основная программа}
Begin
    Writeln('ВВеfl>тге значение n');
    Readln(n);
    fac: = factorial(n); {Вызов функции FACTORIAL}
    >\тке1п('факториал =', fac);
    Readln;
End.
```

Функция вычисления факториала имеет следующие особенности:

• при вычислении факториала происходит обращение функции к самой себе (подчеркнуто в выражении), но с меньшим значением аргумента ***n*** — 1 по сравнению с первым вызовом ***n:***

factorial := factorial(n - 1) * ***n;***

• при вычислении факториала не используется цикл, что является существенной особенностью рекурсивного алгоритма.

Рассмотрим последовательность действий при рекурсивном вычислении факториала для ***n*** = 3:

1) внешний вызов из основной программы factorial(3);

2) первый рекурсивный вызов factorial(2) в операторе

factorial := factorial(n — 1) * n,

где не происходят никакие вычисления—только вызов (подчеркнуто);

3) второй рекурсивный вызов factorial(l);

4) получение значения factorial(l) := 1;

5) возврат из второго рекурсивного вызова и вычисление факториала factorial(2) := 1 * 2 = 2;

6) возврат из первого рекурсивного вызова и вычисление факториала factorial(3) := 2 * 3 = 6;

7) возврат в основную программу fac := 6.

Замечание. В программу рекурсивного вычисления факториала можно добавить стандартную функцию текущего времени

GETTIME(Var Hour, Minute, Second, SeclOO: WORD).

7.3. Рекурсивные структуры данных

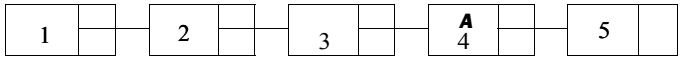
Список—набор элементов, расположенных в определенном порядке.

Список очередности — список, в котором последний поступающий элемент добавляется к нижней части списка.

Список с использованием указателей — список, в котором каждый элемент содержит указатель на следующий элемент списка.

Однонаправленный и двунаправленный список — это линейный список, в котором все исключения и добавления происходят в любом месте списка.

Однонаправленный список отличается от двунаправленного списка только связью, т.е. в однонаправленном списке можно перемещаться только в одном направлении (из начала в конец), а в двунаправленном — в любом (рис. 7.2).



и - н **н - н** И-Н 5

б

Рис. 7.2. Однонаправленный и двунаправленный списки

На рис. 7.3 и 7.4 показано, как добавляется и удаляется элемент из двунаправленного списка. При добавлении нового элемента (обозначен *N)* связь от 3 идет к *N,* а от *N* к 4, а связь между 3 и 4 удаляется.

В однонаправленном списке структура добавления и удаления такая же, только связь между элементами односторонняя.

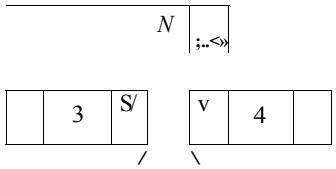


Рис. 7.3. Добавление элемента в список

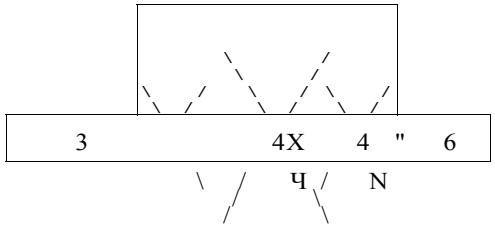


Рис. 7.4. Удаление элемента из списка

Очередь—тип данных, при котором новые данные располагаются следом за существующими в порядке поступления; поступившие первыми данные при этом обрабатываются первыми.

Очередь иногда называют циклической памятью или списком типа FIFO («first-in-first-out» — «первым пришел — первым обслуживается»). Другими словами, у очереди есть голова и хвост.

В очереди новый элемент добавляется только с одного конца (рис. 7.5).

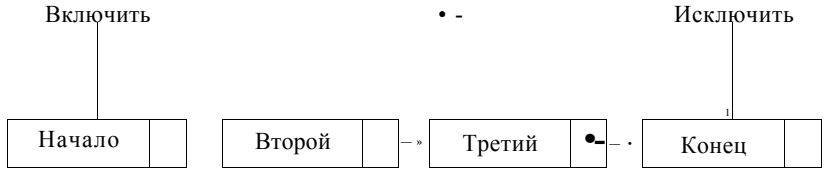


Рис. 7.5. Структура очереди

Удаление элемента происходит на другом конце. В данном случае это может быть только четвертый элемент. Очередь — по сути однонаправленный список, только добавление и исключение элементов происходит на концах списка.

Стек — линейный список, в котором все включения и исключения делаются в одном конце списка. Стек называют (push-down) списком, реверсивной памятью, гнездовой памятью, магазином, списком типа LIFO («last-in-first-out» — «последним пришел — первым обслуживается»). Стек — часть памяти ОЗУ компьютера, которая предназначается для временного хранения байтов, используемых микропроцессором. Действия со стеком производятся при помощи регистра указателя стека. Любое повреждение этой части памяти приводит к фатальному сбою.

Логическая структура стека представлена на рис. 7.6.

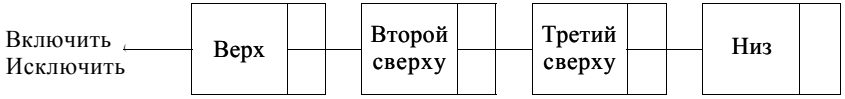


Рис. 7.6. Структура стека

Важнейшие операции доступа к стеку — включение элементов и исключение элементов — осуществляются с вершины стека, причем в каждый момент для исключения или включения доступен один элемент, находящийся на вершине стека.

Вершина стека адресуется с помощью специального указателя. Для включения нового элемента в стек указатель сначала перемещается «вверх» (возможна и другая конфигурация стека, когда стек «надстраивается снизу») на длину слота, или ячейки, а затем по значению указателя (индекса) в стек помещается информация о новом элементе. При исключении элемента из стека сначала прочитывается информация об исключаемом элементе по значению указателя (индекса), а затем указатель смещается «вниз» (или «вверх» при обратной конфигурации стека) на один слот. Стек считается пустым, если указатель смещен «вниз» на длину одной ячейки относительно нижней границы стека. Стек считается полным, если вершина стека (указатель стека) совмещается с верхней границей стека.

П р и м е р 1. Пусть имеется множество элементов {1, 4, 6, 9} и к стеку применена последовательность операций: (/, /, *O, I, O, O,*

/, О), где символ / означает запись элемента в стек, а символ *О* — считывание элемента из стека. Этапы выполнения операций показаны на рис. 7.7.

А .

64

9164

Рис. 7.7. Этапы выполнения операций

Дек (стек с двумя концами) — линейный список, в котором все включения и исключения делаются на обоих концах списка (рис. 7.8). Еще один термин «архив» применялся к декам с ограниченным выходом, а деки с ограниченным входом называли «перечнями», или «реестрами».

Включить или исключить

Включить или исключить

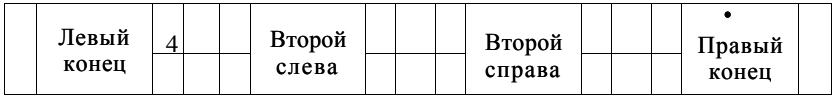


Рис. 7.8. Структура дека

Циклически связанный список обладает той особенностью, что связь его последнего узла идет назад к первому узлу списка (рис. 7.9).

Рис. 7.9. Структура циклически связанного списка

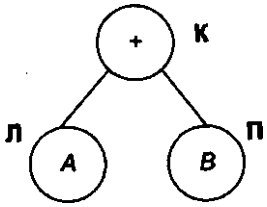
В этом случае можно получить доступ к любому элементу, находящемуся в списке, отправляясь от любой заданной точки. При этом не приходится различать в списке «последний» или «первый» узел.

7.4. Виды обхода бинарных деревьев

Набор способа обхода дерева позволяет ввести отношение порядка для узлов дерева.

Наиболее распространены три способа обхода узлов дерева (рис. 7.10), которые получили следующие названия:

- обход в направлении слева направо (обратный порядок, инфиксная запись);
- сверху вниз (прямой порядок, префиксная запись);
- снизу вверх (концевой порядок, постфиксная запись).



КПП (Корень - Левая - Правая вершины)
Прямой обход + *AB* (префиксная запись)

ЛКП Обратный обход *A+B* (инфиксная запись)

ЛПК Концевой обход *AB+* (постфиксная запись)

Рис. 7.10. Три различных обхода по бинарному дереву и соответствующие формы

В результате обхода дерева, приведенного на рис. 7.11, порождаются следующие последовательности прохождения узлов:

abcdefghi (прямой порядок);

cdbfhigea (концевой порядок).

Рекурсивно можно представить не только алгоритм решения задачи, но и обрабатываемую информацию.

Пример. Пусть задано арифметическое выражение

$$(((A-B)*C) \quad + \quad \{D/(E^F))).$$

Возможно описание этого арифметического выражения с помощью бинарного дерева — рис. 7.12.

Для прохождения этого дерева в прямом порядке начинаем с корня (узел +). Затем пройдем в прямом порядке левое поддерево (с корнем, помеченным *) и далее правое поддерево в прямом порядке

а *б*

Рис. 7.11. Способы обхода бинарного дерева

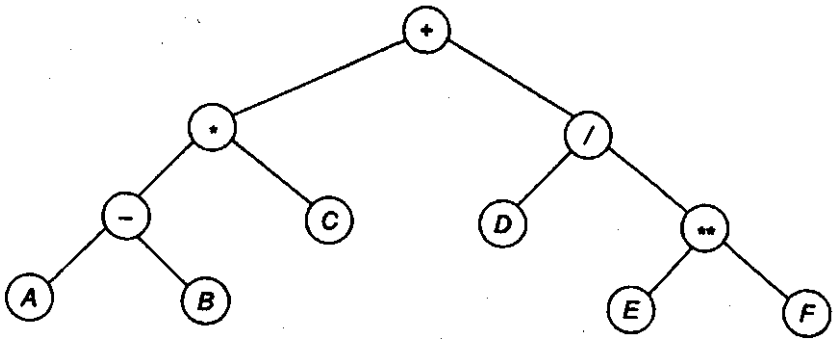


Рис. 7.12. Бинарное дерево для описания арифметического выражения

(с корнем, помеченным /). Прохождение левого поддерева в прямом порядке начинается с корня (помеченного *), затем следует прямое прохождение его левого поддерева (порождающее последовательность ***-AB)*** и далее правого поддерева (последовательность С). Поэтому вся последовательность, порождаемая прохождением левого поддерева основного корня, будет ****-ABQ*** Аналогично прохождение в прямом порядке правого поддерева с корнем, помеченным /, порождает последовательность ***/D * *EF.*** Таким образом, прохождение всего дерева в прямом порядке порождает последовательность

+	****-ABC***	***/P * *EF***
К	***Л***	***п***

Прохождение этого дерева в обратном и концевом порядках порождает соответственно последовательности:

A-B*C	***+***	***D/E**F***	(инфиксная);
Л	***К***	***П***	
AB-C*	***DEF**/***	***+***	(постфиксная).
Л	***п***		

Заметим, что во всех трех выражениях порядок вхождения переменных совпадает; меняется только порядок знаков операций. При этом ни одно из этих выражений не имеет скобок, и, таким образом, если не заданы правила приоритета, значение приведенного выше выражения в инфиксной форме нельзя вычислить однозначно.

Для бинарного дерева на рис. 7.13 имеем следующий порядок обхода узлов:

ABFHGIJKCDE — для прямого обхода;
HFBGJIKACDE — для обратного обхода;
HFJKIGBEDCA — для концевого обхода.

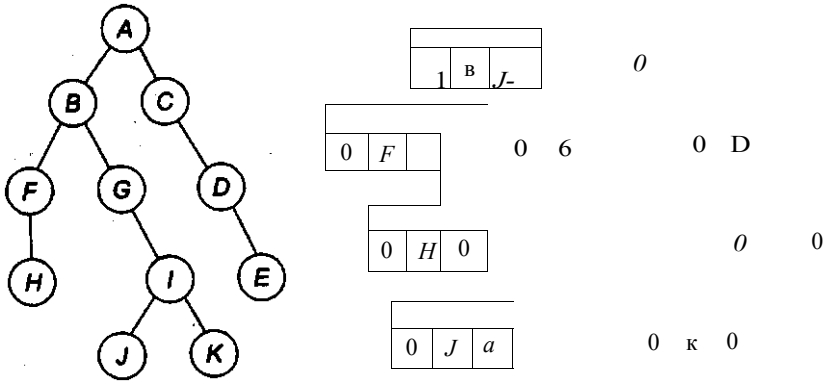


Рис. 7.13. Пример логической структуры ***(а)*** и спискового представления (б) бинарного дерева

Контрольные вопросы

1. Что понимается под рекурсией и итерацией в математике?
2. Каковы особенности итеративного алгоритма?

3. Каковы особенности рекурсивного алгоритма?
4. В чем состоит методика анализа рекурсивного алгоритма?
5. В каких случаях целесообразно использовать рекурсивный или итеративный алгоритм?
6. Приведите примеры итерации и рекурсии.
7. Все ли языки программирования дают возможность рекурсивного вызова процедур?
8. Укажите виды обхода бинарных деревьев.
9. Приведите пример рекурсивной структуры данных.
10. Что такое указатели и динамические переменные в алгоритмических языках?

Глава 8
Основные определения теории графов

Теория графов дает простой, доступный и мощный инструмент построения моделей и решения инженерных задач. В настоящее время существует множество проблем, где требуется построить сложные системы с помощью определенного упорядочения их элементов. Сюда относятся планирование промышленного производства, задачи теории сетевого планирования и управления (СПУ), тактические и логические задачи, проблемы построения систем связи и исследования процессов передачи информации, выбор оптимальных маршрутов и потоков в сетях, методы построения электрических сетей, задачи идентификации в органической химии и способы построения переключательных схем. Такими же являются большой круг экономических задач, проблемы выбора структуры социальных групп, игровые задачи и головоломки и т.д. Таким образом, область возможных применений теории графов очень широка..

Граф (сеть) $G = (V, E)$ состоит из конечного непустого множества m вершин ($m > 1$) и конечного множества п неупорядоченных пар элементов (м, и) ($n > 0$), называемых ребрами (рис. 8.1).

Две вершины ***и*** и ***v*** являются *смежными,* если в графе G существует ребро ***(и, v);*** в противном случае и и и *независимы.* Про ребро ***(и, v)*** также говорят, что оно *инцидентно* вершинам ***и*** и ***v.***

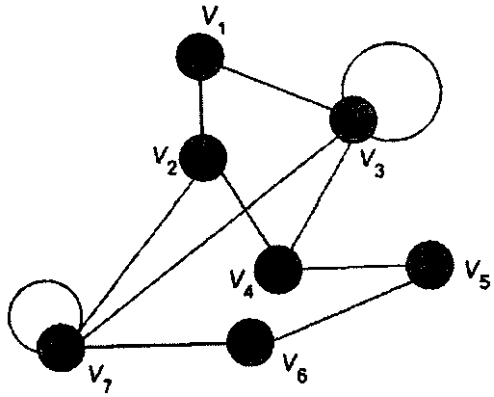


Рис. 8.1. Графовая структура

Приведем основные определения.

Граф называется *вырожденным,* если у него нет ребер.

Граф называется *связным,* если для любых двух вершин существует путь, соединяющий эти вершины.

Подграф называется *остовным подграфом,* если множество его вершин совпадаете множеством вершин исходного графа.

Гранью графа, изображенного на некоторой поверхности, называется часть поверхности, ограниченная ребрами графа.

Пустым называется граф без ребер;

Полным называется граф, в котором каждые две вершины смежные (рис. 8.2).

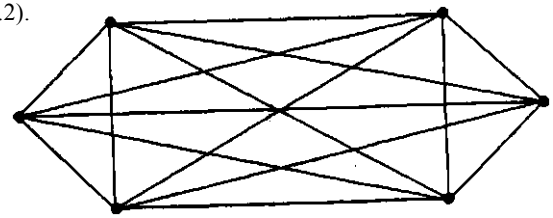


Рис. 8.2. Полный граф

Граф называется полным, если каждые две различные вершины его соединены одним и только одним ребром.

Мост — точка сочленения графа, удаление которой (и всех инцидентных ей ребер) увеличивает число компонентов связности. Например, если сеть поставлена в соответствие коммуникационным

линиям (транспортные сети, линии электропередачи, телефонные линии и т.п.), то удаление моста из сети соответствует разрыву коммуникаций (рис. 8.3).

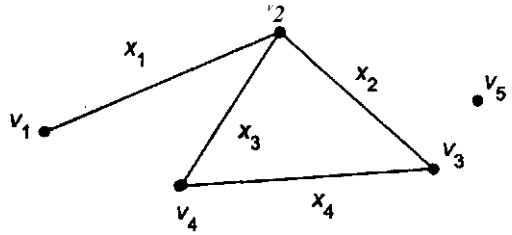


Рис. 8.3. Висячая *ы* и изолированная v_5 вершины

Ребро, соединяющее вершину саму с собой, называется *петлей.*

Если сеть состоит из конечного множества вершин и множества упорядоченных пар *[u, v)* различных вершин, то такая сеть называется *ориентированной (орграфом) (тр»с.8AV(G₃) = {u, v, w, x, y, z}* и *E(G₃) = {[u, v], [v, w], [v, x], [x, w], [w, y], [w, z], [z, v}}.*

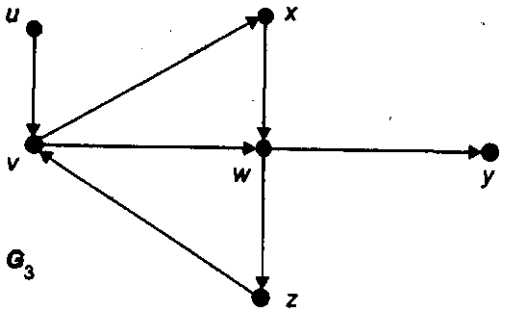


Рис. 8.4. Орграф

При работе с ориентированной сетью *G = (V, E)* элементы *V* обычно называют *узлами,* а элементы *E — дугами.* Если *[u, v]* — дуга ориентированной сети, то мы говорим, что *и* — левый сосед *v,* а *v* — правый сосед *и.*

В большинстве применений можно без потери смысла заменить ненаправленную дугу на двунаправленную, а двунаправленную — на две однонаправленные дуги, например так, как показано на рис. 8.5.

Рис. 8.5. Дуги графа

Взвешенная сеть — это такая сеть, ребрам или дугам которой поставлены в соответствие действительные числа или величины, принимающие действительные значения (рис. 8.6). При изображении сети веса или весовые коэффициенты не обязательно должны соответствовать масштабу изображения.

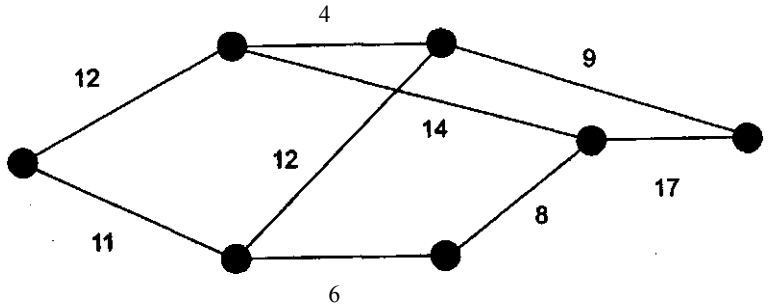


Рис. 8.6. Взвешенная сеть

Если две вершины соединяются более чем одним ребром, то такие ребра называются *кратными,* или *мультиребрами.* Граф с кратными ребрами называют *мультиграфом,* граф с кратными ребрами и петлями — *псевдографом.*

8.1. Изоморфизм графов

Два графа называют *изоморфными,* если существует взаимно-однозначное соответствие между их вершинами, а именно: число ребер, соединяющих любые две вершины одного графа равно числу ребер, соединяющих соответствующие вершины другого графа. Три графа изоморфны (рис. 8.7) при соответствии:

АъР; B++R; C **о Г;** *E S; F^U.*

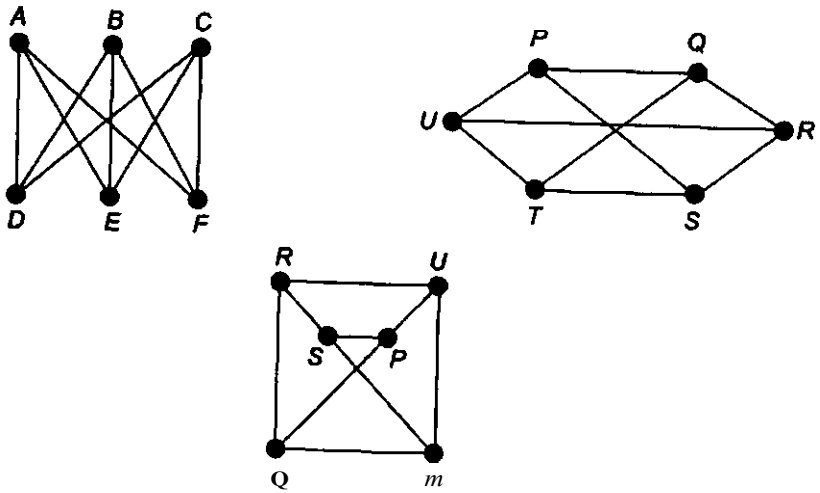


Рис. **8**.7. Изоморфные графы

Понятие изоморфизма для графов имеет наглядное толкование Представим ребра графов эластичными нитями, связывающими узлы-вершины. Тогда изоморфизм можно представить как перемещение узлов и растяжение нитей. Покажем, что следующие два графа на рис. 8.8 изоморфны.

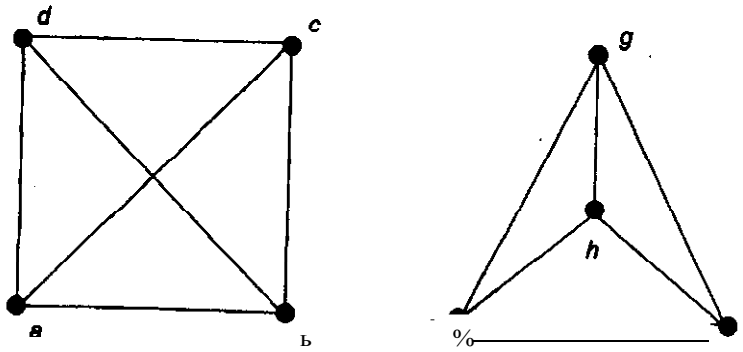


Рис. **8.8.** Изоморфные графы

Действительно, отображение *a « e, Ь « f, c » д, d h,* являющееся изоморфизмом, легко представить как модификацию первого графа, передвигающую вершину *d* в центр рисунка.

8.2. Степень вершины графа

Степень вершины v, обозначаемая как d_v или deg(u), равна числу ребер, инцидентных выбранной вершине, т.е. d_v = = deg(u) = |ЛГ(и)|. Таким образом, если вершины в *G* (рис. 8.9) расположить в порядке *w, v, x, z, u, y, t,* то их соответствующие степени равны 4, 4, 2, 2, 1, 1, 0, т.е. = 4, *dv* = 4, d_x = 2 и т.д. При определении степени вершины графа петля учитывается дважды.

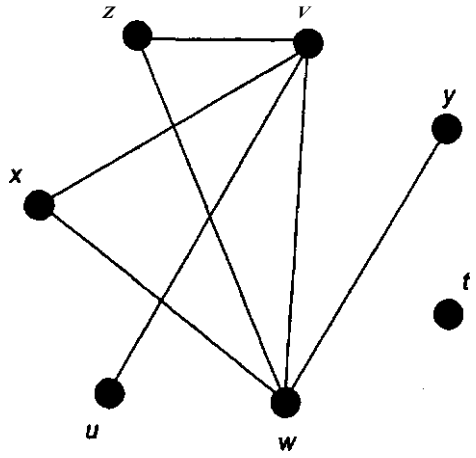


Рис. **8**.9. Исходный граф

На рис. 8.9 вершины *и* и *у* тупиковые, а вершина t — изолированная.

Вершина *v* называется *тупиковой вершиной,* если deg(u) = 1; если deg(u) = 0, то говорят, что *v* — *изолированная вершина.*

Граф называется if-связным, если каждая его вершина связана с *К* другими вершинами; при этом говорят о слабо- и сильносвязных графах (рис. 8.10).

Иногда связность определяют как характеристику не каждой, а одной (произвольной) вершины. Тогда появляются определения типа: граф называется ^-связным, если хотя бы одна его вершина связана с *К* другими вершинами.

В ряде случаев связность определяют как экстремальное значение количественной характеристики. Например, граф является X-связ-

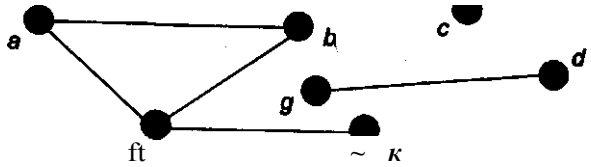


Рис. 8.10. Несвязный фаф

ньтм, если в графе существует хотя бы одна вершина, связанная с *К* смежными вершинами и не существует ни одной вершины, связанной с более чем *К* смежными вершинами (рис. 8.11).

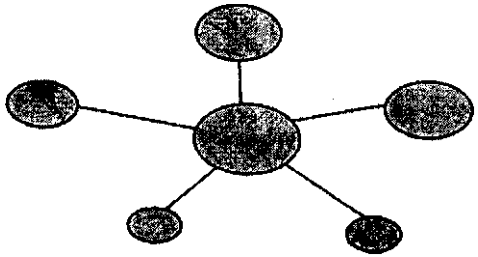


Рис. 8.11. Связность вершин графа

Регулярные графы — графы, у которых степень каждой вершины одинакова (рис. 8.12).

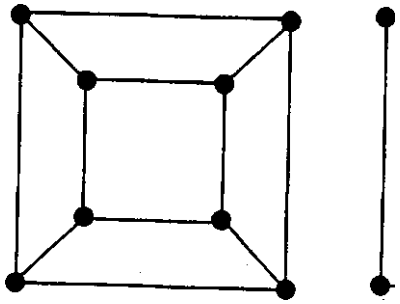


Рис. 8.12. Регулярные графы: ***а*** — степени 3; б— степени 2

Введем в рассмотрение понятие — *инвариант сети.* Инвариантом сети *G* называется параметр, имеющий одно и то же значение

для всех сетей, изоморфных *G.* Среди самых очевидных инвариантов отметим следующие:

- число вершин;
- число ребер;
- число компонент;
- последовательность степеней, т.е. список степеней вершин в убывающем порядке значений.

Например, для двух пятивершинных сетей на рис. 8.13 все четыре из перечисленных выше инвариантов совпадают, но эти сети не изоморфны.

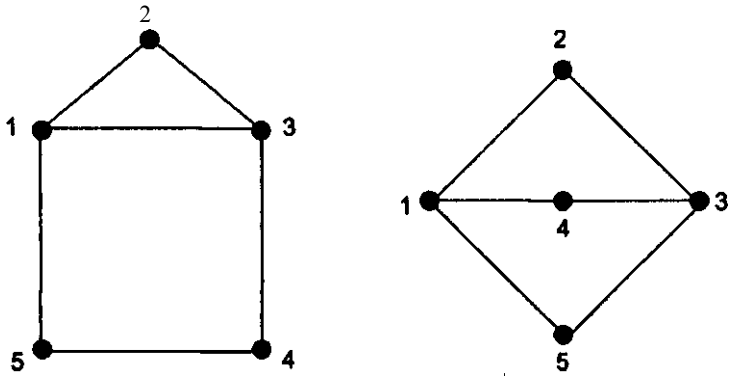


Рис. 8.13. Пример неизоморфных сетей

Двудольный граф. Граф называется двудольным, если множество его вершин можно разбить на два непересекающихся подмножества Vi и *Vi* так, что каждое ребро графа *G* соединяет какую-либо вершину из Vi с какой-либо вершиной из ***V2.*** Такие графы обозначают *G(V\,* ***V2).*** Двудольный граф можно определить в терминах раскраски его вершин двумя цветами (например, синим и красным). При этом фаф называют двудольным, если каждую его вершину можно окрасить красным или синим цветом так, чтобы любое ребро имело один конец красный, другой синий (рис. 8.14).

В двудольном фафе совсем не обязательно, чтобы каждая вершина из Vi соединялась с каждой вершиной из ***V2.***

Плоский граф. Плоским графом называют граф, изображенный на плоскости так, что никакие его два ребра геометрически не пересекаются нигде, кроме инцидентной им обоим вершины. Граф,

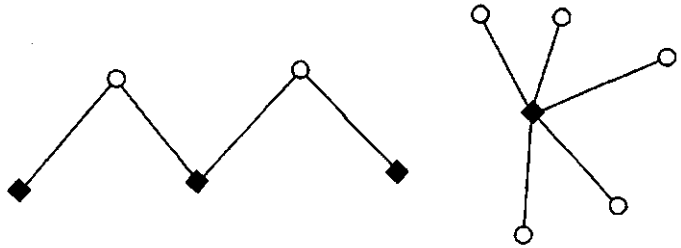


Рис. 8.14. Двудольные графы

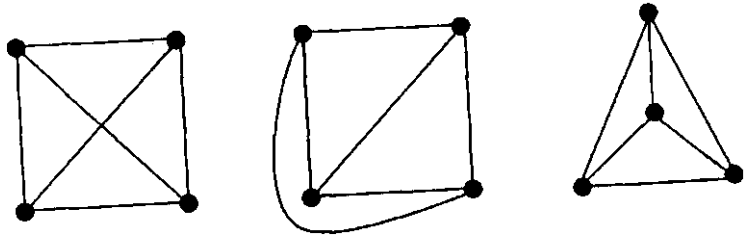
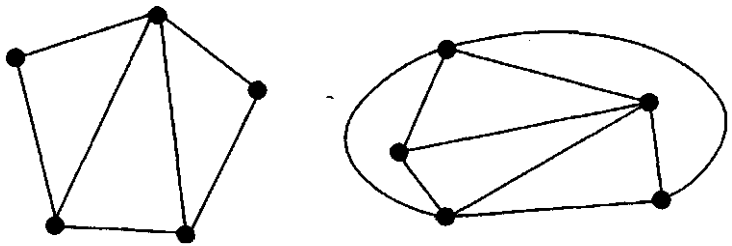


Рис. 8.15. Планарные графы

изоморфный плоскому графу, называют *планарным.* Все три графа (рис. 8.15) планарны, но только второй и третий плоские.

Триангулированный граф. Плоский граф называется максимально плоским, если невозможно добавить к нему ни одного ребра так, чтобы полученный граф был плоским. Каждая *грань,* в плоском представлении максимально плоского графа имеет три вершины. Максимально плоский граф называется еще триангулированным (рис. 8.16). Операция дополнения новых ребер, в результате которой



а б

Рис. 8.16. Плоский («) и триангулированный (б) графы

в плоском представлении каждая грань имеет ровно три вершины, называется *триангуляцией графа.*

Примечание. Существует только один триангулированный граф с четырьмя вершинами и только один с пятью вершинами.

8.3. Понятие подграфа

Сеть $G' = (V',E')$ является подсетью сети $G = (V,E)$, если $V \subset V$ и $E' \subset E$ (рис. 8.17). Если $V = V$, то говорят, что G' — остовная подсеть G. Если v 6 V, то $G - v$ обозначает подсеть, вершины которой есть $V - \{v\}$, а ребра — все ребра в E, не инцидентные v, т.е. $G - v$ получается из G вычеркиванием v и всех инцидентных v ребер.

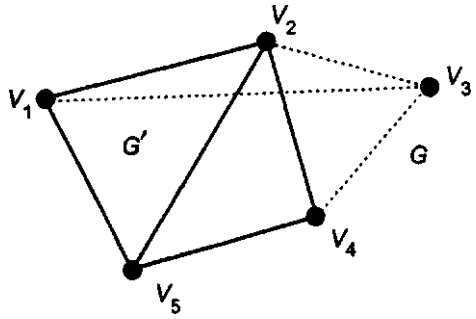


Рис. 8.17. Подсеть G' сети G

8.4. Циклы на графе

Маршрут на графе $G = (V,E)$ из вершины $u\backslash$ в вершину u_n — это конечная последовательность вершин $W = щ, \mathbf{U2}, \bullet\bullet\bullet, u_n$ таких, что $(щ, щ+2) \in E(G)$ для каждого $г$, $1 < г < п - 1$. Маршрут W замкнут, если $ui = u_n$; в противном случае маршрут W открыт. Маршрут называется путем, если ни одна вершина (и, следовательно, ни одно из ребер) не появляется в W более одного раза (рис. 8.18).

На рис. 8.18 *uvxwywzvwvu* есть маршрут; запись *uvxwz* — путь, а *vxwzv* — цикл.

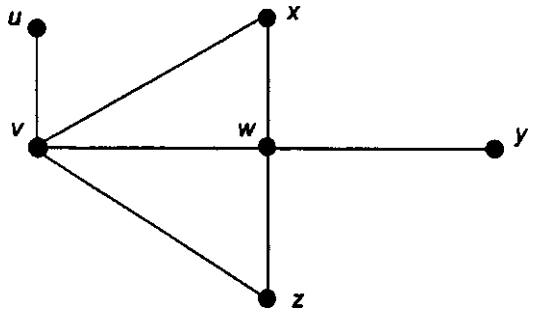


Рис. 8.18. Маршруты и циклы

Цепь — незамкнутый маршрут (путь), в котором все ребра (дуги) попарно различны.

Цикл — замкнутая цепь (в неориентированном графе).

Контур — замкнутый путь (в ориентированном графе).

Простой путь (цепь) — путь (цепь), в котором ни одна дуга (ребро) не встречается дважды.

Простой цикл (контур) — цикл (контур), в котором все вершины попарно различны.

Связный граф без циклов называется *деревом.* Слово «дерево» в теории графов означает граф, в котором нет циклов, т.е. в котором нельзя из некоторой вершины пройти по нескольким различным ребрам и вернуться в ту же вершину. Генеалогическое дерево будет деревом и в смысле теории графов, если в этом семействе не было браков между родственниками. Деревья особенно часто возникают на практике при изображении различных иерархий. На рис. 8.19 показано библейское генеалогическое дерево.

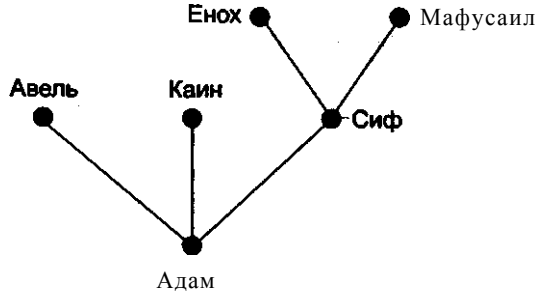


Рис. 8.19. Генеалогическое дерево

Граф является деревом тогда и только тогда, когда каждая пара различных вершин соединяется одной и только одной цепью. Если все вершины графа *G* принадлежат дереву *T,* то считается, что дерево покрывает граф *G,* т.е. имеем покрывающее дерево (рис. 8.20).

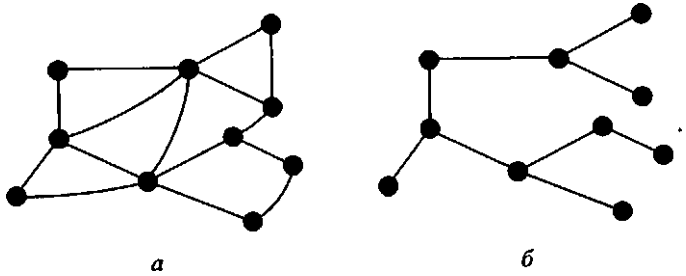


Рис. 8.20. Граф (а) и его покрывающее дерево (б)

Графы являются весьма удобным средством описания и оптимизации алгоритмов вычисления.

В качестве примера рассмотрим вычисление квадратного полинома $ax^2 + bx + c$ по схеме Горнера (рис. 8.21).

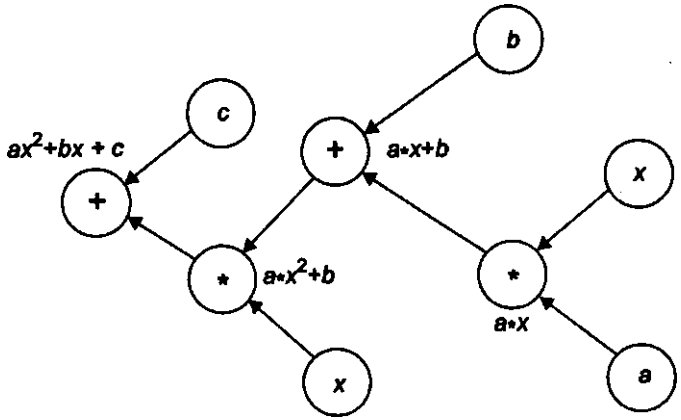


Рис. 8.21. Граф вычисления квадратного полинома по схеме Горнера

Построим граф, изображающий отношение делимости на множестве {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. Принцип такой: если от одного числа до другого есть цепь, ведущая вверх, тогда второе число делится на первое (рис. 8.22).

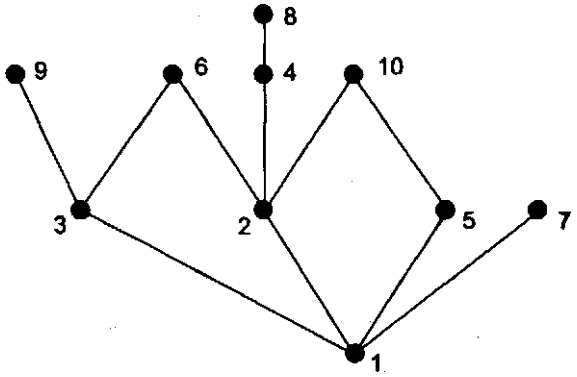
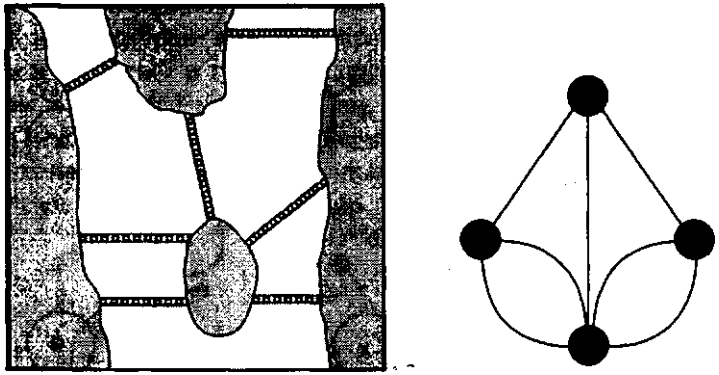


Рис. 8.22. Граф, изображающий отношение делимости на множестве

Эйлеров цикл. Цикл называется эйлеровым, если он проходит через все ребра и при этом только по одному разу. Граф, содержащий эйлеров цикл, называется эйлеровым графом.

Задача 1. Через город Калининград протекает река Прегель, омывающая остров. На реке имеется семь мостов (рис. 8.23). Может ли пешеход обойти все мосты, пройдя по каждому только один раз?



а *б*

Рис. 8.23. Задача о мостах:
а — рисунок мостов; *б* — графовая модель

Решение. Эйлеров цикл в графе существует тогда и только тогда, когда граф связный и все его вершины имеют четные степени. В графе,

имеющем более двух вершин с нечетной степенью, такого обхода не существует.

Задача 2. Начертить рисунок, не отрывая карандаш от бумаги и не проходя линию дважды (рис. 8.24). Цифрами показан один из вариантов построения эйлерова цикла.

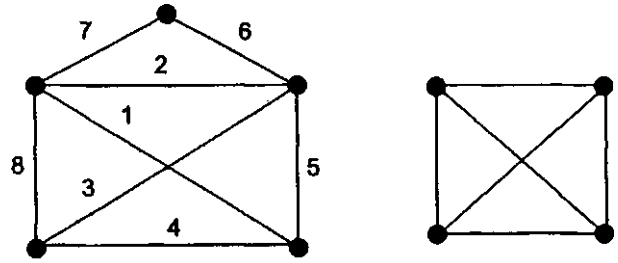


Рис. 8.24. Графы, содержащие циклы:
а — эйлеров цикл; *б* — неэйлеров цикл

Критерий наличия эйлерова цикла в графе: связный граф является эйлеровым тогда и только тогда, когда степени всех его вершин — четные числа.

Гамильтонов цикл. В 1859 г. У. Гамильтон придумал игру «Кругосветное путешествие», состоящую в отыскании такого пути, проходящего через все вершины (города, пункты назначения) графа, изображенного на рис. 8.25, чтобы посетить каждую вершину однократно и возвратиться в исходную. Пути, обладающие таким свойством, называются *гамильтоновыми циклами.*

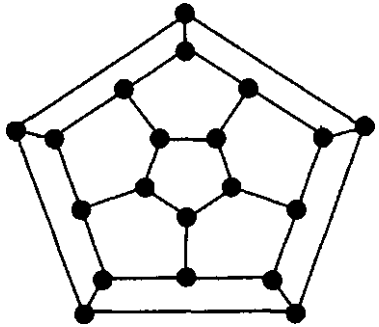


Рис. 8.25. Игра «Кругосветное путешествие»

Например, на графе (рис. 8.26) имеется гамильтонов цикл: 1 - 2 - 3 - 4 - 5 - 6 - 1 .

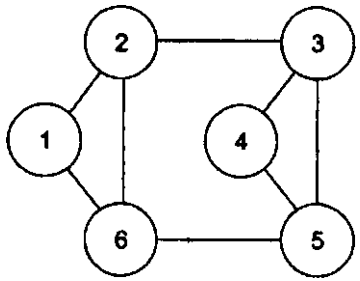


Рис. 8.26. Гамильтонов цикл

8.5. Цикломатическое число графа

Цикломатическим числом графа называется число, равное увеличенной на единицу разности между количеством ребер и количеством вершин графа: $\gamma = n - m + 1$, где n — количество ребер; m — количество вершин.

Цикломатическое число графа показывает, сколько ребер надо удалить из графа, чтобы в нем не осталось ни одного цикла.

Задача 1. Дан граф, у которого $m = 6$, $n = 9$ (рис. 8.27). Определить, сколько ребер на графе нужно удалить, чтобы в нем не осталось ни одного цикла.

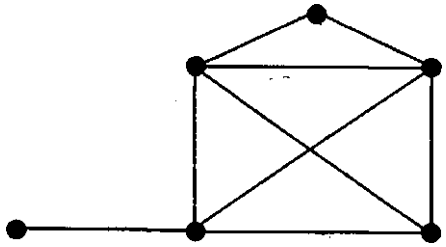


Рис. 8.27. Исходный граф

Для заданного графа цикломатическое число $\gamma = 9 - 6 + 1 = 4$. Это означает, что если на графе удалить 4 ребра, то в нем не останется ни одного цикла. Тогда остовный подграф будет иметь вид, показанный на рис. 8.28.

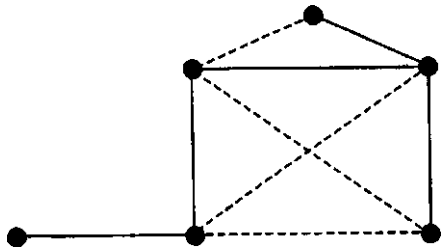


Рис. 8.28. Остовный подграф

Задача 2. Какое минимальное число дверей нужно сделать в замке (рис. 8.29), чтобы попасть во все комнаты (дверь — одно ребро).

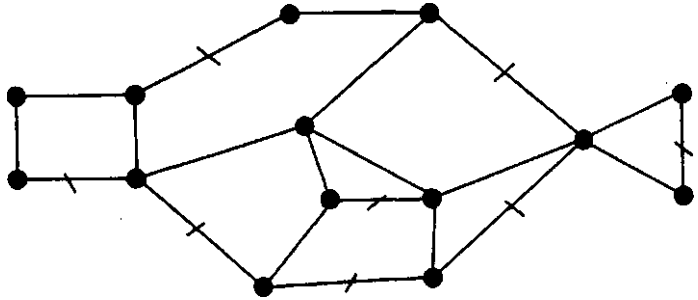


Рис. 8.29. Схема замка (вид сверху)

Решение. Количество вершин $m = 14$, количество ребер $n = 21$. Цикломатическое число $\gamma = 21 - 14 + 1 = 8$. Следовательно, в замке нужно сделать восемь дверей.

Наряду с цикломатическим числом в теории графов вводится понятие ***коцикломатического числа,*** которое равно полному числу ребер в остовном дереве графа.

Дерево — это неориентированная связная сеть без циклов. На рис. 8.30 приведены различные деревья.

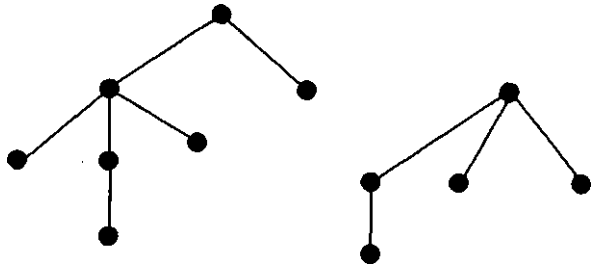


Рис. 8.30. Древовидные структуры

Ориентированная сеть является деревом тогда и только тогда, когда она дерево с дугами, рассматриваемыми как неориентированные ребра. Ациклическая ориентированная сеть не обладает ориентированными циклами. Например, дерево на рис. 8.31 можно описать следующим образом:

$$T = \{1, T_2\}, \quad T_2 = \{2, T_3, T_4\}, \quad T_3 = \{3, T_5, 7/_6, 7/_7\}, \qquad (1)$$

$$7/_5 = \{5\}, \quad 7/_6 = \{6\}, \quad \Gamma_7 = \{7\}, \quad \Gamma_4 = \{4, 7/8\}, \qquad (2)$$

$$7/_8 = \{8, 7/_9, T_{10}\}, \quad T_9 = \{9\}, \quad T_{10} = \{10\}. \qquad (3)$$

В явном виде это описание выглядит так:

$$\Gamma = \{1, \{2, \{3, \{5\}, \{6\}, \{7\}\}, \{4, \{8, \{9\}, \{10\}\}\}\}\}.$$

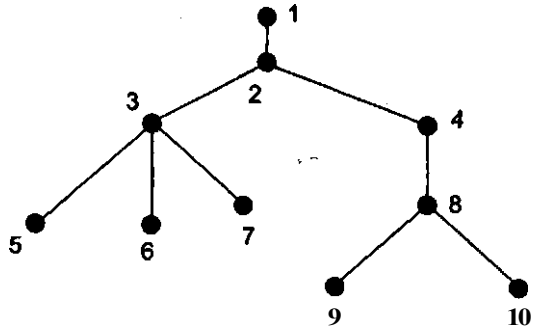


Рис. 8.31. Дерево

8.6. Представление графов в ПЭВМ

Очертанием графа считается любая топологически связанная область, ограниченная ребрами графа. Для многих приложений все узлы графа должны совпадать с узлами технологической сетки. В этом случае ребра графа могут представлять собой кривые линии, дуги или ломаные линии, состоящие из отрезков прямых. Важно помнить, что сеть может быть изображена многими различными способами; например, сети, приведенные на рис. 8.32, — изоморфные.

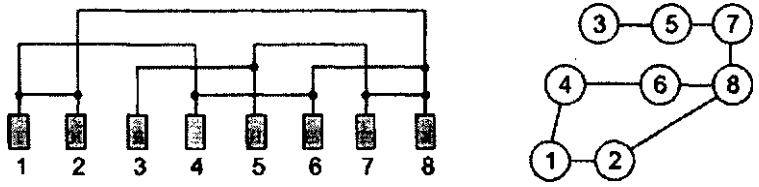


Рис. 8.32. Способы изображения сети

Графы в ПЭВМ можно представить тремя способами: матрицей смежности, матрицей инцидентности и вектором смежности.

Матрица смежности. Любой граф $G = (V,E)$ с M вершинами может быть представлен матрицей размера M x M при условии, что вершинам G приписаны произвольные метки.

Если вершины G обозначены «ь v_2, ***VM,*** ТО элементы а,у матрицы смежности A определяются следующим образом:

$$a_{ij} = \begin{cases} K, & \text{если } Vi \text{ и } Vj \text{ смежны;} \\ 0, & \text{если } Vi \text{ и } VJ \text{ несмежны.} \end{cases}$$

Здесь ***K*** — вес ребра, соединяющего данные вершины. На рис. 8.33 представлены граф и матрица смежности, описывающая его.

Матрица смежности неориентированного графа является симметричной.

Матрица инцидентности. Если вершины графа обозначены $v1$, v_2,..., ***VM,*** а ребра — метками $x1$, x_2,..., ***XN,*** ТО матрица инцидентности определяется следующим образом:

$$a_{ij} = \begin{cases} 1, & \text{если ребро } Xj \text{ инцидентно вершине зд;} \\ 0, & \text{если ребро } Xj \text{ не инцидентно вершине } Vi. \end{cases}$$

$A =$

	1	2		
A	–	2	–	9
2	2	–	–	7
Z	–	–	–	6
A	9	7	6	–

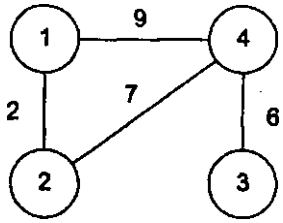


Рис. 8.33. Матрица смежности *(а)* для графа (б)

Часто в матрицах смежности и инцидентности вместо нуля ставят «-» (для наглядности). Единицы в матрицах (рис. 8.34) могут быть заменены целыми числами, характеризующими веса ребер, такая матрица называется *матрицей оценки (весов).*

$B-$

	1	*2	*3	4
"1	1	1	–	1
2	1	–	1	–
Z	–	–	–	1
\	–	1	1	–

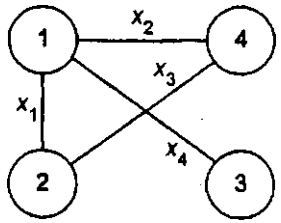
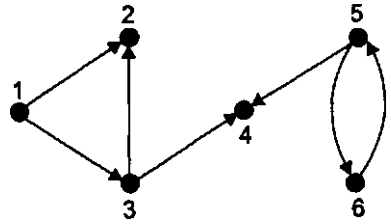


Рис. 8.34. Матрица инцидентности (инциденций) *(а)* для графа (б)

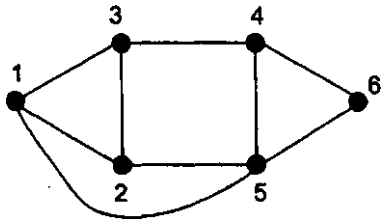
На рис. 8.35 приведены матрицы инциденций для ориентированного и неориентированного графов.

Модель системы часто представляется ориентированным графом $G = (V, E)$ с множеством вершин V и множеством дуг E — упорядоченных пар номеров смежных вершин [г, *j],* E = [ц, ..., [г„, *j,].* Общее количество таких пар обозначим как Q.

Одним из способов представления топологии является матрица изоморфности D, в строках которой представлены номера входящих (с плюсом) и выходящих (с минусом) дуг.



	<1.2>	<2,3>	<3,4>	<5,4>	<5,6>	<6,5>
1	-1	-1	0	0	0	0
2	1	0	1	0	0	0
3	0	1	-1	-1	0	0
4	0	0	0	1	1	0
5	0	0	0	0	-1	-1
6	0	0	0	0	0	1



1	1	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	0	1	0	1	0	1	0	0
4	0	0	0	0	0	1	1	0
5	0	0	1	0	1	0	1	1
6	0	0	0	0	0	0	1	1

б

Рис. 8.35. Графы и их матрицы инциденций:
а) ориентированный граф; б) неориентированный граф

Для графа, приведенного на рис. 8.36, матрицы смежности и изоморфности имеют вид:

$$R = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{vmatrix}, \qquad D = \begin{vmatrix} +6 & -1 \\ +1.+7 & -2 \\ +2 & -3, -7 \\ +3 & -4 \\ +4 & -5 \\ +5 & -6 \end{vmatrix}$$

Избыточность хранимой информации в матрице смежности (нулевые значения) компенсируются простотой вычислительных алгоритмов и скоростью получения требуемой информации из матрицы.

Другой способ представления графа—*вектор смежности,* который выдает списки узлов, с которыми данный узел (вершина) связан непосредственно. Для графа, изображенного на рис. 8.37, такое описание можно представить в виде структуры (таблицы). В колонке S

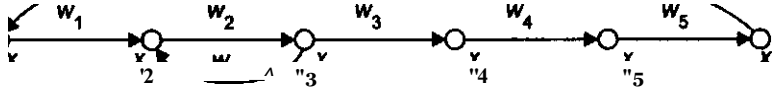
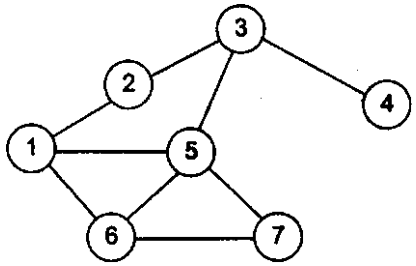


Рис. 8.36. Модель системы в форме графа



а

S	Список смежных узлов			
1	2	5	6	
2	1	3		
3	2	**4**	5	
4	3			
5	1	3	6	7
6	1	5	7	
7	5	6		

б

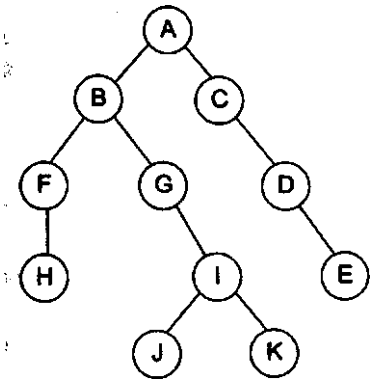
Рис. 8.37. Граф ***(а),*** представленный вектором смежности **(б)**

представлены номера узлов, далее в строке таблицы следует список соседних узлов. По этой причине число колонок в каждой из строк различно.

Представление древовидных структур. При представлении древовидных структур в машинной памяти в виде связного списка каждый элемент списка может содержать три поля, в одном из которых хранятся данные, соответствующие элементу (базовый тип), а в двух оставшихся полях — указатели левого и правого потомков узла (рис. 8.38).

Контрольные вопросы

1. Дайте определение графа.
2. Что такое степень вершины графа?
3. Чем характеризуется изоморфизм?
4. Какой граф называется регулярным?
5. Что такое инвариант сети?
6. Какой граф называется двудольным?
7. Дайте определение подграфа.



Номер строки	Элемент	Левый потомок	Правый потомок
1	*A*	2	3
2	*B*	6	7
3	C	0	4
4	*D*	0	5
5	*E*	0	0
6	*F*	8	0
7	*G*	0	9
8	*H*	0	0
9	*I*	10	11
10	*J*	0	0
11	*K*	0	0

а *о*

Рис. 8.38. Представление древовидной структуры:
а — дерево; **б**—табличное представление

8. Что показывает цикломатическое число графа?
9. Чем маршрут отличается от цикла?
10. В чем различие эйлерова и гамильтонова циклов?
11. Опишите способы представления графов в ПЭВМ.
12. Укажите способы представления древовидных структур.

Глава 9
Алгоритмы построения остовного (покрывающего) дерева сети

Древовидная структура характеризуется следующими свойствами.

1) существует единственный элемент, или узел, на который не ссылается никакой другой элемент и который называется корнем;

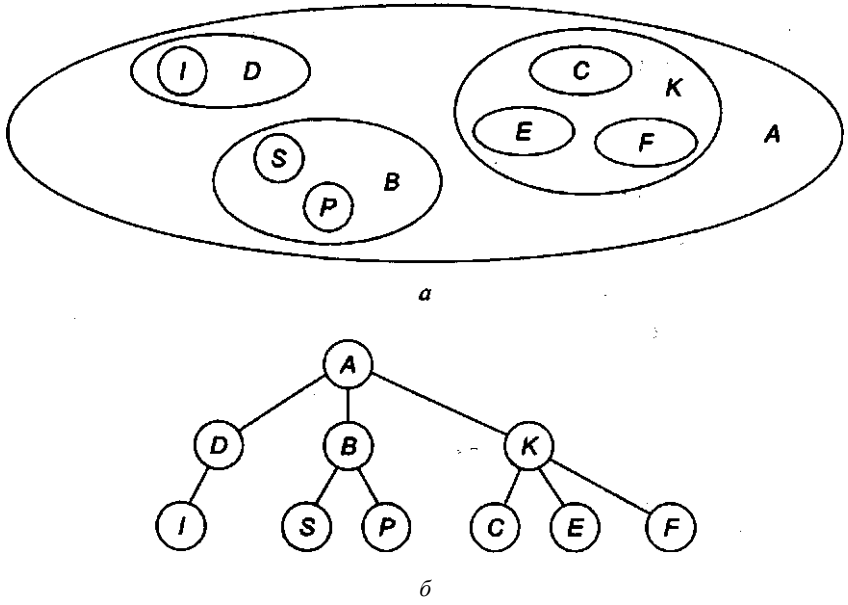
2) начиная с корня и следуя по определенной цепочке указателей, содержащихся в элементах, можно осуществить доступ к любому элементу структуры;

3) на каждый элемент, кроме корня, имеется единственная ссылка, т.е. каждый элемент адресуется единственным указателем.

Узел *y,* который находится непосредственно под узлом *x,* называется непосредственным *потомком x;* если *x* находится на уровне г, то говорят, что *y* расположен на уровне г +1. Наоборот, узел *x* называется непосредственным предком *y.* Считается, что корень дерева расположен на уровне 1. Максимальный уровень какого-либо элемента дерева называется его ***глубиной*** или ***высотой.***

Число непосредственных потомков внутреннего узла называется его ***степенью.*** Максимальная степень всех узлов есть степень дерева. Число ветвей, или ребер, которые нужно пройти, чтобы продвинуться от корня к некоторому узлу *x,* называется длиной пути к *x.* Корень имеет длину пути 1, его непосредственные потомки — длину пути 2 и т.д. Вообще, узел на уровне *г* имеет длину пути г. Длина пути дерева определяется как сумма длин путей всех его узлов. Она также называется длиной ***внутреннего пути.***

Если элемент не имеет потомков, он называется ***терминальным элементом,*** или ***листом,*** а элемент, не являющийся терминальным, называется ***внутренним узлом*** (рис. 9.1).



а

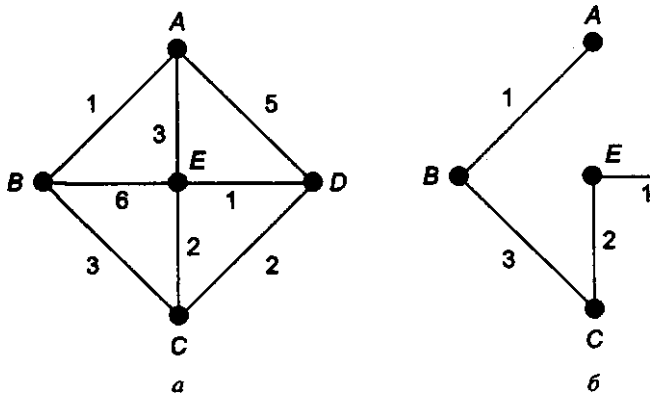
б

Рис. 9.1. Представления древовидной структуры:
а **— вложенное множество;** *б* **— дерево**

Предположим, что необходимо принять решение, связанное с организацией сети компьютеров в различных территориальных пунктах. ***ЭТО*** решение является довольно сложным и зависит от большого количества факторов, которые включают вычислительные ресурсы, доступные в каждом пункте, соответствующие уровни потребностей, пиковые нагрузки на систему, возможное неэффективное использование основного ресурса в системе и, кроме того, стоимость предлагаемой сети. В эту стоимость входят: приобретение оборудования; прокладка линий связи; обслуживание системы и т.д. Необходимо определить стоимость такой сети.

Нетрудно видеть, что сформулированная здесь задача имеет много других аналогов. Например, требуется соединить несколько населенных пунктов линиями телефонной связи таким образом, чтобы все эти пункты были связаны в сеть и чтобы стоимость прокладки коммуникаций была минимальной. Вместо телефонных линий можно говорить о прокладке водопроводных коммуникаций, о строительстве дорог и т.д. Решение подобных задач возможно с использованием теории графов (сетей).

Пусть G — (V,E) — связный неориентированный граф, содержащий циклы, т.е. замкнутые маршруты, где V — множество вершин, а E — множество ребер. Остовным (покрывающим) деревом называется подграф, не содержащий циклов, включающий все вершины исходного графа, для которого сумма весов ребер минимальна (рис. 9.2).



а *б*

Рис. 9.2. Граф (в) и его остовное дерево *(б)*

Цикломатическое число 7 показывает, сколько ребер на графе нужно удалить, чтобы в нем не осталось ни одного цикла: 7 = = ***n — m*** + 1, где ***n*** — количество ребер; ***m*** — количество вершин. Например, для графа, изображенного на рис. 9.2, цикломатическое число равно 7 = n- m + l = 8- 5 + l = 4. Это значит, что если на графе убрать четыре ребра, то в нем не останется ни одного цикла, а суммарный вес ребер будет равен 7.

Для построения остовного дерева графа используются алгоритмы Крускала и Прима.

9.1. Метод Крускала

На рис. 9.3 показана схема алгоритма Крускала.

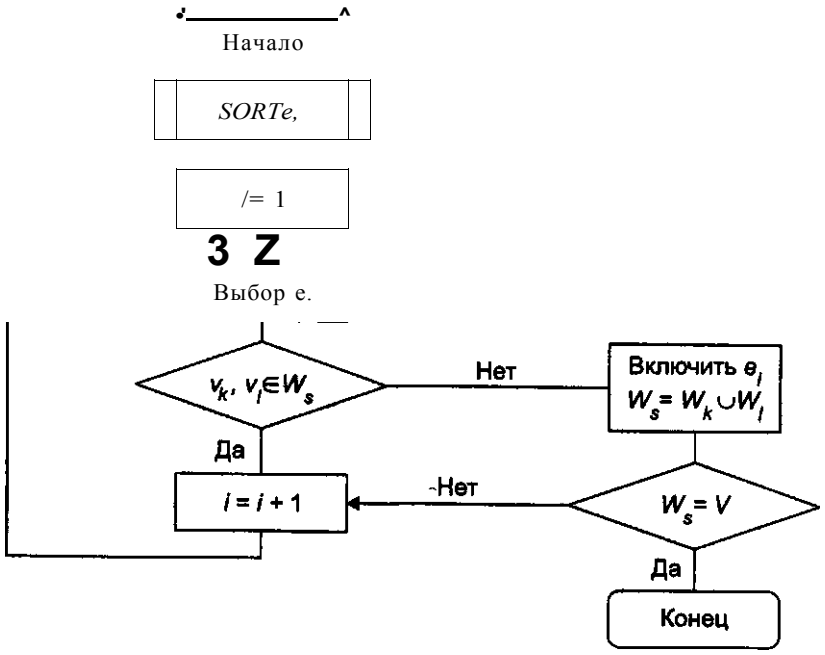


Рис. 9.3. Схема алгоритма Крускала

Блок ***SORT ei*** предполагает предварительную сортировку весов ребер в порядке их возрастания. В начале работы алгоритма предполагается, что в искомом остове не проведено ни одно ребро (т.е. остов состоит из изолированного множества вершин ***v|, v„ ... , v„,*** где ***m*** — количество вершин графа). Поэтому считается, что множество ***W,*** имеет вид: ***W,*** = {{vi}, **{112},** ..., {^n}}» где ***{vi}*** обозначает множество, состоящее из единственной изолированной вершины Vj. Проверка ***Vk,vi € W,*** предполагает установление факта: входят ли вершины ***Vk, vi*** во множество ***W,*** как изолированные или они сами входят в подмножества постепенно увеличивающихся связных вершин Wit, ***Wi,*** каждое из которых имеет вид: ***Wk*** = {•••, ***Vk,*** •••} и Wj = {..., «ь ...}.

Если обе вершины ***v^, vi*** содержатся в одном из подмножеств ***Wk, Wi,*** то ребро ***(к, I) в*** остов не включается, в противном случае данное ребро включается в остов, а множества ***Wk, Wi*** объединяются. Работа алгоритма Крускала заканчивается, когда множество ***W,*** совпадет по мощности с множеством ***V*** — множеством всех вершин графа. Нетрудно видеть, что это произойдет, когда все вершины графа окажутся связными.

На рис. 9.4 показана работа алгоритма Крускала. Ребра рассматриваются в порядке возрастания весов (текущее ребро показано стрелкой). Ребро включается в остовное дерево, если оно не создает цикла и связывает вершины, принадлежащие разным множествам.

Пример 1. Построить остовное дерево графа, заданного матрицей смежности

G	**1**	**2**	**3**	4	**5**
1	0	**50**	**0**	**25**	**10**
2	50	0	**25**	**0**	**30**
3	0	25	0	**50**	**35**
4	25	0	50	0	**15**
5	10	30	35	15	0

Для построения остовного связного дерева минимального веса используется алгоритм Крускала.

Шаг 1. Первоначально каждая вершина исходного графа помещается в одноэлементное подмножество, где все вершины изолированы.

Шаг 2. Ребра сортируются по возрастанию веса.

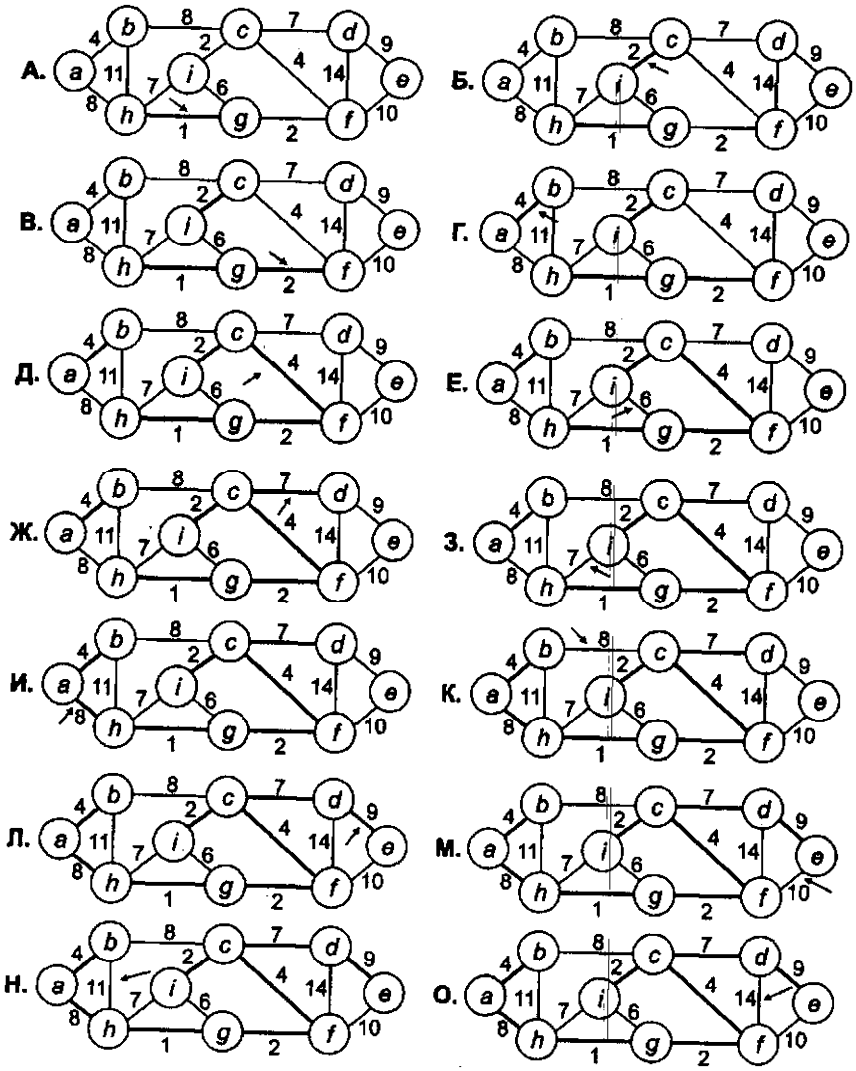


Рис. 9.4. Этапы построения остовного дерева алгоритмом Крускала

Ш а г 3. Ребро включается в остовное дерево, 1сли оно связывает вершины, принадлежащие разным множествам.

Ш а г 4. Алгоритм заканчивает работу, когда все вершины объединяются в одно множество, при этом оставшиеся ребра не включаются в остовное дерево.

Рассмотрим реализацию этого алгоритма на примере построения остовного дерева минимального веса для графа *G* (табл. 9.1).

Таблица 9.1

Этапы построения остовного дерева методом Крускала

Ха п/п	Выполняемые действия	Множество вершин	Граф
1	Построим остовной подграф, содержащий только изолированные вершины	Каждая вершина исходного графа помещается в одноэлементные подмножества: *{Ъ},{Ъ},{Уз},{У*},Ш*	•
2	Найдем ребро минимального веса (в данном случае *Ris)* и добавим его в остовной подграф	Образуется связное подмножество вершин: {Vi, *Vt}*. Одноэлементные подмножества вершин: **{V2}, {V3},** *{V*}*	*V*з. #'4
3	Среди оставшихся ребер найдем ребро минимального веса (в данном случае Д45) и добавим его в остовной подграф	Образуется связное подмножество вершин: {*V\,* ***V5, V4 }.*** Одноэлементные подмножества вершин: ***{V2},*** *{Уз}*	^ 4 5
4	Среди оставшихся ребер найдем ребро минимального веса (в данном случае ***R23)*** и добавим его в остовной подграф	Создается второе связное подмножество: **{V2, V3}.** Существует также первое связное подмножество: {Vi, $V_{в)}$ V_4}	v2< *23 v £ * « F'Zi V .
5	Среди оставшихся ребер найдем ребро минимального веса (в данном случае ***R25)*** и добавим его в остовной подграф	Так как одна вершина ребра входит в одно связное подмножество, а другая вершина в другое связное подмножество, эти подмножества объединяются в единое связное множество: {Vi, $V_в$, V_{it} $V_{2>}$ $V_з$}	B23 V \ 4 5
6	Получен граф, который является: • *остовным,* так как включает все вершины; • *связным,* так как все вершины в нем можно соединить маршрутами; • *деревом,* так как в нем отсутствуют циклы; • *графом минимального веса,* так как в него последовательно включались ребра, отсортированные по возрастанию.		
7	Полученное остовное связное дерево обладает минимальным весом: *R\s* + + **Я45** + ***R23*** + **Я25** = 10 +15 + 25 + 30 = 80.		

Пример 2. Пусть дана схема микрорайона. Необходимо соединить дома телефонным кабелем таким образом, чтобы его длина была минимальной. Схему микрорайона представим взвешенным графом (рис. 9.5).

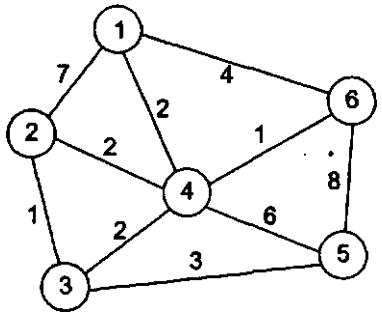


Рис. 9.5. Графическая схема микрорайона

Определим цикломатическое число графа 7 = n — m + 1 = 10 - — 6 + 1 = 5, т.е. на графе необходимо удалить пять ребер.

Первоначально каждая вершина исходного графа помещается в одноэлементное подмножество, считаем, что все вершины изолированы, т.е. не связаны (табл. 9.2).

Таблица 9.2

Реализация метода Крускала

Ребро	Вее	Множества вершин	Операция
		[Vi}, Ш, {V₃}, {V₄}, {V₅}, {V₆}	
(Vi, V_3)	1	{V_2, V_3}, {V}, {Vi}, {Vs}, {Vb}	Включение
(V_{4}, Vt)	1	{$V^{2)}$ V_3}, {Vi, V_6}, {Vi}, {V_6}	Включение
(VuV)	2	{V^{2}, V_3}, {V_U Vi, V_6}, {Vs}	Включение
(Vs, V_4)	2	{V_u V_2, Vs, Vi, V_6}, {Vb}	Включение
($V_{2)}$ V)	2		Исключение
(Vs, V)	3	{V_u V_a, V_3, V_4, V_5, V_6}	Включение

Ребро включается в остовное дерево, если оно связывает вершины, принадлежащие разным подмножествам, при этом вершины объединяются в новое подмножество.

В таблицу последовательно включаются ребра в порядке возрастания их весов. Ребро (Уг, Уз) связывает две вершины, принадлежащие

разным подмножествам {УЬ} и {Уз}. Поэтому ребро включается в остовное дерево, а вершины объединяются в одно подмножество {V_2, Уз}-

Ребро *(Vi, Ve)* также связывает вершины из разных подмножеств, оно включается в остовное дерево, а вершины образуют подмножество {V_4, V_6>.

Вершины Уг и *V* находятся в одном подмножестве, поэтому ребро **(^2,V4)** исключается из рассмотрения.

Алгоритм заканчивает работу, когда все вершины объединяются в одно множество, при этом оставшиеся ребра не включаются в Ьстовное дерево.

Последовательно просматривая таблицу, получим схему соединения телефонным кабелем домов в микрорайоне (рис. 9.6).

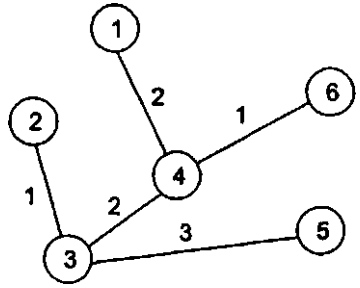


Рис. 9.6. Схема прокладки телефонного кабеля

Пример 3. Пусть дана схема компьютерной сети (рис. 9.7).

Необходимо соединить компьютеры таким образом, чтобы длина проводки была минимальной.

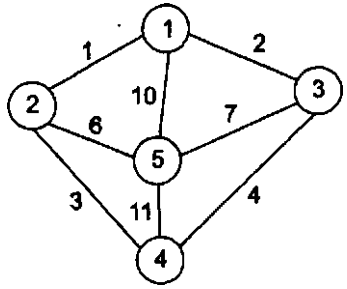


Рис. 9.7. Схема компьютерной сети

Определим цикломатическое число графа:

$$7 = n - m + 1 = 8 - 5 + 1 = 4.$$

Это значит, что на графе требуется удалить четыре ребра.

Решение показано в табл. 9.3.

Таблица 9.3

Реализация метода Крускала

Ребро	Вее	Множества вершин	Операция
		(Vi},{V_2},{V_3},{V_4},{V_5}	
(Vi.Vi)	1	{Vi, V_2}, {Vs}, {V_4}, {V_5}	Включение
(Vi.Vi)	2	{V_{1i} Vs, V_3}, {V_4}, {V_5}	Включение
(Vs, Vi)	3	{Vi, V_2, V_3, V_4}, {V_5}	Включение
(Vs, V_4)	4		Исключение
($V_{2>}$ Vs)	6	{V_{ii} Vs, V_3, V_4, V_5}	Включение

Оставшиеся четыре ребра исключаются из рассмотрения. В итоге получим остовное дерево минимального веса (рис. 9.8).

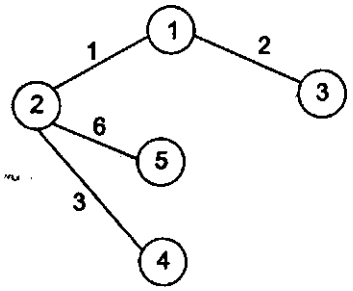


Рис. 9.8. Схема соединения компьютеров

Проведенное исследование метода Крускала показало эффективность его использования для построения остовного дерева минимального веса взвешенного неориентированного графа (сети). Данный метод позволяет обрабатывать сильно связные графы с большим числом вершин, что необходимо в практике разработки и создания эффективных алгоритмов решения на ПЭВМ широкого класса задач.

9.2. Метод Прима

В методе Прима от исходного графа переходим к его представлению в виде матрицы смежности. На графе выбирается произвольная вершина. Выбранная вершина образует первоначальный фрагмент остовного дерева. Затем анализируются веса ребер от выбранной вершины до оставшихся нёвыбранных вершин. Выбирается минимальное ребро, которое указывает на следующую выбранную вершину, и т.д. Процесс продолжается до тех пор, пока в остовное дерево не будут включены все вершины исходного графа.

Этапы работы алгоритма Прима показаны на рис. 9.9.

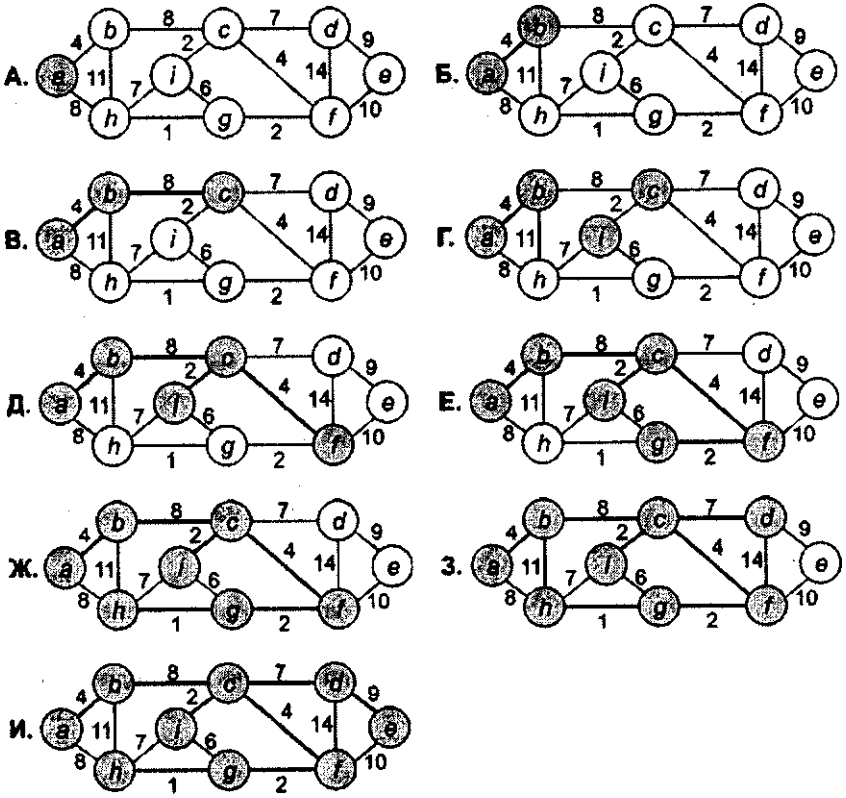


Рис. 9.9. Этапы построения остовного дерева алгоритмом Прима

Пример 1. Пусть дана схема компьютерной сети (рис. 9.10). Необходимо соединить компьютеры таким образом, чтобы длина проводки была минимальной.

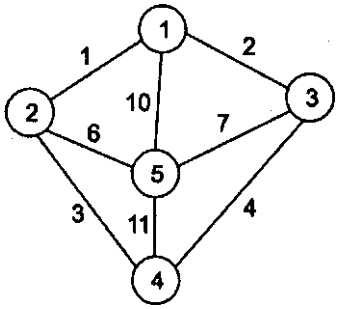


Рис. 9.10. Схема компьютерной сети

Определим цикломатическое число графа: 7 = n — тп + 1 = 8 — — 5 + 1 = 4. Решение показано в табл. 9.4.

Таблица **9.4**

Реализация метода Прима

Выбранные вершины	Невыбранные вершины			
	Vi		$\mathbf{v}_3$	$\mathbf{v}_4$
Vs	10	**(б)**	7	11
$\mathbf{v}_2$	**(1)**	*	7	3
Vi			**(2)**	3
$\mathbf{v}_3$		·		**(3)**

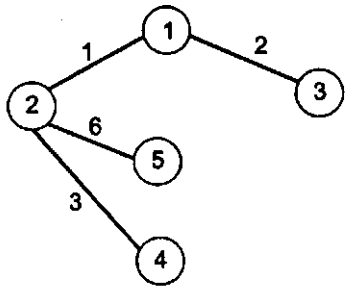


Рис. 9.11. Схема прокладки кабеля

При просмотре строки табл. 9.4 находим минимальную величину веса ребра, выделяем ее, и больше этот столбец, в котором находится эта величина, в рассмотрении-не участвует. Для построения остовного дерева необходимо просмотреть столбцы таблицы снизу вверх и зафиксировать первое появление минимальной величины. В итоге получим остовное дерево минимального веса (рис. 9.11).

Пример **2.** Построить остовное дерево минимального веса для графа, показанного на рис. 9.12.

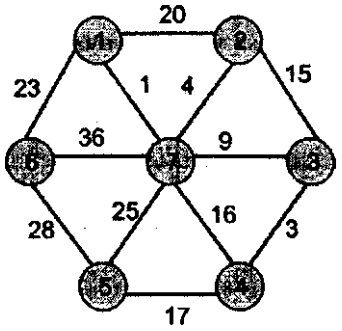


Рис. 9.12. Исходный граф

Таблица 9.5

Реализация метода Прима

Выбранные вершины	v_2	$\mathbf{v}_3$	$\mathbf{v}_4$	V_5	$\mathbf{v}_6$	$\mathbf{v}_7$
Vi	20	_	_	_	23	(1)
v_7	**(4)**	**9**	16	25	23	*
v_2	*	*(9)*	16	25	23	*
$\mathbf{v}_3$	*	*	(3)	25	23	*
$\mathbf{v}_4$	*	*	*	**(17)**	23	·
V_5	*	·	*	*	(23)	*

В табл. 9.5 в скобках отмечены выбранные минимальные элементы. Затем в каждом столбце фиксируются первые появления минимальных элементов (отработка назад), при этом в искомое остовное дерево минимального веса войдут ребра (2,7), (3,7), (4,3), (5,4), (6,1) и (7,1). Порядок просмотра столбцов рабочей таблицы безразличен; важно, чтобы ни один столбец не был пропущен. Полученное в результате выполнения алгоритма остовное дерево изображено на рис. 9.13.

В остовное дерево должны войти все вершины исходного графа, отсюда следует, что в качестве начальной вершины (т.е. первой выбранной строки) может быть выбрана любая вершина графа. В табл. 9.5 в качестве начальной взята вершина 1.

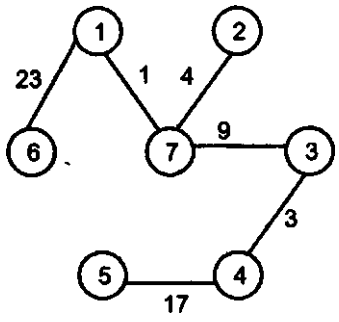


Рис. 9.13. Остовное дерево минимального веса

Пример 3. Построить остовное дерево графа (рис. 9.14) методом Прима.

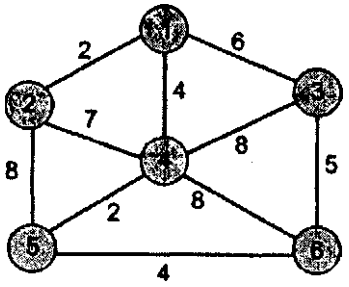


Рис. 9.14. Исходный граф

Таблица 9.6

Реализация метода Прима

Выбранная вершина	Невыбранные вершины				
	Vi	v_2	v_3	$V_в$	v_e
v_4	4	7	8	(2)	8
v_5	4	7	8	•	(4)
Ve	(4)	7	5	*	
Vi	\	(2)	5	*	" «
v_2		*	(5)		*

В табл. 9.6 показаны шаги выполнения метода Прима. Просмотр столбцов табл. 9.6 снизу вверх позволяет определить ребра, включенные в остовное дерево мининального веса (рис. 9.15).

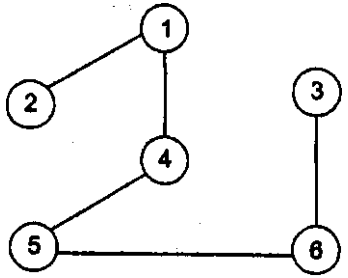


Рис. 9.15. Остовное дерево графа

Контрольные вопросы

1. Что понимается под остовным деревом?
2. Как определяется длина внутреннего пути?
3. Что такое терминальный элемент?
4. Укажите формулу для вычисления цикломатического числа.
5. Каковы особенности методов Крускала и Прима?
6. Каковы критерии завершения работы методов Крускала и Прима?
7. В чем состоит методика анализа сложности алгоритмов построения остовного дерева графа?

Глава 10
Алгоритмы нахождения на графах кратчайших путей

Задача отыскания на графе кратчайшего пути имеет многочисленные практические приложения. С решением подобной задачи приходится встречаться в технике связи (например, при телефонизации населенных пунктов), на транспорте (при выборе оптимальных маршрутов доставки грузов), в микроэлектронике (при проектировании топологии микросхем) и т.д.

Путь между вершинами ***г*** и ***j*** графа считается кратчайшим, если вершины г и j соединены минимальным числом ребер (случай не взвешенного графа) или если сумма весов ребер, соединяющих вершины ***i*** и ***j,*** минимальна (для взвешенного графа).

Важным показателем алгоритма является его эффективность. Применительно к поставленной задаче эффективность алгоритма может зависеть в основном от двух параметров графа: числа его вершин и числа весов его ребер. Рассмотрим три алгоритма для определения кратчайшего расстояния между вершинами графа: построение дерева решений, метод динамического программирования и метод Дейкстры.

10.1. Построение дерева решений

Метод построения дерева решений обычно используется для нахождения кратчайшего пути в ориентированном графе, при этом от исходного графа переходят к матрице смежности. Затем просматривают строки матрицы смежности, и граф последовательно «расслаивается» в дерево решений.

Пример 1. Найти кратчайший путь от вершины 1 до вершины 6 в графе, изображенном на рис. 10.1.

В матрице смежности весов данного графа символ «прочерк» обозначает отсутствие связности между соответствующими вершинами.

Рис. 10.1. Граф и соответствующая ему матрица смежности

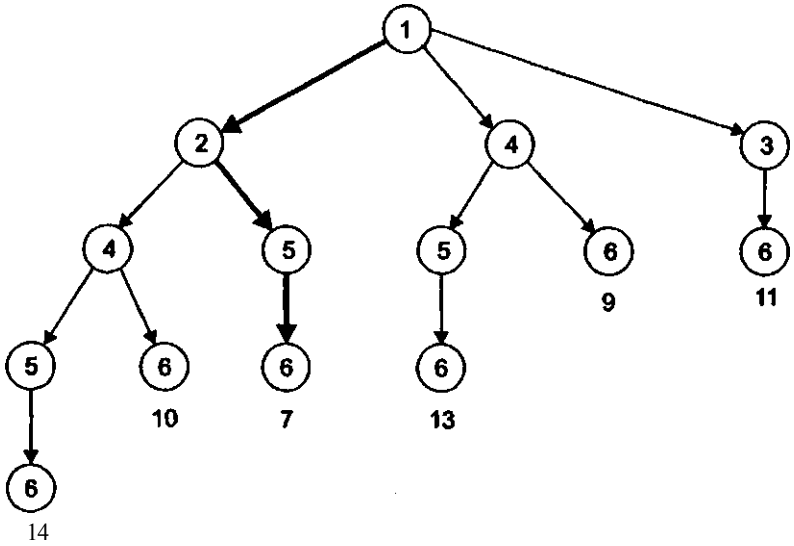


Рис. 10.2. Дерево перебора решений

Полное дерево перебора, построенное по данному алгоритму, изображено на рис. 10.2.

Из рис. 10.2 нетрудно видеть, что кратчайший путь из вершины 1 в вершину 6 равен 7 и имеет вид (1 - 2 - 5 - 6) .

Пример 2. Найти кратчайший путь от вершины 1 до вершины 6 в ориентированном графе, изображенном на рис. 10.3.

Рис. 10.3. Граф и соответствующая ему матрица смежности

Последовательно просматриваем строки матрицы смежности и строим дерево решений. Полное дерево перебора изображено на рис. 10.4.

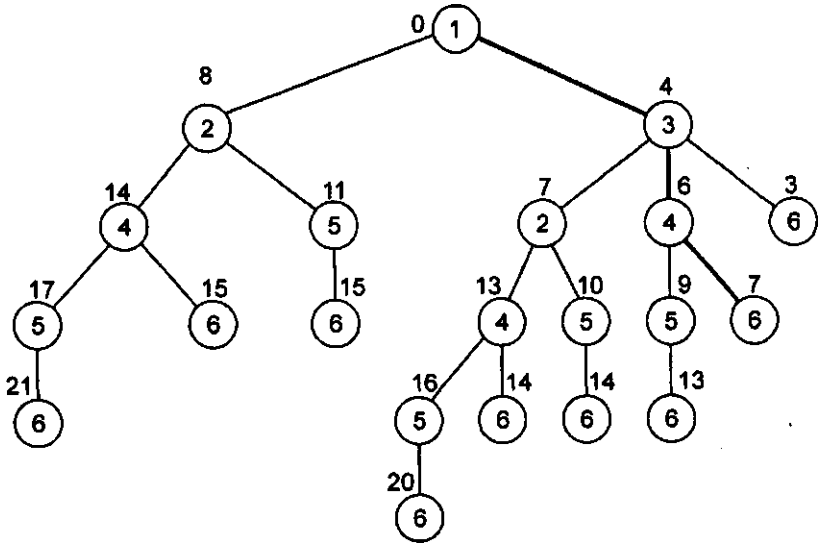


Рис. 10.4. Дерево перебора решений

Дерево решений показывает все возможные пути. Имеется девять вариантов путей от первой до шестой вершины. Кратчайший путь равен *V1 — V3 — V4 — Ve.*

10.2. Метод динамического программирования

Метод динамического программирования основан на пошаговой оптимизации целевой функции, где в качестве целевой функции может быть стоимость, ресурсные затраты, финансово-экономические затраты, а также кратчайшие пути.

Рассмотрим пример топологической оптимизации.

Пример 1. Пусть дана карта местности (рис. 10.5). Разбиваем карту местности координатной сеткой. Шаг сетки выбирается исходя из условия точности. Узлы координатной сетки считаем вершинами графа.

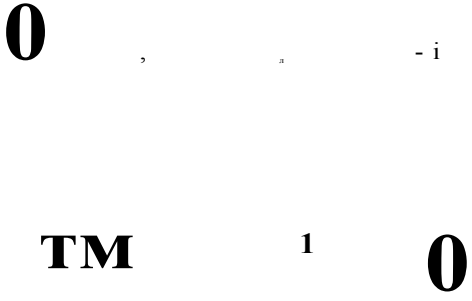


Рис. 10.5. Карта местности

Проставляем веса ребер исходя из того, какая цель преследуется. Построение пути осуществляется из конечной вершины в начальную исходя из следующих приоритетных направлений: вниз и вправо (при движении к конечной вершине).

На первом этапе движения до конечной вершины *B* вынужденные, т.е. вниз и вправо (рис. 10.6, а). На последующих шагах направление движения определяется оптимизацией длин путей (рис. 10.6, б).

В результате выполнения операций определяется длина пути между вершинами (22 единицы) и траектория движения (по стрелкам) — рис. 10.6, е.

Метод динамического программирования рассматривает многостадийные процессы принятия решения. При постановке задачи

а *б* *в*

Рис. 10.6. Выбор кратчайшего пути:
а — первый шаг; *б* — второй шаг; *в* — траектория пути

динамического программирования формируется некоторый критерий. Процесс разбивается на стадии (шаги), в которых принимаются решения, приводящие к достижению общей поставленной цели. Таким образом, метод динамического программирования—метод пошаговой оптимизации.

Введем функцию *fu* определяющую минимальную длину пути из начальной в вершину г. Обозначим через *Sij* длину пути между вершинами г и *j,* а *fj* — наименьшую длину пути между вершиной *j* и начальной вершиной.

Выбирая в качестве *i* такую вершину, которая минимизирует сумму *Sij* + *fj,* получаем уравнение

$$/\text{i} = mm(Sij + fj) \quad \text{либо} \quad U = mm(Sji + fj).$$

Трудность решения данного уравнения заключается в том, что неизвестная функция входит в обе части равенства. В такой ситуации приходится прибегать к классическому методу последовательных приближений (итераций), используя рекуррентную формулу:

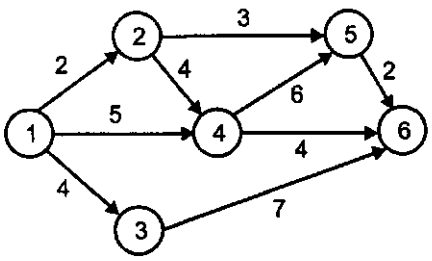
$$f?^{+1)} = mm(Si_{j} \quad + \quad f(^{k}\backslash$$

где $ff^{k>>}$ — fc-е приближение функции.

Возможен другой подход к решению поставленной задачи с помощью метода стратегий. При движении из начальной точки *г* в конечную jfc получается приближение $/\}^{0)} = S_{ik}$, где S_{ik} — длина пути между точками *ink.*

Следующее приближение — поиск решения в классе двухзвенных ломаных. Дальнейшие приближения ищутся в классе трехзвенных, четырехзвенных и других ломаных.

Пример 2. Найти, используя метод динамического программирования, кратчайший путь между вершинами 1 и 6 в графе на рис. 10.7.



	V_1	V_2	V_3	V_4	V_5	V_6
V_1	–	2	4	5	–	–
V_2	–	–	–	4	3	–
V_3	–	–	–	–	–	7
V_4	–	–	–	–	6	4
V_5	–	–	–	–	–	2
V_6	–	–	–	–	–	–

Рис. 10.7. Граф и соответствующая ему матрица смежности

Первый этап алгоритма — поиск длины минимального пути. Для вершины 1 имеем Д = 0, так как для этой вершины никакой путь еще не пройден. Имеем *fi* = min(5i2 + Л) = 2 + 0 = 2. Аналогично имеем /з = min(5i3 + Д) = 4 + 0 = 4. Из рис. 10.7 видно, что из вершины 1 в вершину 4 возможны два пути: (1 - 4) и (1 - 2 - 4), поэтому функция /4 будет иметь вид

$$/4 = \min\{5i4 + /i,\ S24 + /2\} = \min\{5 + 0,\ 4 + 2\} = \min\{5,\ 6\},$$

откуда выбирается минимальное значение функции /4 = 5. Аналогичным способом определяется значение функций /5 и /в:

$$/_5 = \min\{5_25 + /2,\ 5_{45} + /4\} = \min\{3 + 2,\ 6 + 5\} = \min\{5,\ 11\};$$

$$/_6 = \min\{Sb6 + /5,\ S46 + /4,\ 5зб + /3\} = \min\{2 + 5,\ 4 + 5,\ 7 + 4\} = \min\{7,\ 9,\ 11\}.$$

Из последних соотношений находим,, что имеют место следующие минимальные значения: /5 = 5, /б = 7. Таким образом, в результате выполнения первого этапа алгоритма найдено, что кратчайший путь из вершины 1 в вершину 6 равен 7.

На втором этапе алгоритма будет найдена последовательность вершин, через которые проходит вычисленный минимальный путь. Для этого необходимо найти последовательность тех функций, которым

соответствовал выбираемый минимум. Для функции /б минимум вычислялся через значение функции /5. В свою очередь, минимум функции /5 был вычислен через минимальное значение функции /2, а функции /2 — через минимальное значение функции *f*. Следовательно, «отработка назад» от конечной вершины 6 искомого пути до его начальной вершины 1 позволяет указать последовательность проходимых при этом вершин графа— (1 - 2 - 5 - 6).

Пример 3. Определить кратчайший путь из вершины 1 в вершину 10 для графа, представленного на рис. 10.8.

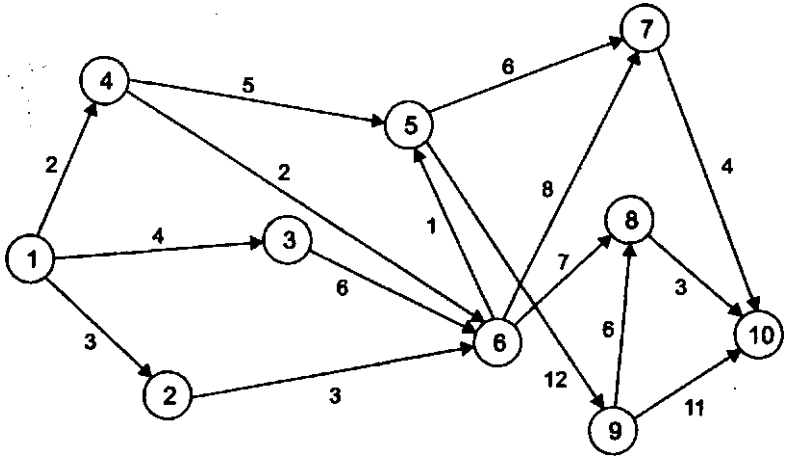


Рис. 10.8. Взвешенный ориентированный граф

Начальные условия: /1 = 0.

Находим последовательно значения функции *fi* (в условных единицах) для каждой вершины ориентированного графа:

$/_2 = \min\{5_2i + /i\} = 3 + /i = 3 + 0 = 3;$

$/3 = \min\{5_3i + /i\} = 4 + /_1 = 4 + 0 = 4;$

$/4 = \min\{S_{41} + /1\} = 2 + fx = 2 + 0 = 2;$

$fb = \min\{S\$4 + /4,\ S_{56} + /б\} = \min\{5 + 2,\ 1 + 4\} = 5;$

$/_6 = \min\{564 + /4,\ 5вз + /з,\ *M + h\} = \min\{2+2,\ 6+4,\ 3 + 3\} = 4;$

$Л = \min\{S75 + /5,\ 5_{76} + h\} = \min\{6 + 5,\ 8 + 4\} = 11;$

$/9 = \min\{595 + /5\} = 12 + 5 = 17;$

$/_8 = \min\{S_{86} + /б,\ 5_{89} + /9\} = \min\{7 + 4,\ 6 + 17\} = 11;$

$До = \min\{5io, 7 + /7,\ £10,8 + /в,\ ^ю.э + /9\} =$

$= \min\{4 + 11,\ 3 + 11,\ 11 + 17\} = 14.$

Длина кратчайшего пути составляет 14 условных единиц. Для выбора оптимальной траектории движения следует осуществить просмотр функций *fi* в обратном порядке, т.е. с $Д_{10}$. Пусть $fi = Д_{10}$. В данном случае

$$/io = \min\{4 + /_7,\ 3 + /_8,\ 11 + /_9\} = \min\{4 + 11,\ 3 + 11,\ 11 + 17\} = 14.$$

Получаем, что $3 + /_8 = 14$, т.е. $fj = /_8$. Значит, из вершины 10 следует перейти к вершине 8. Имеем $fi = /_8$. Рассмотрим функцию

$$/_8 = \min\{7 + /_6,\ 6 + /_9\} = \min\{7 + 4,\ 6 + 17\} = 11,$$

т.е. *fj* = /б и т.д.

Таким образом, получаем кратчайший путь от вершины 1 к вершине 10:

$$(1 - 4 - 6 - 8 - 10).$$

Рассмотренный метод определения кратчайшего пути может быть распространен и на неориентированные графы.

Пример 4. Для графа (рис. 10.8) найти кратчайший путь от вершины 1 до вершины 10, рассматривая граф как неориентированный. Матрица смежности весов графа в этом случае имеет вид:

	1	2	3	4	5	6	7	8	9	10
1	–	3	4	2	–	–	–	–	–	
2	3	–	–	–	–	3	–	–	–	
3	4	–	–	–	–	6	–			
4	2	–	–	–	5	2	–	–	–	–
5	–	–	–	5	–	T	6	–	12	–
6	–	3	6	2	1	–	8	7	–	–
7	–	–	–	–	6	8	–	–	–	4
8	–	–	–	–	–	7	–	–	6	3
9	–	–	–	–	12	–	–	6	–	11
10	–	–	–	–			4	3	11	–

Запишем выражения для функции Д:

$Л = 0;$

$/2 = \min\{/1 + 3,\ /_6 + 3\};$ $/3 = \min\{/1 + 4,\ /_6 + 6\};$

$/4 = \min\{/1 + 2,\ /5 + 5,\ /_6 + 2\};$

$/5 = \min\{/4 + 5,\ /в + 1,\ /э + 12,\ /_7 + 6\};$

$/_6 = \min\{/_2 + 3,\ /3 + 6,\ /4 + 2,\ /5 + 1,\ /7 + 8,\ /_8 + 7\};$

$/7 = \min\{/5 + 6,\ /в + 8\};$ $/_8 = \min\{/_6 + 7,\ /9 + 6\};$

$/9 = \min\{/5 + 12,\ /_8 + 6\};$ $До = \min\{/_7 + 4,\ /_8 + 3,\ /_9 + 11\}.$

Указанные целевые функции представляют собой систему линейных алгебраических уравнений (в примере имеется 10 уравнений и 10 неизвестных).

Рассмотрим один из вариантов ее решения, учитывая, что $/1 = 0$.

Тогда имеем:

$/_8 = 3;$ $/_3^* = 4;$ $/4 = \min\{2, /_5 + 5, /_6 + 2\} = 2;$

$/5 = \min\{7, /б + 1, /9 + 12, /7 + 6\} = \min\{7, /_6 + 1\};$

$/б = \min\{6,\ 10,\ 4,\ /5 + 1,\ /7 + 8,\ /_8 + 7\} = \min\{4,\ /5 + 1\} = 4.$

Подставив выражение для /в в $/5$, получим

$$/_5 = \min\{7,\ 4 + 1\} = 5.$$

Тогда

$/_7 = 11;$ $/_8 = \min\{11, /_9 + 6\};$

$/_9 = \min\{17,\ /_8 + 6\};$ $До = \min\{15,\ /_8 + 3,\ /_9 + И\}.$

Подставляя $/9$ в $/_8$, получаем

$$/_8 = \min\{11,\ 17 + 6,\ /_8 + 6 + 6\} = 11.$$

Окончательно имеем: $/_9 = 17$; До = 14.

Таким образом, кратчайший путь равен (1 - 4 - 6 - 8 - 10).

10.3. Метод Дейкстры

Алгоритм Дейкстры предназначен для нахождения кратчайшего пути между вершинами в неориентированном графе.

Идея алгоритма следующая: сначала выберем путь до начальной вершины равным нулю, и заносим эту вершину во множество выбранных вершин, расстояние от которых до оставшихся невыбранных вершин определено. На каждом следующем этапе находим следующую выбранную вершину, расстояние до которой наименьшее и соединенную ребром с какой-нибудь вершиной из множества невыбранных (это расстояние будет равно расстоянию до выбранной вершины плюс длина ребра).

Пример 1. Найти кратчайший путь на графе от вершины *L* до вершины *D* (рис. 10.9).

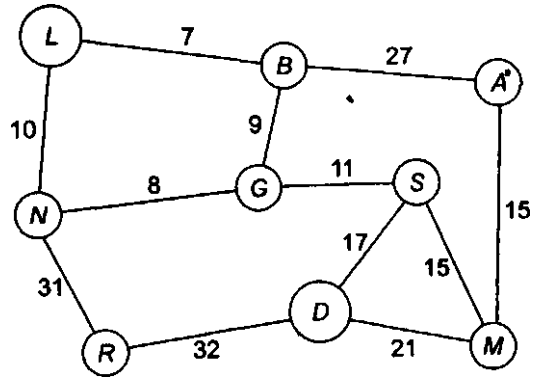


Рис. 10.9. Взвешенный неориентированный граф

Запишем алгоритм в виде последовательности шагов (табл. 10.1).

Таблица 10.1

Метод Дейкстры (первый шаг)

Выбранная вершина	Путь до выбранной вершины	Невыбранные вершины							
		B	***G***	***N***	***R***	***D***	***S***	***M***	***A***
L	0	(7)	00	10	оо	оо	оо	со	оо
B	7								

Шаг 1. Определяются расстояния от начальной вершины *L* до всех остальных.

Шаг 2. Выбираем наименьшее расстояние от *L* до *B,* найденная вершина *B* принимается за вновь выбранную. Найденное наименьшее расстояние добавляется к длинам ребер от новой вершины *B* до всех остальных. Выбирается минимальное расстояние от *Вдо N.* Новая вершина *N* принимается за выбранную (табл. 10.2).

Таблица 10.2

Метод Дейкстры (второй шаг)

Выбранная вершина	Путь до выбранной вершины	Невыбранные вершины							
		B	***G***	ЛГ	***R***	***D***	***S***	***M***	***A***
L	0	(7)	оо	10	00	00	оо	оо	оо
B	7		16	**(10)**	00	00	оо	оо	34
N	10								

Для наглядности в дальнейшем вместо знака оо будем ставить знак «-».

Шаг 3. Определяются расстояния от вершины *N* до всех оставшихся (за исключением *БuB)* (табл. 10.3).

Таблица 10.3

Метод Дейкстры (третий шаг)

Выбранная вершина	Путь до выбранной вершины	Невыбранные вершины							
		B	***G***	***N***	***R***	***D***	5	***м***	***A***
L	0	(7)	00	10	оо	00	оо	оо	оо
B	7		16	**(10)**	00	00	оо	оо	34
N	10		**(16)**	,*: *	41	00	оо	оо	34
G	16								

Расстояние от вершины *L* через вершину *N* до вершины *G* равно 18. Это расстояние больше, чем расстояние *LB* + *BG* = 16, поэтому оно не учитывается в дальнейшем. Продолжая аналогичные Построения, построим табл. 10.4 (величины, указанные в квадратных скобках, не учитываются). Таким образом, найдена длина кратчайшего пути между вершинами L и *D* (44 условные единицы).

Траектория пути определяется следующим образом. Осуществляем обратный просмотр от конечной вершины к начальной. Просматриваем столбец, соответствующий вершине, снизу вверх и фиксируем

первое появление минимальной величины. В столбце, соответствующем вершине *D,* впервые минимальная длина 44 появилась снизу в четвертой строке. В этой строке указана вершина *S,* к которой следует перейти, т.е. следующим нужно рассматривать столбец, соответствующий вершине *S.*

Таблица 10.4

Табличная реализация метода Дейкстры

Выбранная вершина	Путь до выбранной вершины	Невыбранные вершины							
		B	***G***	***N***	***R***	***D***	S	***M***	***A***
L	0	**(7)**	_	10	_	_	_	_	_
B	7	.	16	**(10)**	_	_	_	_	34
N	10	* ;	**(16)**		41	_	_	_	34
G	16	.	»		41	–	16+11 =(27)	–	34
S	27		*w*		41	27+17 =44	..?,-**, . '	27+15 =42	**(34)**
A	34		* i>* .'Л с	*l *•/{*	**(41)**	44		42 [34+15]	
R	41		•	\'	*	44 [41+32]	*	**(42)**	*
M	42			• *i.R.W* ft	*	**(44)** [42+21]	•	*	*

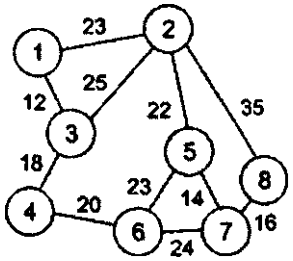
В этом столбце минимальная длина, равная 27, указывает следующую вершину *G,* к которой следует перейти, и т.д. Таким образом, получаем траекторию пути: *(L — B — G — 5 — D).*

Пример **2.** Найти кратчайший путь на графе между 1-й и 7-й вершинами (рис. 10.10).

Определяем следующую выбранную вершину, расстояние до которой наименьшее, и соединенную ребром с одной из вершин, принадлежащих множеству невыбранных. Это расстояние будет равно расстоянию до выбранной вершины плюс длина ребра (табл. 10.5).

Осуществляем обратный просмотр от конечной вершины к начальной. Просматриваем столбец, соответствующий вершине, снизу вверх и фиксируем первое появление минимальной величины.

Кратчайший путь при этом будет равен ***{V; — V5 — V2 — V1).***



m .	23	12	co	co	00	co	co
23	'O*	25	00	22	co	00	35
12	25		18	co	co	CO	CO
00	co	18	...0	CO	20	co	co
oo	22	co	co	' t)	23	,14	CO
00	co	00	20	23	:.o	24	co
oo	co	oo	co	14	24	0	16
oo	35	oo	co	co	CO	16	

Рис. 10.10. Граф и соответствующая ему матрица смежности

Таблица 10.5

Табличная реализация метода Дейкстры

Выбранная вершина	Путь до выбранной вершины	Невыбранные вершины						
			v_3		V_b		v_7	v_a
Vi	0	23	**(12)**	_	_	_	_	_
V2	23	*J fir**,.		**(30)**	45	_	_	58
V_3	12	**(23)**		30	_	_	_	_
	30		*Ф * -«*		ft **(45)**	50	_	58
$V_в$	45	*		*	^ ·	**(50)**	59	58
v_e	50	*	ф		«4»	*.'. .	59	**(58)**
V_s	58		*fi > , i.*	*		*	**(59)**	

10.4. Алгоритм Флойда

Пусть в матрице *A[i,j]* записаны длины ребер графа (элемент *A[i,j]* равен весу ребра, соединяющего вершины с номерами *i* к j. Если же такого ребра нет, то в соответствующем элементе записано некоторое очень большое число. Построим новые матрицы *Ck[i,j] (k = 0,...,N).* Элемент матрицы *Ck[i,j]* будет равен минимально возможной длине такого пути из г в *j,* что в качестве промежуточных вершин в этом пути используются вершины с номерами от 1 до А, ,' т.е. рассматриваются пути, которые могут проходить через вершины с номерами от 1 до *к,* но заведомо не проходят через вершины с Номерами от *к* + 1 до *N.* В матрицу записывается длина кратчайшего 1в таких путей. Если таких путей не существует, записывается то же большое число, которым обозначается отсутствие ребра.

Если вычислена матрица *Ck-\[i,j],* то элементы матрицы $C_k[i, j]$ можно вычислить по следующей формуле

$$C_k[i,j] = mm(C_k\text{-}i[i,j], C_k\text{-}i[i,k] + C_{fc}\_i[fc,j]).$$

В самом деле, рассмотрим кратчайший путь из вершины *i* в вершину *j,* который в качестве промежуточных вершин использует только вершины с номерами от 1 до *к.* Тогда возможно два случая:

• этот путь не проходит через вершину с номером *к.* Тогда его промежуточные вершины — это вершины с номерами от 1 до *к* - 1. Но тогда длина этого пути уже вычислена в элементе C_k-*i[i,j];*

• этот путь проходит через вершину с номером *к.* Но тогда его можно разбить на две части: сначала из вершины *г* доходим оптимальным образом до вершины *к,* используя в качестве промежуточных вершины с номерами от 1 до А; - 1 (длина такого оптимального пути вычислена в C_k-i *[г, к]),* а потом от вершины *к* идем в вершину *j* опять же оптимальным способом, используя в качестве промежуточных вершин только вершины с номерами от 1 до А; *(C_k-i[k, j]).*

Выбирая из этих двух вариантов минимальный, получаем $C_k[i,j]$.

Последовательно вычисляя матрицы Со, *Си* Сг и т.д., получим искомую матрицу *С^* кратчайших расстояний между всеми парами вершин в графе.

Алгоритм Флойда можно свести к последовательности шагов.

Присвоить *aj* = 0 для всех г = 1,2,... ,п и Cj,- = оо, если в графе отсутствует дуга *[x{,* ***XJ].***

Присвоение начальных значений.

Ш а г 1. Присвоить *к* = 0.

Ш а г 2. А; = *к + 1.*

Ш а г 3. Для всех *г ф к,* таких, что c_{ik} *ф* оо, и для всех; *ф к,* таких, что *сц, Ф* оо, *(Hj* = min {су, $c_{,fc}$ + c_{kj}}.

Проверка на окончание.

Ш а г 4. Если А; = *п,* то матрица [с*,] дает длины всех кратчайших путей. Остановка. Иначе перейти к шагу 2.

10.5. Алгоритм Йена

Пусть граф задан матрицей *A[г, j].* Вершина *s* выбрана как начальная. Найдем длины *к* наименьших путей до каждой вершины от вершины *s* (ребра в одном пути могут повторяться неоднократно), при условии, что эти пути существуют. Результат будет храниться в матрице *В* размером *N* х *к,* где *N* — количество вершин. Элемент массива *B[i,j]* равен *j-му* по длине пути до вершины *г.*

Для каждой вершины в массиве *С* будем хранить количество уже найденных до нее наименьших путей. Изначально все элементы массива *С* равны нулю. Все элементы матрицы *В* делаем равными какой-нибудь большой константе, заведомо большей всех возможных путей. Во время исполнения алгоритма в матрице *В* будем хранить лучшие *к* путей до каждой вершины, найденные во время исполнения, при этом первые *С[г]* путей для вершины *г* уже найдены окончательно (элементы матрицы *В — B[г,* 1], *B[г,2],B[г, к]* для всех *i* упорядочены в возрастающем порядке).

Следовательно, можно действовать следующим образом: пусть уже найдены какие-то кратчайшие пути, тогда чтобы найти следующий по длине, попробуем удлинить каждый из уже полученных на одно ребро. Найдем наикратчайший из них, причем оканчивающийся на вершину, до которой найдено менее *к* путей. Его можно вносить в таблицу результата.

Алгоритм Йена можно свести к последовательности шагов.

Присвоение начальных значений.

Ш а г 1. Найти *P'.* Присвоить *к* — 2. Если существует только один кратчайший путь P^1, включить его в список ***LQ*** **И** перейти к шагу 2. Если таких путей несколько, но меньше, чем *К,* включить один и них в список $\boldsymbol{L}_c$ **,** а остальные в список ***L \ .*** Перейти к шагу 2. Если существует *К* или более кратчайших путей P^1, то задача решена. Остановка.

Нахождение отклонений.

Ш а г 2. Найти все отклонения *Pf* (А;-1)-го кратчайшего пути P^{fc-1} для всех i = 1, 2, ..., q_k-\, выполняя для каждого *г* шаги с 3-го по 6-й.

Ш а г 3. Проверить, совпадает ли подпуть, образованный первыми {вершинами любого из P^j путей *(j* = 1, 2,..., *к* — 1). Если да, то присвоить *c{x\~'* = оо; иначе ничего не изменять. (При выполнении алгоритма вершина xi обозначается *s.)*

Ш а г 4. Найти кратчайшие пути 5* от *x^~^* к *t,* исключая из рассмотрения вершины *s, x*~~', ж*^{-1}*, Если существует несколько кратчайших путей, то взять в качестве один из них.

Ш а г 5. Построить и поместить Р в список ***L \ .***

Ш а г 6. Заменить элементы матрицы весов, измененные на шаге их первоначальными значениями и возвратиться к шагу 3.

Выбор кратчайших отклонений.

Ш а г 7. Найти кратчайший путь в списке *L\.* Обозначить этот путь $P^{к}$ и переместить его из ***L*** в ***LQ.*** Если *к* = ***К,*** то остановка. Список ***LQ*** — список /Г-кратчайших путей.

Если *к* < *К,* то присвоить А; = *к* + 1, перейти к шагу 2. Если в *L* имеется более чем один кратчайший путь (h-путь), то поместить в ***LQ*** любой из них и продолжать вычисления до тех пор, пока увеличенное на *h* число путей, уже находящихся в *L\,* не совпадет с *К* или не превысит его. Тогда — остановка.

10.6. Алгоритм Беллмана — Форда

Пусть в матрице *A[i,j]* записаны длины ребер графа. Найдем кратчайшие расстояния от заданной вершины *s* до всех остальных вершин графа. Алгоритм Беллмана — Форда решает эту задачу даже при наличии ребер отрицательного веса. Обозначим через МинСт(з, *и, к)* наименьшую стоимость проезда из *з* в *v* менее чем с *к* пересадками:

МинСт(в,**1**/,А;+1) = тш{МинСт(а,**1**',А:), *МивСт(з,1,к)* + *A[i,v]}.*

Искомым ответом является МинСт(5,г,тг) для всех г = 1..п.

Чтобы найти не только длины наименьших путей до всех вершин, но и сами эти пути, используем следующий прием. Параллельно с вычислением массива *x* будем вычислять матрицу *D[i, j].* Если между вершинами *s* и *j* существует путь, то в элементе массива JD[j,0] будет хранится количество вершин в этом пути, а цепочка вершин, составленная из элементов с *D\j,* 1] по *D{j, D\j,* 0]], будет этим самым путем. Путь до вершины *s* содержит единственную вершину (£>[з,0] = 1; .D[s,l] = s). Для вычисления матрицы *D* потребуется дополнить текст процедуры:

```
к:=1;
for i := 1 to n do begin x[i] := a[s][i]; end;
{инвариант: x[i] := MHHCr(s,i,k)}
```

Обозначим через МинСт(з,г,к) наименьшую стоимость проезда из *s* в г менее чем с А; пересадками. Тогда выполняется такое соотношение:

```
for i := 1 to n do begin D[i,0]:=2; D[i,l]:=s; D[i,2]:=i; End;
D[s,0]:=l; while k <> n do begin
    for i := 1 to n do begin
        for j := 1 to n do begin
            if x[i] > x[j]+a(j][i] then begin
                x[i]:=x[j]+a[j]M;
                D[i]:=DD];
                D[i,0]:=DD,0]+l;
                D[i,D[i,0]]:=i;
            end;
        end
        {x[i] = MHHdis.iJc+l)}
    end;
    k:=k+l;
end;
```

Контрольные вопросы

1. В чем особенность построения дерева решений?
2. Что показывает дерево решений?
3. Каким образом используется метод оптимизации для определения кратчайшего пути по карте местности?
4. В решении каких прикладных задач используются алгоритмы определения в графе кратчайших расстояний между заданными вершинами?
5. Может ли быть применен алгоритм Дейкстры к определению кратчайшего расстояния в ориентированном графе?
6. Как работает алгоритм Дейкстры?
7. Как работает алгоритм динамического программирования применительно к задаче определения в графе кратчайших расстояний между вершинами?
8. В чем особенность алгоритма Флойда?
9. В чем особенность алгоритма Йена?
10. Укажите основной принцип построения алгоритма Беллмана—Форда.

Глава 11
Эвристические алгоритмы

Эвристический алгоритм — это алгоритм, в котором на определенном этапе используется интуиция разработчика. Если решение, принятое разработчиком, окажется неверным, результат вс}е равно будет получен, но за большее число шагов. Таким образов, в эвристических алгоритмах можно увеличить скорость получения правильного результата.

К эвристическим алгоритмам относятся: волновой, двухлучево^, четырехлучевой, маршрутный, алгоритмы составления расписания.

11.1. Волновой алгоритм

Волновой алгоритм, или алгоритм Ли, первоначально ис пользовался для поиска пути в лабиринте или в игровых задачах настоящее время алгоритм Ли (волновой) является основным в микро электронике для трассировки (соединения) элементов интегральных схем. Особенность алгоритма состоит в следующем:

• в лабиринте (на подложке) выбираются две точки *A* (начальная) и *B* (конечная). Из начальной точки в четырех направлениях выходшт волна (рис. 11.1). Цифрами обозначается номер фронта волны или ее путевые координаты;

• путевые координаты определяют шаг распространения волны Каждый элемент первого фронта волны является источником вторич ной волны (рис. 11.2).

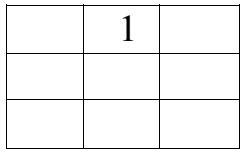


Рис. 11.1. Направления распространения лучей

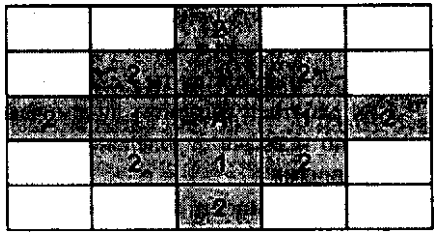


Рис. 11.2. Второй фронт волны

Элементы второго фронта генерируют третий фронт и т.д. От запрещенных элементов волна не распространяется. Процесс продолжается до тех пор, пока не будет достигнут конечный элемент.

Траектория пути определяется обратным просмотром, от конечного к начальному. При этом задаются приоритеты движения:

вверх, вниз, влево, вправо.

От того, в каком порядке заданы приоритеты, зависит скорость решения задачи.

При построении траектории используются два принципа:

• движение осуществляется строго по заданным приоритетам;

• при построении трассы, т.е. траектории движения, значения путевых координат должны уменьшаться.

П р и м е р **1.** Пусть задан лабиринт, где запрещенные элементы заштрихованы. Найти путь между элементами Я и *К.*

На первом этапе от начального элемента трассы исходит волна, которая останавливается, достигнув конечного элемента (рис. 11.3). Волна распространяется по четырем направлениям: вверх, вниз, вправо и влево. Первый фронт волны представляет собой четыре единицы. Следующий фронт представляют собой распространение волны на одну клетку от элементов предыдущего фронта в стороны от начального элемента. При распространении волны элементам присваивают значения номера фронта (2, 3, 4 и т.д.). Если элемент запрещенный, то

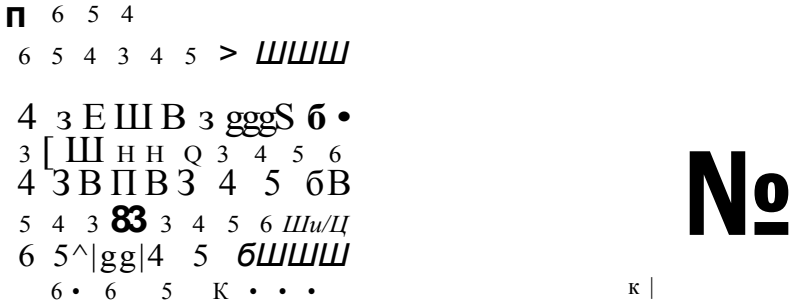


Рис. **11.3. Распространение волны (а) и построение трассы (б)**

к нему это не относится — волна распространяется только по незапрещенным элементам сетки. Также волна не распространяется за границы координатной сетки.

Следующим этапом является нахождение пути, причем поиск ведется от конечного элемента (рис. 11.3). Выбираются приоритетные направления движения от конечного элемента к начальному: вверх, влево, вниз, вправо. Трасса прокладывается с учетом уменьшения путевых координат.

Пример **2.** Пусть задан лабиринт, где запрещенные элементы заштрихованы. Найти путь между элементами *A* и *B* волновым алгоритмом (рис. 11.4).

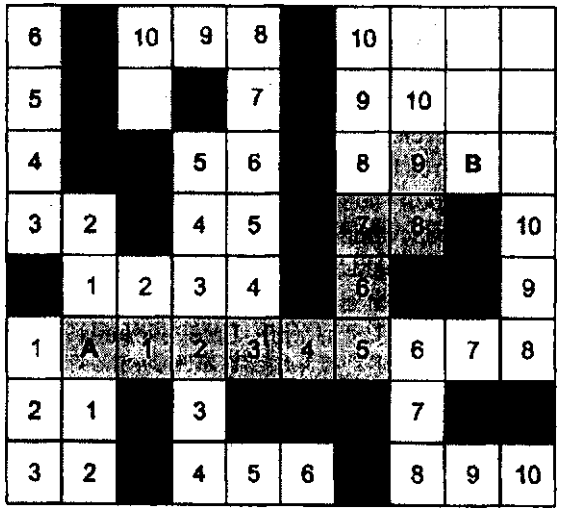


Рис. 11.4. Траектория пути между элементами ***A*** и ***B***

На первом этапе от элемента *A* распространяется волна до тех пор, пока она не достигнет конечного элемента *B*. На втором этапе выбираются приоритеты движения от конечной точки *B* к начальной *A*. Приоритеты выбираются исходя из взаимного расположения начального и конечного элемента. Предположим, что выбраны парадоксальные (логически неверные) приоритеты : вверх, вправо, вниз, влево. В этом случае трасса все равно будет построена, но за большее число шагов (сравнений).

11.2. Двухлучевой алгоритм

В двухлучевом алгоритме из начального и конечного элементов одновременно выходят по два луча, трасса считается проведенной, если пересекаются два разноименных луча (от разных источников). Если на пути луча встречается запрещенный элемент, его обход осуществляется по второму приоритетному направлению, характерному для лучей, выходящих из одной точки.

Если же оба направления оказываются заблокированными запрещенными элементами либо достигнут край координатной сетки, то движение луча прекращается.

Выбор направления движения лучей происходит исходя из интуиции разработчика. Существуют четыре варианта распространения лучей, которые определяются после вычисления значений *a* и */3:*

если $X_A - X_B > 0$; если $Y_A - Y_B > 0$;
если $X_A - X_B < 0$; **1** 0, если $Y_A - Y_B < 0$,

где ***(XA, УА)*** — координаты начального элемента;
(Хв, Ув) — координаты конечного элемента.

Исходя из значений *a* и *ft* выбирают направления лучей (рис. 11.5).

a=0, /3=0 =1,y?=0 a=0, /3=1 a=1, /5=1

• 8 В *

Рис. 11.5. Варианты распространения лучей

Пример **1.** Осуществить трассировку элементов *A и B,* расположенных на подложке интегральной схемы (рис. 11.6).

В данном случае $X_A - X_B = 3 - 10 = -7$, $Y_A - Y_B = 9 - 2 = 7$. Значит, a = 0, /? = 1.

Каждый из двух лучей, выходящих из одного элемента, движется по заданному приоритетному направлению (рис. 11.5). Если луч встречает на пути запрещенный элемент, то его обход осуществляется по второму приоритетному направлению. Если оба направления

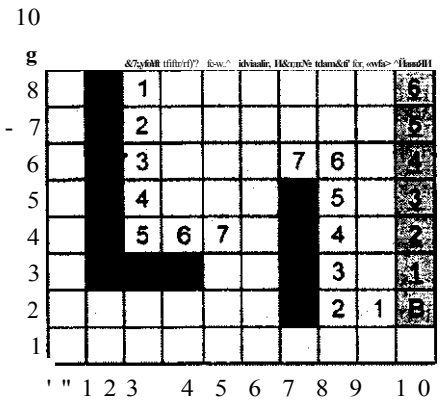


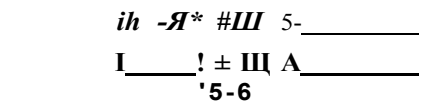
Рис. 11.6. Построение трассы двухлучевым алгоритмом

оказываются заблокированными запрещенными элементами, то движение луча прекращается. При достижении лучом края координатной сетки он затухает.

Трасса считается найденной, если встречаются два разноименных луча, т.е. лучи от разных источников.

Пример 2. Осуществить трассировку элементов *A* и *B,* расположенных на подложке интегральной схемы (рис. 11.7).

В данном случае $X_A - X_B = 2-7 = -5$, Уд - Уд = 7 - 2 = 5. Значит, *а* = 0, /3 = 1. Исходя из значений *а* и /3 определяются направления движения лучей (рис. 11.5).



1.

В

Рис. 11.7. Трассировка двухлучевым алгоритмом
(* — луч прекращает движение)

11.3. Четырехлучевой алгоритм

Из начальной и конечной точек выходят одновременно по четыре луча (рис. 11.8). Лучи движутся строго по заданным направлениям и «затухают», если достигли края координатной сетки либо встретили запрещенный элемент.

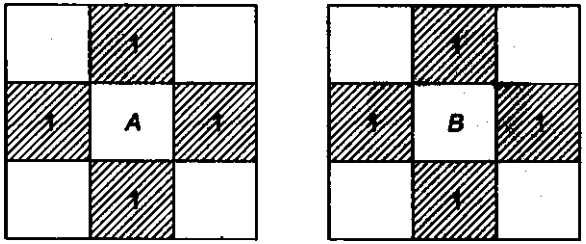


Рис. 11.8. Направления движения лучей из точек *A* и *B*

Пример 1. Осуществить трассировку (соединение) элементов интегральной схемы четырехлучевым алгоритмом (рис. 11.9).

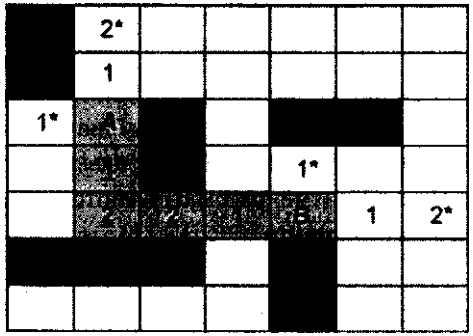


Рис. 11.9. Трассировка четырехлучевым алгоритмом

Если луч встречает на пути запрещенный элемент или достигает края координатной сетки, то он затухает. Трасса считается найденной, если встречаются два разноименных луча.

Пример 2. Осуществить трассировку элементов четырехлучевым алгоритмом (рис. 11.10). В данном примере трассу провести нельзя.

Рис. 11.10. Трассировка четырехлучевым алгоритмом

11.4. Маршрутный алгоритм'

Маршрутный алгоритм получил свое название, потому что осуществляет одновременно и формирование фронта, и прокладывание трассы. Источником волны на каждом шаге является конечный элемент участка трассы, проложенной на предыдущих шагах.

В маршрутном алгоритме рассматриваются восьмиэлементная окрестность исходного элемента (рис. 11.11).

/-1,7-1		/-1.У+1
". 7-1	*A*	/, у +1
/+1, у-1	;'+1, у	/+1, У+1

Рис. 11.11. Восьмиэлементная окрестность

От каждого элемента окружения оценивается расстояние *d* до конечного элемента *B:* $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ или $d = |x_i - x_e| + |y_i - y_e|$.

Таким образом определяются восемь значений расстояний, из которых выбирается минимальное. Элемент, для которого *d* оказалось минимальным, считаем элементом трассы. Процесс повторятся до тех пор, пока расстояние не будет равным нулю *(d* = 0), т.е. пока не будет достигнут конечный элемент. Обход запрещенных элементов осуществляется исходя из интуиции разработчика.

Пример 1. Осуществить трассировку элементов маршрутным алгоритмом (рис. 11.12).

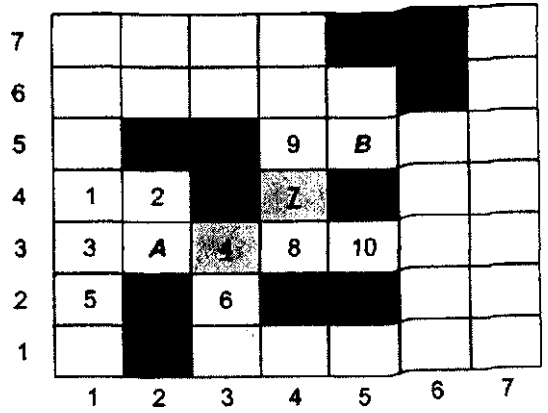


Рис. 11.12. Нумерация элементов, на данном уровне в маршрутном алгоритме

Определим расстояния элементов окрестности точки *A* до точки *B*.

$d_1 = \sqrt{(1-5)^2 + (4-5)^2} = \sqrt{17}, \quad d_2 = \sqrt{(2-5)^2 + (4-5)^2} = \sqrt{10},$

$d_3 = \sqrt{(1-5)^2 + (3-5)^2} = \sqrt{20}, \quad d_4 = \sqrt{(3-6)^2 + (3-5)^2} = \sqrt{13},$

$d_5 = \sqrt{(1-5)^2 + (2-5)^2} = \sqrt{25}, \quad d_6 = \sqrt{(3-5)^2 + (2-5)^2} = \sqrt{13}.$

Минимальным является расстояние $d^{\wedge} = y/S$- Далее считаем расстояния от точки *B* до точек окрестности найденной точки:

$$d_7 = \pi/2, \quad d_8 = \sqrt{b}$$

Минимальное расстояние *dy.* Проводим вычисления далее:

$$dg = 1, \quad d_{10} = V2, \quad d_{n} = 0.$$

Получили маршрут (Л - 4 - 7 - *В).*

11.5. Геометрическая модель задачи о лабиринте

Пример **1.** В лабиринте с произвольными препятствиями найти кратчайший путь между заданными точками *S и T* (рис. 11.13).

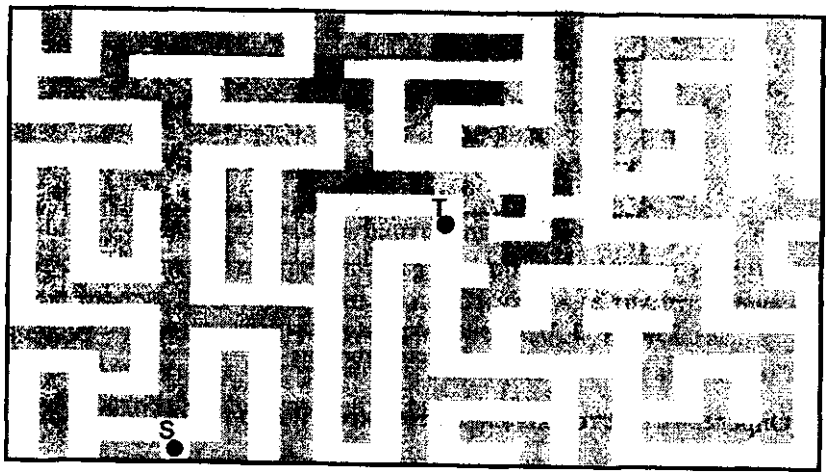


Рис. 11.13. Исходный лабиринт

Решение. Так как препятствия на местности образуют многоугольники или какие-либо другие геометрические фигуры (которые с некоторыми погрешностями тоже можно изобразить в виде многоугольников), то кратчайшая трасса будет являться ломаной с узлами в вершинах этих многоугольников. Звено ломаной — это либо сторона многоугольника, либо прямолинейный отрезок, проходящий вне -•чогоугольников и соединяющий две вершины одного и того же или разных многоугольников. Для решения этой задачи нужно построить сеть (ломаную), а также соединить точки *s* и *t* с простреливаемыми из них вершинами, если эти точки не являются вершинами многоугольников.

Формирование сети, т.е. матрицы расстояний *С* размером п х п (п — общее число вершин всех многоугольников плюс два для учета старта и финиша) представляет собой тройной цикл: внешний по г — перебор вершин, откуда стреляют; средний по j Q' от г + 1 до *n,* чтобы не повторяться) — это перебор вершин, куда стреляют; и внутренний по А: — это проверка, не пересекает ли *к-я* сторона какого-либо многоугольника отрезок соединения (рис. 11.14).

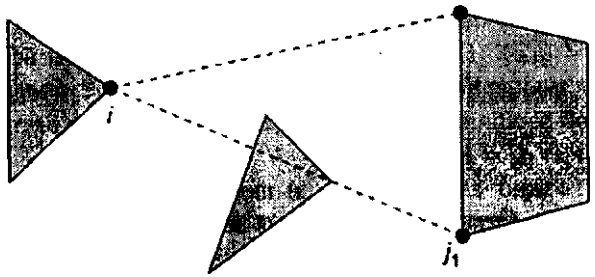


Рис. 11.14. Направления стрельбы из точки г (показаны пунктиром): (i, *j\)* не входит в сеть, (г, j_2) входит в сеть

Последнее условие проверяется по стандартным формулам аналитической геометрии: записывается уравнение прямой, проходящей через (г, *j),* и уравнение прямой, проходящей через концы отрезка *к.* Решением системы из этих двух уравнений находится точка пересечения и устанавливается, лежит ли точка пересечения внутри рассматриваемых отрезков (рис. 11.15). Если да, то *dij* = оо, конец цикла по *к;* если нет пересечения по окончании цикла по *к,* то вычисляется евклидово расстояние *dij.*

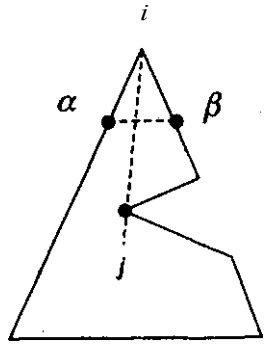


Рис. 11.15. Определение точки пересечения отрезков

V

Пусть между вершинами г и j не проходит никакой стены, а *j* из г не простреливается. Чтобы преодолеть эту трудность, нужно ввести характеристику г угла препятствия присвоив & = 0, если «вогнутый» угол, или <Л = 1, если «выпуклый» угол. Так, для угла с вершиной i (ft = 1, а для угла с вершиной *j gi* = 0.

Если крайние вершины x* и x^з (XJ, ij+i, x*.**2.** xi+з—последовательные вершины многоугольника) лежат по одну сторону от прямой, проходящей через соседние вершины **Xi+i, Xi+2,** то *gt+* = **<?i+2>** иначе *9i+\ Ф 9i+2-*

$$(x - X_{,}i)(y_{,}2 \sim Уг+l) - (Xi+2 - Xi+l)(V - Уг+l) = 0.$$

Если при подстановке в это уравнение точек (XJ, *yi)w{xi+z, Vi+з)* в левой части получаются числа с одинаковым знаком, то *∂{+* **= &+2,** иначе *gi+i Ф* р,.г. После этого цикла будут известны все & точно или с точностью до наоборот. Остается абсолютно установить *gi* хотя бы для одной вершины. Это легко сделать, потому что экстремальная вершина имеет $g_{\varrho} = 1$.

Теперь можно справиться с вышеизложенной проблемой. Из вершины *г* не простреливается никакая вершина j, защищенная углом с вершиной г. Чтобы исключить из рассмотрения загороженные вершины, нужно отступить от вершины г по сторонам угла на величину £, заведомо меньшую, чем длина стороны, построив таким образом точки а и /3. После этого нужно ввести бинарную величину *B,* полагая *B = 1,* если отрезки а/9 и *ij* пересекаются, и *B* = 0, если отрезки а/3 и *ij* не пересекаются (рис. 11.16).

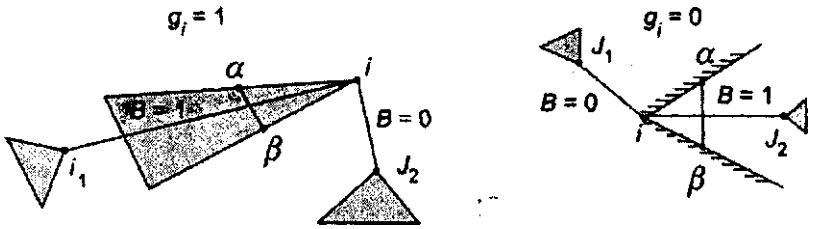


Рис. 11.16. Характеристики угла препятствия

Всего имеется четыре возможности:

1) В = 1 и & = 0;

2) В = 0 и $_{л}$ - = 1;

3) В = 1 и & = 0;

4) В = 1 и я = 1.

Ясно, что вершина *j* не простреливается в случаях 2 и 3 (при нечетном *В + д).* Теперь можно построить сеть.

После того как сеть построена, можно приступать к нахождению кратчайших путей, воспользовавшись любым из рассмотренных выше алгоритмов (в зависимости от поставленной задачи) — рис. 11.17.

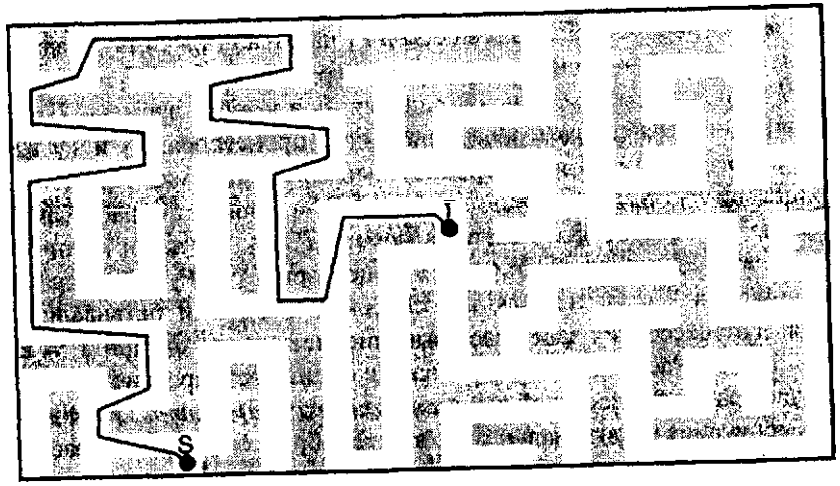


Рис. 11.17. Траектория пути в лабиринте

11.6. Алгоритмы составления расписания

Предположим, что имеется *n* одинаковых процессоров, обозначенных *P\, **P2,** Pn,* и *m* независимых заданий J i, *J?,..., J„,* которые нужно выполнить. Процессоры могут работать одновременно, и любое задание можно выполнять на любом процессоре. Если задание загружено в процессор, оно остается там до конца обработки. Время обработки задания У» известно и равно *U, г* = 1,2,..., т. Организовать обработку заданий таким образом, чтобы выполнение всего набора заданий было завершено как можно быстрее.

Система работает следующим образом: первый освободившийся процессор берет из списка следующее задание. Если одновременно освобождаются два или более процессоров, то выполнять очередное задание из списка будет процессор с наименьшим номером.

Пример. Пусть имеются три процессора и шесть заданий, время выполнения каждого из которых $t\backslash = 2$, $i_2 = 5$, $t\$ = 8$, $Ц = 1$, $tb = 5$, $te = 1$.

Рассмотрим расписание $L = (J_2, J_5, J_1, J_4, J_6, J_3)$. В начальный момент времени Т = 0 процессор Pi начинает обработку задания J_2, процессор P_2 — задания **J5**, а процессор Рз — задания $J\backslash$. Процессор Рз заканчивает выполнение задания $J\backslash$ в момент времени Т = 2 и начинает обрабатывать задание **J4**, пока процессоры P_2 и P_2 все еще работают над своими первоначальными заданиями. При Г = 3 процессор P_3 опять заканчивает задание **J4** и начинает обрабатывать задание $J\$$, которое завершается в момент Г = 4. Тогда он начинает выполнять последнее задание J_3. Процессоры P_2 и P_2 заканчивают задания при $T = 5$, но, так как список L пуст, они останавливаются. Процессор Рз завершает выполнение задания J_3 при Т = 12 (рис. 11.18). Рассмотренное расписание проиллюстрировано временной диаграммой, известной как схема *(диаграмма) Гантпа.* Очевидно, что расписание не оптимально. Можно подобрать оптимальное расписание!,* = (7з, J_5, **J5**, $J\backslash$, **J4**, J_6), которое позволяет завершить все задания за минимальное время $T^* = 8$ единиц (рис. 11.19).

Можно рассмотреть другой тип задач по составлению расписания для многопроцессорных систем. Вместо вопроса о быстрейшем завер-

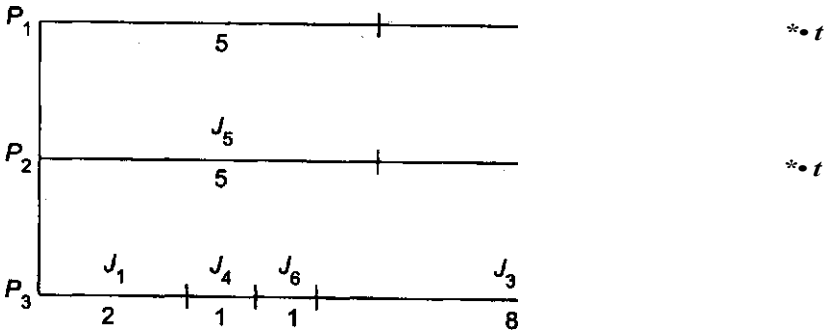


Рис. **11.18.** Схема Ганта: расписание ***L***

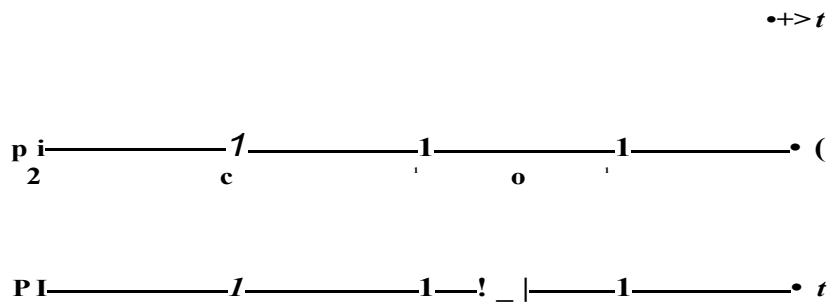


Рис. 11.19. Схема Ганта: расписание!,*

шении набора заданий фиксированным числом процессоров теперь поставим вопрос о минимальном числе процессоров, необходимых для завершения данного набора заданий за фиксированное время То. Конечно, время То будет не меньше времени выполнения самого трудоемкого задания.

11.7. Задача упаковки

Задача составления расписания эквивалентна следующей задаче упаковки. Пусть каждому процессору ***Pj*** соответствует ящик ***Bj*** размера То. Пусть каждому заданию Jj соответствует предмет размера ***ti,*** равного времени выполнения задания Jj, где г = 1,2,..., ***n.*** Теперь для решения задачи по составлению расписания нужно построить алгоритм, позволяющий разместить все предметы в минимальном количестве ящиков. Конечно, нельзя заполнять ящики сверх их объема Го, и предметы нельзя дробить на части.

Пусть дано множество предметов: два из них имеют размер 3 и три — размер 2. Какое требуется минимальное количество ящиков размера 4, чтобы поместить в них все предметы? Ответ показан на рис. 11.20, где заштрихованные участки обозначают пустые места в ящиках.

Для решения этой задачи существует несколько простых эвристических алгоритмов. Каждый из них может быть описан в нескольких строках. Рассмотрим четыре таких алгоритма.

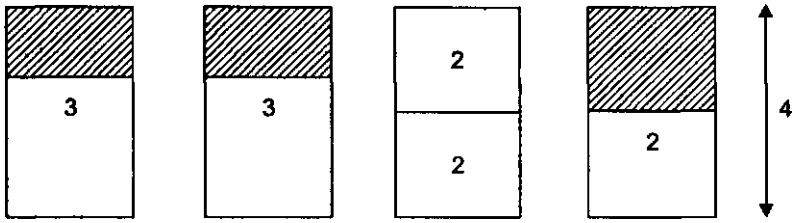


Рис. 11.20. Минимальное число ящиков размера 4, требуемое для размещения предметов размерами 2, 2, 2, 3, 3

Алгоритм 1 *(первое попавшееся размещение для заданного списка).* Пусть L — некоторый порядок предметов. Первый предмет из L кладем в ящик B\. Второй предмет из L кладем в Bi, если он туда помещается. В противном случае кладем его в следующий ящик B^. Вообще, очередной предмет из L кладем в ящик Bi, куда этот предмет поместится, причем индекс i наименьший среди всех возможных. Если предмет не помещается ни в один из $к$ частично заполненных ящиков, кладем его в ящик $B_к$+\. Этот основной шаг повторяется, пока список L не будет исчерпан.

Алгоритм 2 *(первое попавшееся размещение с убыванием).* Этот алгоритм отличается от алгоритма 1 тем, что список L упорядочен от больших предметов к меньшим.

Алгоритм 3 *(лучшее размещение для заданного списка).* Пусть L — заданный список заданий. Основной шаг тот же, что и в алгоритме 1, но очередной предмет кладется в тот ящик, где остается наименьшее неиспользованное пространство. Таким образом, если очередной предмет имеет, например, размер 3 и есть четыре частично заполненных ящика размера 6, в которых осталось 2, 3, 4 и 5 единиц незаполненного пространства, тогда предмет кладется во второй ящик, и тот становится полностью заполненным.

Алгоритм 4 *(лучшее размещение с убыванием).* Этот алгоритм такой же, как и алгоритм 3, но список L упорядочен от больших предметов к меньшим.

Для задачи упаковки было найдено несколько хороших верхних оценок. Одна из лучших, принадлежащая Грэхему, выглядит следующим образом. Пусть NQ — минимальное число необходимых ящиков, полученное при помощи точного алгоритма. Если N/a обозначает

приближенное решение, найденное алгоритмом 2, то для любого $e > 0$ и достаточно большого NQ имеет место соотношение

$$\frac{N_A 2}{No} < \frac{\Pi}{9} Ve.$$

Таким образом, алгоритм 2 дает решение, которое не может быть хуже оптимального более чем на 23%. Это гарантированная оценка; для любой конкретной задачи ошибка может быть намного меньше.

Задачи упаковки (и вообще некоторые эвристические алгоритмы) удивительно обманчивы. Например, если нужно упаковать в ящики размера 1000 предметы размерами

$$L = (760, 395, 395, 379, 379, 241, 200, 105, 105, 40),$$

то эвристический алгоритм 2 потребует для этого $N_A 2 = 3$ ящика. Легко проверить, что это на самом деле оптимальное решение. Но если уменьшить размеры каждого предмета на 1, что приводит к списку

$$L' = (759, 394, 394, 378, 378, 240, 199, 104, 104, 39),$$

то эвристический алгоритм 2 уже даст $N_A 2 — 4$ ящика. Это очевидно не оптимальное и совершенно неожиданное решение.

Обратим внимание на другую аномалию. Применяя эвристический алгоритм 1, упакуем $L = (7, 9, 7, 1, б, 2, 4, 3)$ в ящики размера 13. Находим, что N_A\ = 3, как показано на рис. 11.21; очевидно, что этот результат оптимальный.

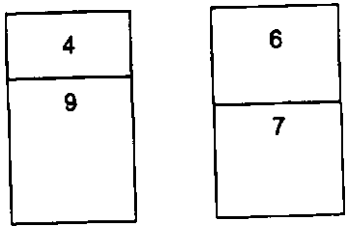


Рис. 11.21. Оптимальная упаковка

Уберем из списка предмет размера 1, что приводит к списку $V = (7, 9, 7, 6, 2, 4, 3)$. Казалось бы должен получиться результат не хуже предыдущего. Однако нетрудно убедиться, что решение ухудшается. Это решение показано на рис. 11.22.

На рис. 11.22 заштрихованные участки показывают неиспользованные объемы тары.

Рис. 11.22. Использование эвристического алгоритма первого попавшегося размещения

11.8. Задача о джипе

Пусть необходимо пересечь на джипе 1000-километровую пустыню, израсходовав при этом минимум горючего. Объем топливного бака джипа 500 л, горючее расходуется равномерно, по одному литру на километр. При этом в точке старта имеется неограниченный резервуар с топливом. Так как в пустыне нет складов с горючим, необходимо установить свои собственные хранилища и наполнять их топливом из бака машины.

Итак, идея задачи ясна: нужно из точки старта отъезжать с полным баком на некоторое расстояние, устраивать там первый склад, оставлять там какое-то количество горючего из бака, но такое, чтобы хватило вернуться назад. В точке старта вновь производится полная заправка и делается попытка второй склад продвинуть в пустыню дальше. Но где обустраивать эти склады и сколько горючего оставлять в каждом из них?

Подойдем к этой задаче с помощью метода отрабатывания назад. С какого расстояния от конца можно пересечь пустыню, имея запас горючего в точности А; баков? Рассмотрим этот вопрос для $к$ = 1,2, 3,..., пока не будет найдено такое целое п, что n полных баков позволяет пересечь всю 1000-километровую пустыню.

Для $к$ = 1 ответ, очевидно, равен 500 км, как показано на рис. 11.23. Можно заправить машину в точке B и пересечь оставшиеся 500 км пустыни. Ясно, что это наиболее отдаленная точка, стартуя из которой можно преодолеть пустыню, имея в точности 500 л горючего.

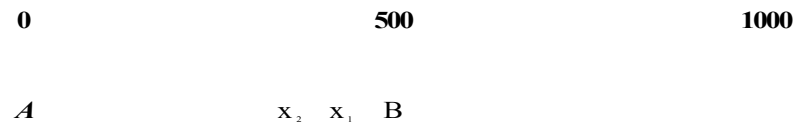


Рис. 11.23. Схема пути через пустыню

Вначале была поставлена частная цель: какое расстояние можно проехать на заданном количестве топлива?

Предположим, что $к$ = 2, т.е. имеется два полных бака (1000 л). Будем рассматривать этот случай, опираясь на результат для $к$ = 1. Данная ситуация иллюстрируется на рис. 11.23. Каково максимальное значение x_1, такое, что, отправляясь с 1000л горючего из точки 500 - x_1, можно перевезти достаточно горючего в точку, чтобы завершить поездку, как в случае $к$ = *11*

Один из способов определения приемлемого значения x_1 состоит в следующем. Заправляемся в точке 500 - x_1, едем x_1 километров до В и переливаем в хранилище все горючее, кроме литров, которые потребуются для возвращения в точку 500 — x_1. В этой точке бак становится пустым. Теперь наполняем второй бак, проезжаем x_1 километров до *B,* забираем в *B* горючее, оставленное там, и из УЗ едем в С с полным баком. Общее пройденное расстояние состоит из трех отрезков по x_1 километров и одного отрезка *BC* длиной 500 км. Тогда x_1 находим из уравнения

$$3x_1 + 500 = 1000.$$

Отсюда находим решение: $x_1 = 500/3$. Таким образом, два бака (1000 *л)* позволяют проехать

$$D_2 = 500 + x_1 = 500 (1 + |) .$$

Заметим, что исходная предпосылка является недальновидной и грубой. Когда имеются в распоряжении $к$ баков горючего, мы просто стараемся продвинуться назад как можно дальше от точки, найденной для $к$ — *1* баков.

Рассмотрим $к$ = 3. Из какой точки можно выехать с 1500 л топлива так, что машина сможет доставить 1000 л в точку 500 - x_1? Возвращаясь к рис. 11.23, найдем наибольшее значение x_2, такое,

что, выезжая с 1500 л топлива из точки 500 — x_1 — хг, можно было доставить 1000 л в точку 500 — x_1. При этом выезжаем из точки 500 — x_1 — ***X2,*** доезжаем до 500 - x_1, переливаем все горючее, кроме хг л, и возвращаемся в точку 500 — xi - хг с пустым баком. Повторив эту процедуру, затратим 4x2 л на проезд и оставим 1000 — 4хг л в точке 500 — xi. Теперь в точке 500 - xi — хг осталось ровно 500 л. Заправляемся последними 500 л и едем в точку 500 - x_1, израсходовав на это хг л.

Если находимся в точке 500 — xi, то на проезд будет затрачено 5x2 л топлива. Здесь оставлено в общей сложности 1500 - 5хг л. Это количество должно быть равно 1000 л, т.е. **X2** = 500/5. Из этого заключаем, что 1500 литров позволяют проехать

$$£»з = 500 + xi + x_2 = 500(1 + \text{^} + \text{^}) \text{ км.}$$

Продолжая индуктивно процесс отрабатывания назад, получаем, что *n* баков горючего позволяют нам проехать D_n км, где

$$P_n = 500(1 + i + i + \ldots + \text{^}n).$$

Нужно найти наименьшее значение *n,* при котором $D_n > 1000$. Простые вычисления показывают, что для *n — 7* имеем *Dj* = 997,5 км, т.е. семь баков, или 3500 л, топлива дадут возможность проехать 977,5 км. Полный восьмой бак — это было бы уже больше, чем нам потребуется, чтобы перевезти 3500 л из точки *A* в точку, отстоящую на 22,5 км (1000 - 977,5) от *A*. При этом для доставки 3500 л топлива к отметке 22,5 км достаточно 337,5 л. Таким образом, для того чтобы пересечь на машине пустыню из *A* в *C,* нужно 3837,5 л горючего.

Теперь алгоритм транспортировки горючего может быть представлен следующим образом. Стартуем из *A,* имея 3837,5 л. Здесь как раз достаточно топлива, чтобы постепенно, перевезти 3500 л к отметке 22,5 км, где окажемся с пустым баком и запасом горючего на семь полных заправок. Этого топлива достаточно, чтобы перевезти 3000 л к точке, отстоящей на 22,5 + 500/13 км от *A,* где бак машины будет опять пуст. Последующие перевозки приведут нас к точке, отстоящей на 22,5 + 500/13 + 500/11 км от *A,* с пустым баком машины и 2500 л на складе.

Продолжая таким образом, мы продвигаемся вперед благодаря анализу, проведенному методом отрабатывания назад. Вскоре мы окажемся у отметки 500(1 —1/3) км с 1000 л топлива. Затем перевезем 500 л в *B,* зальем их в бак машины и доедем без остановки до *C*. Рис. 11.24 иллюстрирует этот процесс.

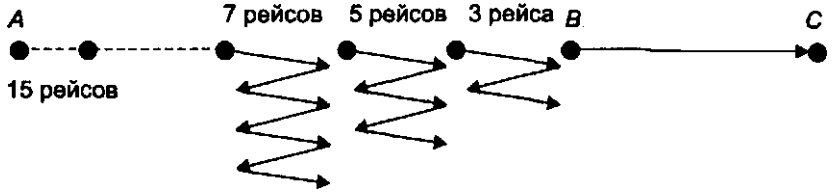


Рис. 11.24. Схема решения задачи о джипе

Возникает вопрос, можно ли проехать 1000 км, затратив меньше чем 3837,5 л горючего. Оказывается, что нельзя. Для этого можно высказать следующий, довольно правдоподобный довод. Очевидно, действуя наилучшим образом для $k = 1$, при $k = 2$ используем план для $k = 1$ и затем вводим в действие второй бак горючего для того, чтобы оказаться как можно дальше от *B*. Исходная предпосылка для *k* баков заключается в том, чтобы определить, как действовать наилучшим образом в случае с *k* — 1 баками, и отодвигаться как можно дальше назад с помощью *k-го* бака.

11.9. Задача о кодовом замке

Пусть кодовый замок состоит из набора *N* переключателей, каждый из которых может быть в положении «вкл» или «выкл». Замок открывается только при одном наборе положений переключателей, из которых не менее *[N/2]* (целая часть от *N/2),* например, находятся в положении «вкл». Предположим, что забыта комбинация, но необходимо отпереть замок. Предположим также, что можно перепробовать все комбинации, что для замка с *N* переключателями приведет к перебору 2^N возможных комбинаций. Неплохие будут шансы решить задачу полным перебором всех комбинаций, если, скажем, $N < 10$. А если значительно боль-

ше 10? Здесь и пригодится использование условия *[N/2],* которое позволит многие комбинации не просматривать. Важно лишь так построить алгоритм перебора комбинаций, чтобы не пропустить нужную и не набирать ту, которая заведомо к успеху не приведет.

Промоделируем каждую возможную комбинацию вектором из *N* нулей и единиц. На г'-м месте будет 1, если г-й переключатель находится в положении «вкл» и 0, если г-й переключатель — в положении «выкл». Множество всех возможных iV-векторов хорошо моделируется с помощью структуры, которая называется бинарное (или двоичное) дерево.

Каждая вершина А;-го уровня этого дерева будет соответствовать определенному набору первых *к* компонент iV-вектора. Две ветви, идущие вниз из вершины этого уровня, соответствуют двум возможным значениям *(к* + 1)-й компоненты в Л^-векторе. Если количество переключателей в замке равно *N,* то в дереве просмотра будет *N* уровней. На рис. 11.25 конструкция бинарного дерева изображена для $N = 4$.

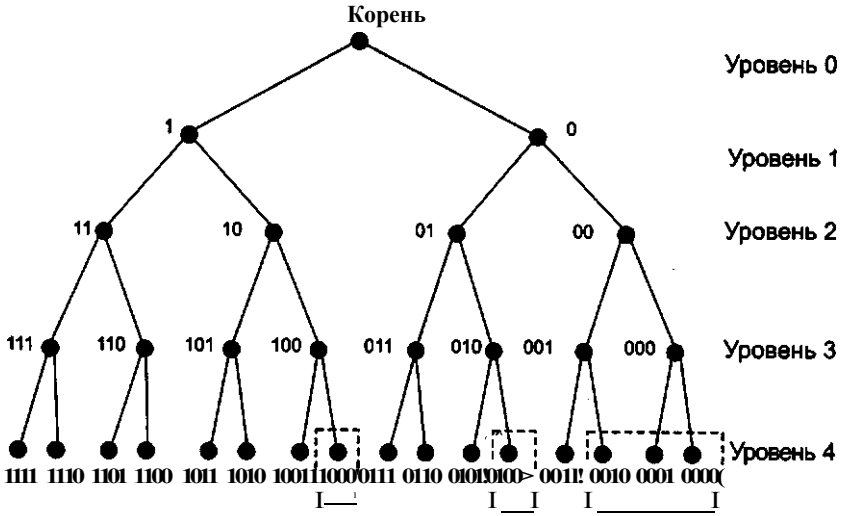


Рис. 11.25. Бинарное дерево

Условие, заключающееся в том, что число переключателей в положении «вкл» должно быть не меньше *[N/2],* позволяет не рассматривать те части дерева, которые не могут привести к правильной комбинации. Например, рассмотрим вершину 00 на рис. 11.25. Так как правая ветвь 000 не может привести к допустимой комбинации, нет необходимости ее формировать. Если какие-то вершины, следующие за рассматриваемой вершиной, не удовлетворяют ограничению задачи, то эти вершины не надо рассматривать. В данном случае вершины, находящиеся внутри пунктирных линий, не нужно исследовать и даже формировать.

Теперь, воспользовавшись этой моделью двоичного дерева, можно изложить процедуру отхода назад для образования только тех комбинаций, в которых по крайней мере *[N/2]* переключателей находятся в положении «вкл». Алгоритм сводится к просмотру дерева в определенном порядке. Движемся вниз по дереву от корня, придерживаясь левой ветви, до тех пор, пока это возможно. Достигнув конечной вершины, анализируем соответствующую комбинацию. Если она не подходит, поднимаемся на один уровень и проверяем, можно ли спуститься вниз по другой ветви.

Алгоритм заканчивает работу, когда не остается не просмотренных ветвей.

Этот пример иллюстрирует основные свойства, общие для всех алгоритмов с отходом назад. Если можно сформулировать задачу так, что все возможные решения могут быть образованы построением ./V-векторов, то ее можно решить при помощи процедуры с отходом.

Контрольные вопросы

1. Какова теоретическая сложность алгоритмов, рассмотренных в данной работе?
2. Назовите особенности работы волнового и лучевых алгоритмов?
3. Назовите особенности работы маршрутного алгоритма?
4. По какой формуле осуществляется вычисление расстояния между двумя точками?
5. Как влияет выбор приоритетов на длину трассы?
6. В чем заключаются принципы составления оптимального расписания работы параллельных процессоров?
7. Укажите основные особенности задачи упаковки?
8. Приведите принципы решения задачи о джипе?
9. Как построить дерево решений в задаче о кодовом замке?

Глава 12
Метод ветвей и границ. Задача коммивояжера *

Пусть $M = \{m_1, \ldots, m_s\}$ — конечное множество и $f: M \to R$ — вещественнозначная функция на нем; требуется найти минимум этой функции и элемент множества, на котором этот минимум достигается.

Когда имеется та или иная дополнительная информация о множестве, решение этой задачи иногда удается осуществить без полного перебора элементов всего множества M. Но чаще всего полный перебор производить приходится. В этом случае обязательно возникает задача, как лучше организовать перебор.

Метод ветвей и границ применим в том случае, когда выполняются специфические дополнительные условия на множество M и минимизируемую на нем функцию, а именно, предположим, что имеется вещественнозначная функция φ на множестве подмножеств множества M со следующими двумя свойствами:

1) для любого i $\varphi(\{m_j\}) = f(m_j)$ (здесь $\{m_j\}$ — множество, состоящее из единственного элемента m_j);

2) если $U \subset V \subset M$, то $\varphi(U) \geq \varphi(V)$.

В этих условиях можно организовать перебор элементов множества M с целью минимизации функции на этом множестве так: разобьем множество M на части (любым способом) и выберем ту из его частей $Л_i$, на которой функция φ минимальна; затем разобьем на несколько частей множество $C1\setminus$ и выберем ту из его частей **C12,** на которой функция φ минимальна; затем разобьем на несколько частей и выберем ту из них, где минимальна φ, и т.д., пока не придем к какому-либо одноэлементному множеству $\{m_i\}$.

Это одноэлементное множество $\{m_i\}$ называется *рекордом.*

Функция φ, которая используется при этом выборе, называется *оценочной.* Очевидно, что рекорд не обязан доставлять минимум функции f; однако возникает возможность сократить перебор при благоприятных обстоятельствах.

Описанный выше процесс построения рекорда состоял из последовательных этапов, на каждом из которых фиксировалось несколько множеств и выбиралось затем одно из них. Пусть $Л_1, \ldots, A_s$ —

подмножества множества M, возникшие на предпоследнем этапе построения рекорда, и пусть множество $A\setminus$ оказалось выбранным с помощью оценочной функции. Именно при разбиении $A\setminus$ и возник рекорд, который для определенности обозначим через $\{m_i\}$. Согласно сказанному выше, $\varphi(A_i) \leq \varphi(Л_i)$, $i = 1,\ldots,s$; кроме того, по определению оценочной функции, $\varphi(Ai) \leq \varphi(\{m_i\}) = f(m_i)$.

Предположим, что $f(m_i) \leq \varphi(Ar)$; тогда для любого элемента m множества M, принадлежащего множеству Ai, будут верны неравенства $f(m_i) \leq \varphi(m) \leq f(m)$; это значит, что при полном переборе элементов из M элементы из $Ач$ уже вообще не надо рассматривать. Если же неравенство $f(m_i) \leq \varphi(Ai)$ не будет выполнено, то все элементы из **A2** надо последовательно сравнить с найденным рекордом и как только отыщется элемент, дающий меньшее значение оптимизируемой функции, надо им заменить рекорд и продолжить перебор. Последнее действие называется *улучшением рекорда.*

Метод ветвей и границ связан с естественной графической интерпретацией всего изложенного: строится многоуровневое дерево, на нижнем уровне которого располагаются элементы множества M, на котором ветви ведут к рекорду и его улучшениям и на котором часть ветвей остается «отсеченной», потому что их развитие оказалось нецелесообразным.

Идея метода ветвей и границ заключается в создании некоторой процедуры построения ограниченного дерева перебора. Правда, в каких-то особых случаях «ограниченный» перебор может совпасть с полным: в этом проявляется эвристичность метода ветвей и границ.

Пусть имеется конечное множество M вариантов решения и функция F, принимающая различные значения от выбранного варианта. Требуется среди множества вариантов найти оптимальный, т.е. такой, на котором функция F принимает максимальное или минимальное (в зависимости от требований задачи) значения. Метод ветвей и границ отыскания оптимального варианта состоит из двух основных этапов:

- ветвления, т.е. построения дерева перебора;
- отсечения ветвей, т.е. прекращения построения дерева в тех ветвях, которые заведомо не содержат оптимального решения.

Рассмотрим основные принципы метода ветвей и границ на примере решения задачи расшифровки криптограмм и о радиоактивном шаре.

12.1.
Расшифровка криптограмм

Пусть дана криптограмма .

BEST + MADE = MASER.

Здесь каждая буква шифрует какую-то одну цифру из множества {О, 1, 2, 3, 4, 5, б, 7, 8, 9}. Требуется расшифровать данную криптограмму. Если воспользоваться деревом полного перебора, то можно показать, что восемь букв и десять цифр могут быть сопоставлены друг с другом различными способами, число которых достигает значения 1814400. Однако если воспользоваться определенными свойствами десятичной системы счисления, правилами сложения десятичных чисел, а также операциями ветвления и отсечения ветвей, то пространство поиска в данной задаче можно качественным образом сузить.

Для удобства рассуждений введем следующие обозначения: столбцы зашифрованных цифр пронумеруем слева направо от 1 до 5. Символы Pi, ***P2,*** P3, ***P4*** будут обозначать соответствующую цифру переноса из одного разряда в другой. Если цифра г'-го разряда больше или равна 10, то P* = 1, в противном случае P_i = 0 (рис. 12.1). Очевидно, что при сложении двух десятичных цифр величина переноса не может превышать одной единицы следующего разряда.

P4	P3	P2	P1		**Переносы**
	B	***E***	S	***T***	
	м	***A***	***D***	***E***	
м	***A***	***S***	***E***	***R***	
5	4	3	***2***	1	**Номера разрядов**

Рис. 12.1. Задача о криптограмме

Начнем анализ с пятого разряда. Можно уверенно утверждать, что *M* = 1, так как *M* не может равняться нулю, поскольку не принято писать нуль в начальной позиции, также очевидно, что *M* не может быть больше единицы. Так как в пятом разряде *M* целиком образована значением переноса P4, то получаем *M* = 1. Определен корень ограниченного дерева поиска (рис. 12.2, *а).*

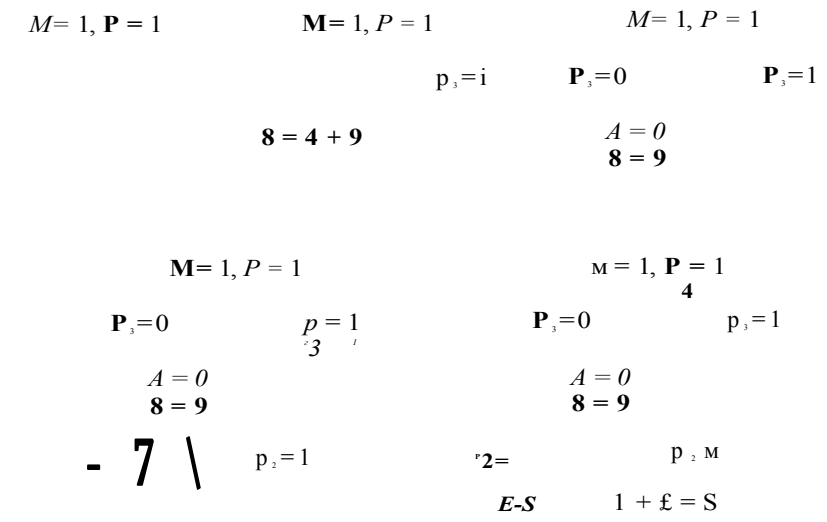


Рис. 12.2. Этапы построения дерева перебора для расшифровки криптограммы

Обращаясь к четвертому разряду, нельзя с уверенностью сказать, чему равно P3 — единице или нулю. Отметим оба варианта на дереве поиска (рис. 12.2,б). Левая ветвь: Рз = 0, так как P4 = 1 и P_3 = 0, то *B* + *M* должно быть равно *A* + 10. Но из-за того, что *M* = 1, получается, что *B* = *A* + 9. Но В не может быть больше 9, поэтому *A* = 0 (рис. 12.2, в). Теперь проанализируем фрагмент дерева, изображенный на рис. 12.2, г.

Если рассматривать ветвь Рг = 0, то нетрудно видеть, что Рг + *E* = = *S.* Поскольку Рг = 0, *A* = 0, то имеем *E* = *S,* что запрещено правилами построения криптограммы. Следовательно, путь Рз = 0, P_2 = 0 тупиковый, его дальше продолжать не следует. Если же двигаться по ветви P2 = 1, получаем 1 + *E* = *S,* и этот путь, возможно, следует просматривать далее (рис. 12.2, д). Продолжая аналогичное построение, получим ограниченное дерево поиска для данной крип-

тографической задачи (рис. 12.3). Звездочками обозначены ветви, не имеющие продолжения.

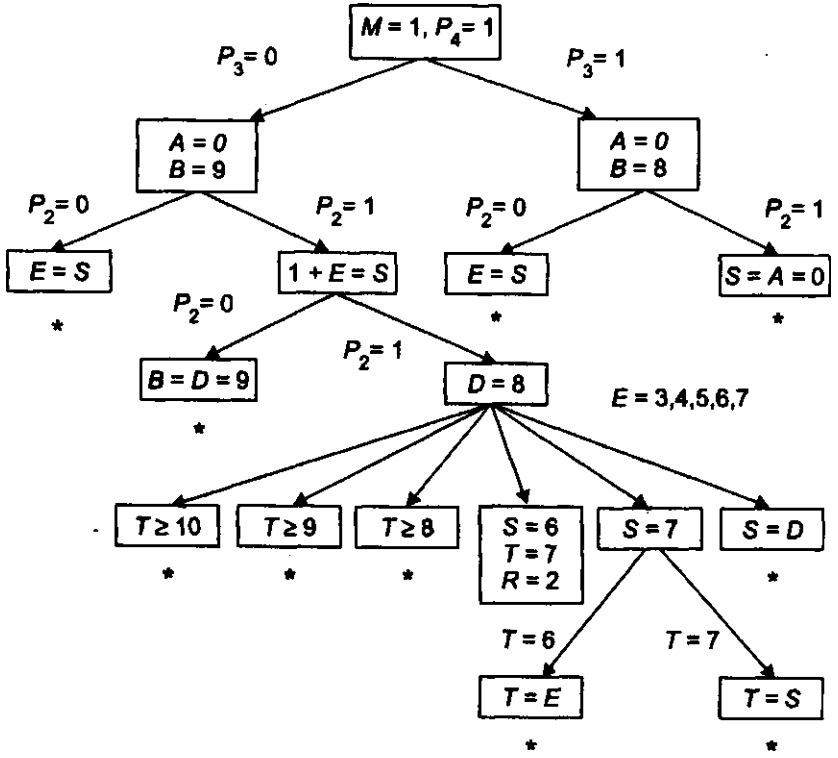


Рис. 12.3. Ограниченное дерево перебора для расшифровки криптограммы

Следует отметить, что сначала в качестве принципа ветвления использовался признак наличия или отсутствия переноса, а затем, когда этот признак был исчерпан, пришлось сделать полный перебор по всем возможным значениям буквы *E* (перечислены на рисунке слева направо). Таким образом, вместо полного дерева перебора, содержащего около 1814400 листьев, удается построить ограниченное дерево, рассмотрение которого позволяет легко найти нужное решение. В результате можно сделать следующий вывод. Многие комбинаторные задачи существенно упрощаются, если вместо комбинаторно полного дерева перебора удается построить ограниченное дерево, обязательно включающее пространство, содержащее решение.

12.2. Задача о радиоактивном шаре

Рассмотрим задачу об отыскании одного радиоактивного шарика среди ***n*** одинаковых шариков. В качестве прибора, определяющего радиоактивность, используется дозиметр, два различных показания которого (есть радиоактивность или нет радиоактивности) естественно принять за значения оценочной функции, например, значения 0 или 1. В данном случае точное значение оценочной функции определено на любом подмножестве шариков. Всякое подмножество, для которого значение оценочной функции равно нулю, исключается из дальнейшего рассмотрения (помечено *).

На рис. 12.4 приведено ограниченное дерево перебора для ***n*** = 100, из которого видно, что количество испытаний (число 7) удовлетворяет оценкам. Сравним: в алгоритме полного перебора в худшем случае пришлось бы проверить 100 шариков, т.е. осуществить 100 проб. Для ограниченного дерева таких проверок потребуется только 7.

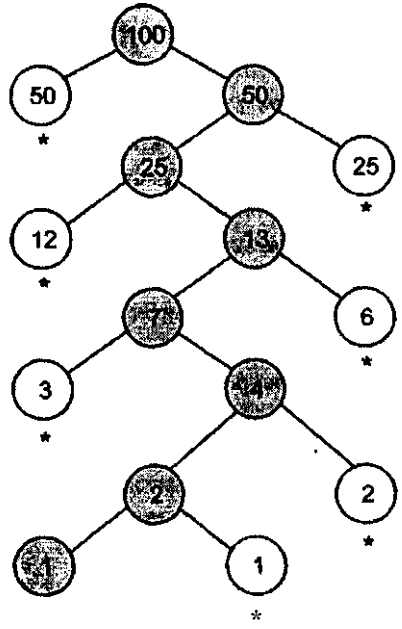


Рис. 12.4. Ограниченное дерево перебора

12.3. Задача коммивояжера

Имеется несколько городов, соединенных дорогами с известной длиной; требуется установить, имеется ли путь, двигаясь по которому можно побывать в каждом городе только один раз и при этом вернуться в город, откуда путь был начат («обход коммивояжера»), и, если такой путь имеется, установить кратчайший из них.

Формализуем условие в терминах теории графов. Города будут вершинами графа, а дороги между городами — ориентированными (направленными) ребрами графа, на каждом из которых задана весовая функция: вес ребра—это длина соответствующей дороги. Путь, который требуется найти, — это ориентированный остовный простой цикл минимального веса в ориентированном графе (орграфе). (Напомним: цикл называется *остовным,* если он проходит по всем вершинам графа; цикл называется *простым,* если он проходит по каждой своей вершине только один раз; цикл называется *ориентированным,* если начало каждого последующего ребра совпадает с концом предыдущего; *вес* цикла—это сумма весов его ребер; наконец, орграф называется *полным,* если в нем имеются все возможные ребра); такие циклы называются также *гамильтоновыми.*

Очевидно, в полном орграфе циклы указанного выше типа есть. Заметим, что вопрос о наличии в орграфе гамильтонова цикла достаточно рассмотреть как частный случай задачи о коммивояжере для полных орграфов. Действительно, если данный орграф не является полным, то его можно дополнить до полного недостающими ребрами и каждому из добавленных ребер приписать вес оо, считая, что оо — это «компьютерная бесконечность», т.е. максимальное из всех возможных чисел. Если во вновь построенном полном орграфе найти теперь легчайший гамильтонов цикл, то при наличии у него ребер с весом оо можно будет говорить, что в данном, исходном графе «цикла коммивояжера» нет. Если же в полном орграфе легчайший гамильтонов цикл окажется конечным по весу, то он и будет искомым циклом в исходном графе.

Отсюда следует, что задачу коммивояжера достаточно решить для полных орграфов с весовой функцией. Сформулируем теперь это в окончательном виде: пусть $G = (A, B)$ — полный ориентированный

граф и $v: B \to SK$ — весовая функция; найти простой остовный ориентированный цикл («цикл коммивояжера») минимального веса.

Пусть $A = \{ai, \ldots, ap\}$ — конкретный состав множества вершин и M = *(mij), i,j = l,...,p* — весовая матрица данного орграфа, т.е. *rriij --- v(ai,a,j),* причем для любого *i mц* = оо.

Рассмотрим использование метода ветвей и границ для решения задачи коммивояжера.

Шаг 1. Фиксируем множество всех обходов коммивояжера (т.е. всех простых ориентированных остовных циклов). Поскольку граф— полный, это множество заведомо непустое. Сопоставим ему число, которое будет играть роль значения на этом множестве оценочной функции: это число равно сумме констант приведения данной матрицы весов ребер графа. Если множество всех обходов коммивояжера обозначить через Г, то сумму констант приведения матрицы весов обозначим через *<p(T).* Приведенную матрицу весов данного графа следует запомнить; обозначим ее через *M* таким образом, итог первого шага: множеству Г всех обходов коммивояжера сопоставлено число *ip(T)* и матрица *M\.*

Шаг 2. Выберем в матрице *M* самый тяжелый ноль (функцию штрафа); пусть он стоит в клетке фиксируем ребро графа и разделим множество Г на две части: на часть **JJ**, состоящую из обходов, которые проходят через ребро и на часть состоящую из обходов, которые не проходят через ребро Сопоставим множеству Г(^) следующую матрицу *Mi*: в матрице *M* заменим на оо число в клетке Затем в полученной матрице вычеркнем строку номер г и столбец номер *j,* причем у оставшихся строк и столбцов сохраним их исходные номера. Наконец, приведем эту последнюю матрицу и запомним сумму констант приведения. Полученная приведенная матрица и будет матрицей Мхд. Только что запомненную сумму констант приведения прибавим к </?(Г) и результат, обозначаемый в дальнейшем через *(p(T^),* сопоставим множеству Г^).

Теперь множеству тоже сопоставим некую матрицу Mi,2- Для этого в матрице *Mi* заменим на оо число в клетке и полученную в результате матрицу приведем. Сумму констант приведения ; запомним, а полученную матрицу обозначим через *Mi*,2. Прибавим запомненную сумму констант приведения к числу у(Г), и полученное число, обозначаемое в дальнейшем через *fi^JJj))'* сопоставим множеству

Выберем между множествами Г(^) и Г^у то множество, на котором минимальна функция <? (т.е. то из множеств, которому соответствует меньшее из чисел <p(Г(^) и <^(Г^у).

Заметим теперь, что в проведенных рассуждениях использовался в качестве исходного только один фактический объект — приведенная матрица весов данного орграфа. По ней было выделено определенное ребро графа и были построены новые матрицы, к которым, конечно, можно все то же самое применить.

При каждом таком повторном применении будет фиксироваться очередное ребро графа. Условимся о следующем действии: перед тем, как в очередной матрице вычеркнуть строку и столбец, в ней надо заменить на оо числа во всех тех клетках, которые соответствуют ребрам, заведомо не принадлежащим тем гамильтоновым циклам, которые проходят через уже отобранные ранее ребра.

К выбранному множеству с сопоставленными ему матрицей и числом *ip* применим до тех пор, пока это возможно, указанный выше алгоритм.

Доказывается, что в результате получится множество, состоящее из единственного обхода коммивояжера, вес которого равен очередному значению функции <p; таким образом, оказываются выполненными все условия, обсуждавшиеся при описании метода ветвей и границ.

После этого осуществляется улучшение рекорда вплоть до получения окончательного ответа.

Жадный алгоритм. Найти приближенное значение кратчайшего тура *(TOUR)* со стоимостью *COST* для задачи коммивояжера с *N* городами и матрицей стоимости *C,* начиная с вершины *U.*

Жадный алгоритм — алгоритм нахождения кратчайшего расстояния путем выбора самого короткого, еще не выбранного ребра, при условии, что оно не образует цикла с уже выбранными ребрами. Алгоритм использует стратегию «иди в ближайший город». «Жадным» этот алгоритм назван потому, что на последних шагах приходится жестоко расплачиваться за жадность.

Ш а г 1 (Инициализация). *TOUR* := 0; *COST* := 0. Пометить вершину *U* как «выбранную», т.е. *V := U,* где *V* — текущая переменная, а все другие вершины — как «не выбранные».

Ш а г 2 (Посещение всех городов). FOR г := 1 TO *N* - 1 DO.

Ш а г 3 (Выбор следующего ребра). Пусть *(V,W)* — ребро с наименьшей стоимостью, ведущее из У в любую «невыбранную» вершину *W. TOUR:=TOUR+(V,W);COST:=COST + C(V,W).* Помечаем *W* как «выбранную», т.е. *V := W.*

Ш а г 4 (Завершение тура). *TOUR := TOUR + (V, U); COST := COST + C(V,U).*

На рис. 12.5 проиллюстрирована последовательность выполнения данного алгоритма для графа, имеющего пять вершин (тур начинается с вершины 1). Пройденные вершины обозначены черным квадратиком, а не пройденные — кружком. Жадный алгоритм для данного графа нашел тур со стоимостью 14, хотя оптимальный тур имеет стоимость 13. Ясно, что «грубый алгоритм» применительно к задаче коммивояжера не всегда находит тур с минимальной стоимостью.

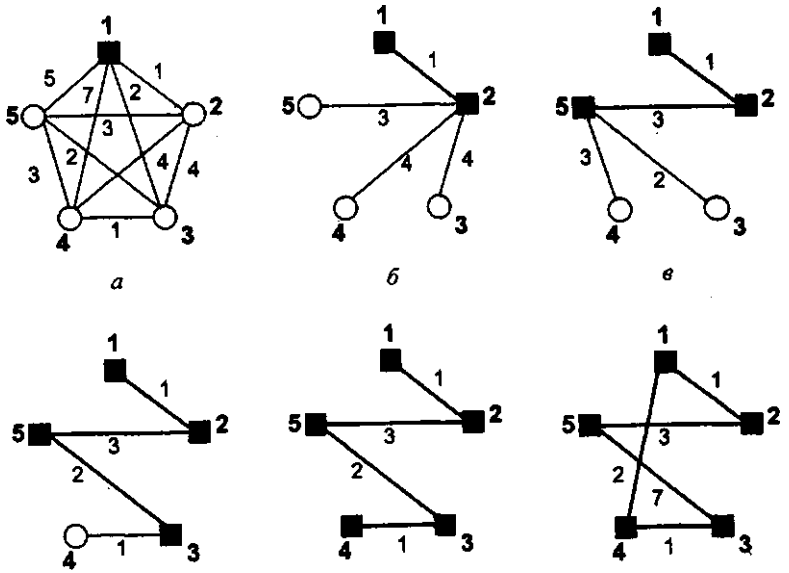


Рис. 12.5. Иллюстрация работы жадного алгоритма:
а — стоимость = 0; *б* — стоимость = 1; в — стоимость = 4;
г — стоимость = 6; *д* — стоимость = 7; е — стоимость = 14

Жадный алгоритм основан на идее подъема. Цель — найти тур с минимальной стоимостью. Задача сведена к набору частных целей — найти на каждом шаге «самый дешевый» город, чтобы посетить

его следующим. Алгоритм не строит плана вперед^текущий выбор делается безотносительно к следующим выборам.

Для произвольной задачи коммивояжера с *n* горЪдами требуется *0(n²)* операций, чтобы обработать матрицу стоимостей *С*. Поэтому нижняя граница сложности любого алгоритма, спо| обного дать нетривиальное возможное решение этой задачи, равна с >(^²)- Нетрудно проверить, что для любой разумной реализации umrj **IB** 1-3 требуется не больше чем 0(п²) операций. **j**

Деревянный алгоритм. В основе деревянного г лгоритма лежит построение остовного дерева, которое, с учетом эврис гических подходов, преобразуется в тур коммивояжера.

Рассмотрим этапы решения задачи коммивояж! ра деревянным алгоритмом. I

Шаг 1. Построим на входной сети кратчайшее пстовное дерево и удвоим все его ребра. Получим связный граф *С'* с вершинами, имеющими только четные степени.

Шаг 2. Построим эйлеров цикл *G,* начиная с *М* эшины 1. Цикл задается перечнем вершин.

Шаг 3. Рассмотрим перечень вершин, начина! с 1, и будем вычеркивать каждую вершину, которая повторяет уж! встреченную в последовательности. Останется тур, который и являф-ся результатом работы алгоритма.

Пример. Дана полная сеть, заданная следующей матрицей смежности:

Вершины	1	2	3	4	5		6
1		6	4	8	7		14
2	6		7	11	7		10
3	4	7		4	3		10
4	8	11	4		5		11
5	7	7	3	5		I	7
6	14	10	10	11	7		

Найти тур коммивояжера жадным и деревянным алго| итмами.

Решение. Если справедливо неравенство треуголЪника, то

$$41,3]<d[1,2]+d[2,3] \quad \text{и} \quad d[3,5]<d[3,4]+\mathbf{44,5].}$$

Сложив эти два неравенства, получим

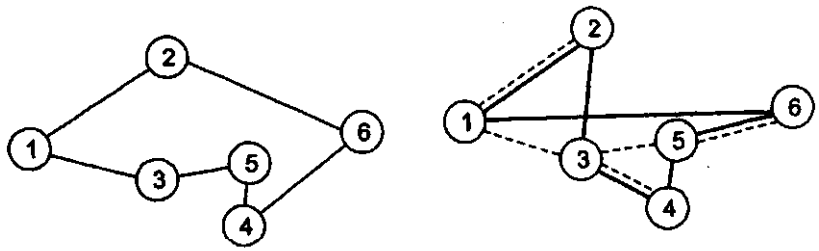
$$d[l,3]+d[3,5]<d[l,2]+d[2,3]+d[3,4]+d\backslash 4,5].$$

По неравенству треугольника получим, $d[1,5] < d[l,2] + d[3,5]$. Окончательно имеем

$$\mathbf{41,}5] < d[l,\ 2] + d[2,3] + d[3,4] + d[4,5].$$

Итак, если справедливо неравенство треугольника, то для каждой цепи верно, что расстояние от начала до конца цепи меньше (или равно) суммарной длине всех ребер цепи. Это обобщение расхожего убеждения, что прямая короче кривой.

Жадный алгоритм (иди в ближайший город) дает тур **1** — (4) - 3 - (3) - 5 - (5) - 4 - **(11)** - 6 - **(10)** - 2 - (6) -**1,** где без скобок показаны номера вершин, а в скобках — длины ребер. Длина тура равна 39, тур показан на рис. 12.6, а.



б

Рис. 12.6. Гамильтонов цикл, построенный жадным *(а)* и деревянным (б) алгоритмами

Деревянный алгоритм вначале строит остовное дерево штриховой линией, затем эйлеров цикл 1 — 2 - 1 - 3 - 4 - 3 — 5 — 6 — 5 — 3 - 1, затем тур 1 — 2- 3 — 4 — 5- 6 - 1 длиной 43, который показан сплошной линией на рис. 12.6,6.

12.4. Примеры решения задачи коммивояжера

Пример 1. Решить методом ветвей и границ задачу коммивояжера для графа, содержащего 6 вершин. Пусть исходный ориентированный граф задан матрицей стоимости:

	1	2	3	4	5	6
1	oo	68	73	24	70	9
2	58	oo	16	44	11	92
3	63	9	oo	86	13	18
4	17	34	76	oo	52	70
5	60	18	3	45	co	58
6	16	82	11	60	48	oo

Начальное приведение матрицы стоимости. Пусть имеется некоторая числовая матрица. *Привести строку* этой матрицы означает выделить в строке минимальный элемент (его называют *константой приведения)* и вычесть его из всех элементов этой строки. В результате в этой строке на месте минимального элемента окажется нуль, а все остальные элементы будут неотрицательными. Матрица стоимости называется *приведенной,* если она имеет в каждой строке и каждом столбце хотя бы один нуль. Сумма констант приведения образует нижнюю граничную оценку стоимости любого возможного тура.

Вычисление функции штрафа. Функция штрафа — это множество чисел, вычисленных для каждого нуля приведенной матрицы посредством суммирования двух минимальных чисел из той строки и из того столбца, в которых расположен нулевой элемент.

Выбор ребра ветвления. Для ветвления необходимо выбирать ребро, которому соответствует максимальная функция штрафа. Если существует несколько одинаковых максимальных значений функции штрафа, выбор среди них может быть произвольным.

Весом элемента матрицы называют сумму констант приведения матрицы, которая получается из данной матрицы заменой анализируемого элемента на oo. Следовательно, выражение *самый тяжелый нуль* в матрице означает, что в матрице подсчитан вес каждого нуля, а затем фиксирован нуль с максимальным весом.

Вычисление граничной оценки для ветви, соответствующей не включению ребра в тур. Эта оценка вычисляется как сумма граничной оценки, соответствующей предыдущему узлу дерева перебора, и выбранному значению функции штрафа.

Вычисление граничной оценки для ветви, соответствующей включению ребра в тур. Для вычисления граничной оценки необходимо:

• вычеркнуть в матрице стоимости строку и столбец, соответствующие выбранному ребру;

• скорректировать полученную матрицу таким образом, чтобы устранить возможность досрочного завершения тура (устранить циклы);

• сделать приведение (если необходимо) полученной матрицы: если константа приведения отлична от нуля, сложить эту константу с граничной оценкой предыдущего узла.

Проверка на окончание решения. Если скорректированная матрица имеет размер 2 x 2, а узел дерева, которому соответствует 3 x 3 матрица, имеет минимальную граничную оценку, то решение задачи заканчивается: два оставшихся нуля этой матрицы соответствуют двум последним ребрам, которые включаются в тур непосредственно, при этом стоимость тура не изменяется.

Решение. Сначала приводим исходную матрицу по строкам (матрица ***Cic):***

	1	2	3	4	5	6	***hi***
1	oo	59	64	15	61	0	9
2	47	**00**	5	33	0	81	11
3	54	0	**00**	77	4	9	9
4	0	17	59	oo	35	53	17
5	57	15	0	42	co	55	3
6	5	71	0	49	37	co	11

В последнем столбце этой матрицы *1ц* записаны константы приведения по строкам.

Полученную матрицу необходимо привести по столбцам. Результате получаем матрицу ***Cj,*** в которой в строке ***hj*** записаны константы приведения по столбцам:

	1	2	3	4	5	6
1	**00**	59	64	0	61	0
2	47	oo	5	18	0	81
3	54	0	oo	62	4	9
4	0	17	59	co	35	53
5	57	15	0	27	oo	55
6	5	71	0	34	37	co
hj	0	0	0	15	0	0

Сумма констант приведения ***H*** дает нижнюю граничную оценку стоимости всех туров:

=]Г Л* + **J**^/i.,- = 9 + 11 + 9 +17 + 3 + 11 + 15 = 75.
i *j*

Вычисляем для ребер, помеченных нулем в матрице *C[,* значения функций штрафа, которые обозначим символом *Dif*

£>16 = 0 + 9 = 9; D₄**i** = 1 7 + 5 = 22;

£>₂₅ = 5 + 4 = 9; **E>53** = 15 + 0=15;

#₃₂ = 4 +15 = 19; £>₆з = 5 + 0 = 5.

Максимальное значение функции штрафа равно 22, на основании чего в качестве ребра ветвления выбираем ребро (4,1). Разбиваем множество всех туров *R* на два подмножества: {4, 1} — подмножество всех туров, в которое входит ребро (4,1), и подмножество всех туров {4, 1*}, в которое ребро (4,1) не входит. Строим первый фрагмент ограниченного дерева пере- г-

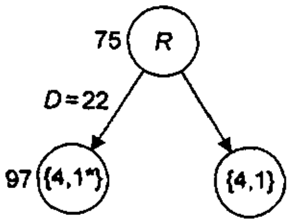


Рис. 12.7. Первый фрагмент ограниченного дерева перебора

⁶ ° Р ᵃ 0>ⁿᶜ- ¹²-⁷>- ⁿ Р ᴵᴵ ᵌ ᵀ ᵒ ᴹ граничная оценка в ветви {4, 1*} находится непосредственно как сумма 75 + 22 = 97.

Вычеркиваем в матрице *C[* четвертую строку и первый столбец, а чтобы запретить досрочное завершение тура по ребру графа (1,4), весу этого ребра присвоить значение оо. В результате подобной корректировки получаем матрицу

	2	3	4	5	6
1	59	64	oo	61	0
2	oo	5	18	0	81
3	0	co	62	4	9
5	15	0	27	oo	55
6	71	0	34	37	oo
hj	0	0	18	0	0

Ее необходимо привести по четвертому столбцу (константа приведения / i₄ = 18), в результате чего получаем приведенную матрицу *C'₂*

и вычисляем граничную оценку в ветви {4, 1}, которая будет равна 75 +18 = 93 (рис. 12.8).

Сравниваем оценки в листьях и приходим к выводу, что ветвление целесообразно продолжать в листе с оценкой 93. Так как этому листу соответствует матрица *C'₂*, то с ней необходимо поступать как с исходной матрицей, т.е. вычислять для нее значения функций штрафа, выбирать максимальное значение штрафа и на его основе выбирать следующее ребро ветвления:

97, 93

Я П

Рис. 12.8. Первый фрагмент ограниченного дерева перебора с граничными оценками в листьях

C₂ =

	2	3	4	5	6
1	59	64	oo	61	0
2	oo	5	0	0	81
3	0	oo	44	4	9
5	15	0	9	00	55
6	71	0	16	37	oo

Для матрицы C₂ имеем следующие значения функций штрафа:

#₁₆ = 59 + 9 = 68; 1)**32** = 4 + 15 = 19;

#24 =0 + 9 = 9; **#53** = 9 + 0 = 9;

£>25 = 0 + 4 = 4; # **з**₂ = 16 + 0 = 16.

Выбираем максимальное значение 68, которому соответствует ребро ветвления (1,6), после чего можно построить второй фрагмент ограниченного дерева перебора (рис. 12.9).

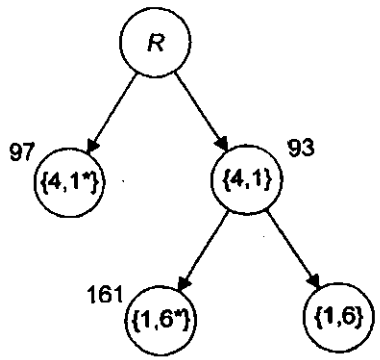


Рис. 12.9. Второй фрагмент ограниченного дерева перебора

Затем приступаем к вычислению граничной оценки в узле дерева {1, 6}. Для этого вычеркиваем в матрице ***C₂*** первую строку и шестой столбец, а также предпринимаем меры для исключения досрочного завершения тура матрицы C₃.

Рис. **12.10.** Исключение досрочного завершения тура

В самом деле, к данному моменту в тур включено ребро (4,1) и есть предложение включить в него ребро (1,6). Из рис. 12.10 видно, что досрочно завершить тур могли бы ребра (6,1) и (6,4). Однако элемент (6,1) в матрице Сз уже отсутствует, следовательно, эту «роль» может выполнить лишь элемент (6,4), поэтому заменяем его символом со. После этого окончательно матрица Сз будет иметьвид:

$C_3 = C_3 =$

	2	3	4	5
2	∞	5	0	0
3	0	∞	44	4
5	15	0	9	∞
6	71	0	со	37

Матрица Сз оказалась приведенной (поэтому ее можно, как это было ранее принято для приведенных матриц, пометить штрихом), а следовательно, оценка в узле (1,6) остается неизменной (рис. 12.11).

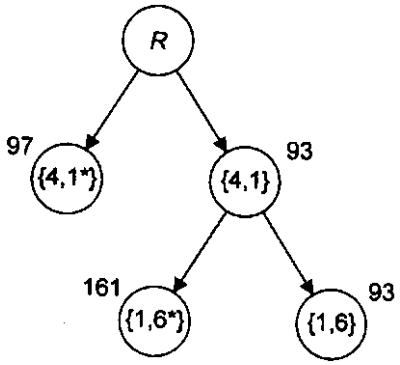


Рис. **12.11.** Второй фрагмент ограниченного дерева перебора с граничными оценками в листьях

Вычисляем значения функций штрафа для матрицы C_3:

$1>24 = 0 + 9 = 9;$

$£25 = 0 + 4 = 4;$

$D_{3\,2} = 4 + 15 = 19;$

$A>з = 9 + 0 = 9;$

$£>63 = 37 + 0 = 37.$

На основе вычисленных значении определяем следующее ребро ветвления (6, 3) и вычисляем граничную оценку в том узле дерева перебора, которому соответствуют туры, не содержащие ребра (6,3) (рис. 12.12). При этом ветвление продолжается от узла с минимальной оценкой, равной 93.

Рис. **12.12.** Третий фрагмент ограниченного дерева перебора

Вычеркиваем в матрице C'_3 шестую строку и третий столбец и корректируем ее на исключение досрочного завершения тура. Это достигается присваиванием символа бесконечности элементу (3,4). После этого получаем матрицу Ci, а после ее приведения — матрицу C'_4:

$C_4 =$

	2	4	5
2	со	0	0
3	0	сю	4
5	15	9	∞

$c'_4 =$

	2	4	5
2	∞	0	0
3	0	∞	4
5	6	0	∞

Так как матрица С4 не была приведена, то граничная оценка в узле {6, 3} увеличивается по сравнению с предыдущим узлом ветвления на 9 и становится равной 102. Третий фрагмент ограниченного дерева перебора с граничными оценками в листьях будет иметь вид, изображенный на рис. 12.13.

Рассмотрение дерева, приведенного на рис. 12.13, показывает, что ветвление нужно продолжать из узла {4, 1*}, так как граничная оценка там меньше (97), чем в узле, из которого ветвление осуществлялось до сих пор. Подобная ситуация называется *перескоком,* и в этом случае может оказаться, что все полученные до сих пор данные будут утеряны.

Рис. 12.13. Третий фрагмент ограниченного дерева перебора с граничными оценками в листьях

Итак, переходим к узлу {4, 1*}, которому соответствует матрица

	I	2	3	4	5	6
1	∞	68	73	24	70	9
2	58	∞	16	44	11	92
3	63	9	∞	86	13	18
4	∞	34	76	oo	52	70
5	60	18	3	45	oo	58
6	16	82	11	60	48	oo

Рис. 11. Матрица C_5, приведенная после выполнения операции приведения. Константы приведения

получаем матрицу C'_5:

1	1	2	3	4	5	6
1	oo	59	64	0	61	0
2	42	∞	5	18	0	81
3	49	0	oo	62	4	9
4	oo	0	42	∞	18	36
5	52	15	0	27	oo	55
6	0	71	0	34	37	oo

У матрицы C_5 сумма констант приведения будет равна 0+ 42 = 42. С учетом вычисленной

оценке в узле {4, 1*}. Этот факт подтверждает правильность выполненных вычислений.

Вычисляем для матрицы C_5 значения функций штрафа:

$$D_{14} = 0 + 18 = 18; \qquad \text{£>42} = 18 + 0 = 18;$$
$$D_{15} = 0 + 9 = 9; \qquad \text{£>33} = 15 + 0 = 15;$$
$$D_{25} = 5 + 4 = 9; \qquad \text{-Pel} = 0 + 42 = 42;$$
$$D_{32} = 4 + 0 = 4; \qquad \text{#63} = 0 + 0 = 0.$$

Очередным ребром ветвления из узла {4, 1*} будет ребро (6,1), так как ему соответствует максимальное значение функции штрафа. После корректировки матрицы C'_5 (ребру (1, 6) приписывается бесконечный вес) и вычисления граничных оценок окончательный четвертый фрагмент дерева перебора будет иметь вид, представленный на рис. 12.14.

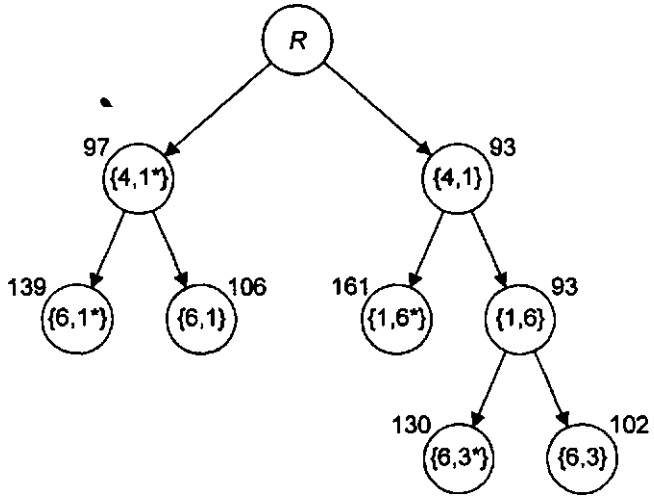


Рис. 12.14. Четвертый фрагмент ограниченного дерева перебора с граничными оценками в листьях

Из этого рисунка видно, что минимальная граничная оценка (102) имеет место в узле {6, 3}, поэтому целесообразно осуществить обратный «перескок» в этот узел и продолжать дальнейшее ветвление из него. Этому узлу соответствует ранее вычисленная матрица C'_4; дадим ей для удобства сквозной нумерации новое обозначение $Ce = C_6$.

Вычислим для матрицы C_6 значения функций штрафа:

$$\text{1>24} = 0 + 0 = 0; \qquad £>_{32} = 4 + 6 = 10;$$
$$\text{I>25} = 0 + 4 = 4; \qquad Z>_{51} = 6 + 0 = 6.$$

Максимальное значение функции штрафа, равное 10, связано с ребром (3,2), поэтому выбираем его в качестве ребра ветвления, продолжая построение дерева из листа с граничной оценкой 102.

После исключения из матрицы C'_6 третьей строки, второго столбца, а также ее корректировки, получаем приведенную матрицу *Ст.*

	4	5
2	oo	0
5	0	oo

Так как матрица CV уже приведена и имеет размер 2 х 2, то ребра (2,5) и (5,4) непосредственно включаем в тур, при этом граничная оценка 102 не изменяется.

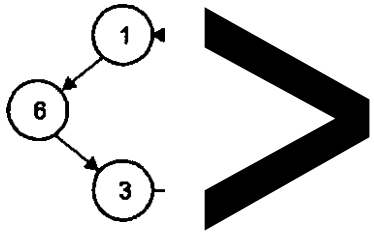


Рис. 12.15. Оптимальный тур

Окончательный вид тура представлен на рис. 12.15, а ограниченное дерево перебора — на рис. 12.16. Легко проверить, что сумма весов ребер, вошедших в тур, равна 102, что подтверждает правильность полученного решения.

Пример 2. Решить задачу коммивояжера, используя алгоритм Литтла, для графа из 6 вершин, заданного матрицей смежности

	1	2	3	4	5	6
1		6	4	8	7	14
2	6	–,	7	11	7	10
3	4	7		4	3	10
4	8	11	4	–	5	11
5	7	7	3	5		7
6	14	10	10	11	7	

Общая идея метода: нужно разделить огромное число перебираемых вариантов на классы и получить оценки (снизу — в задаче

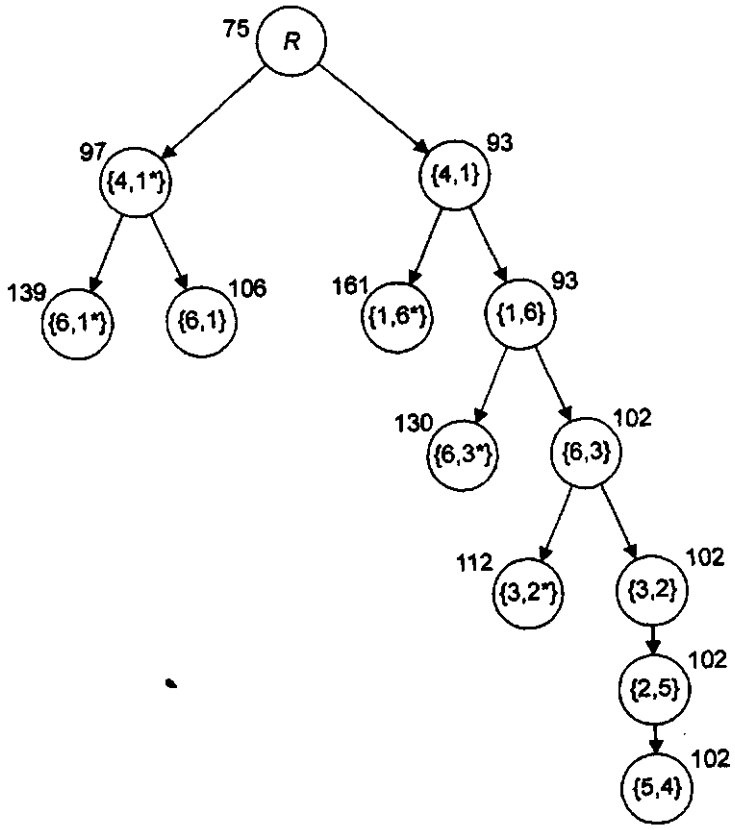


Рис. 12.16. Окончательный вид ограниченного дерева перебора

минимизации, сверху — в задаче максимизации) для этих классов, чтобы иметь возможность отбрасывать варианты не по одному, а целыми классами. Трудность состоит в том, чтобы найти такое разделение на классы (ветви) и такие оценки (границы), чтобы процедура была эффективной.

Будем трактовать Су как стоимость проезда из города г в город *j*. Допустим, что добрый мэр города *j* издал указ выплачивать каждому въехавшему в город коммивояжеру 5 долл. Это означает, что любой тур подешевеет на 5 долл., поскольку в любом туре нужно въехать в город *j*. Но поскольку все туры равномерно подешевели, то прежний минимальный тур будет и теперь стоить меньше всех. Добрый

же поступок мэра можно представить как уменьшение всех чисел *j-ro* столбца матрицы *C* на 5. Если бы мэр хотел спровадить коммивояжеров из *j-ro* города и установил награду за выезд я размере 10 долл., это можно было бы выразить вычитанием 10 из всех элементов j - й строки. Это снова изменило бы стоимость каждого тура, но минимальный тур остался бы минимальным. Итак, доказана следующая лемма: *вычитая любую константу из всех элементов любой строки или столбца матрицы C, оставляем тур минимальным.*

Для алгоритма удобно получить побольше нулей в матрице *C,* не имея в ней отрицательных чисел. Для этого вычтем из каждой строки ее минимальный элемент (это называется приведением по строкам):

$Ci =$

	1	2	3	4	5	е	*Ы*
1		2	0	4	3	10	4
2	$\overline{0}$		1	5	1	*4*	6
3	1	4	–	1	0	7	3
4	4	7	0	–	1	7	4
5	4	4	0	2	.–	*4*	3
6	7	3	3	4	0	~*ii*	7

IMI2UIT*°тм столбца полу ченной мэт **р 4** в тм его минимальный элемент, получив матрицу, приведеннук! по столбцам:

$C_2 =$

	1	2	3	4	5	6
1		0	0	3	3	6
2	$\overline{0}$	–	1	4	1	0
3	1	2	–	0	0	3
4	4	5	0	–	1	3
5	4	2	0	1	у'ii»'':.	0
б	7	1	3	3	0	
	0	2	0	1	0	4

Прочерки по диагонали означают, что из города | в город ***г*** проехать нельзя. Заметим, что сумма констант приведения гао строкам равна 27, сумма по столбцам 7,

я=]Г/ц+**53** л . * = ²⁷ + ⁷ = ³¹

* *з* *1*

Для тура из шести городов выделенных элементов, включенных в тур, должно быть шесть, так как в туре из шести городов есть шесть ребер. Каждый столбец должен содержать ровно один выделенный элемент (в каждый город коммивояжер въехал один раз), в каждой строке должен быть один такой элемент (из каждого города коммивояжер выехал один раз). Кроме того, выделенные элементы должны описывать один тур, а сумма чисел выделенных элементов есть стоимость тура.

В приведенной матрице необходимо построить правильную систему выделенных элементов, т.е. систему, удовлетворяющую вышеописанным требованиям, и эти выделенные элементы будут обозначены нулями, тогда для этой матрицы получим минимальный тур. Таким образом, минимальный тур не может быть меньше 34 (оценка снизу для всех туров).

Теперь приступим к ветвлению, оценивая нулевые элементы. Рассмотрим нуль в клетке (1, 2) приведенной матрицы *Сг*- Он означает, что цена перехода из города 1 в город 2 равна 0. А если коммивояжер не поедет из города 1 в город 2? Тогда все равно нужно въехать в город 2 за цены, указанные во втором столбце; дешевле всего за 1 (из города 6). Далее, все равно надо будет выехать из города 1 за цену, указанную^ первой строке; дешевле всего в город 3 за 0. Суммируя эти два минимума, имеем 1 + 0 = 1: если не ехать «по нулю» из города 1 в город 2, то надо заплатить не меньше 1. Таким образом вычисляется функция штрафа. Оценки всех нулей показаны в матрице Сз правее и выше нуля (оценки, равные нулю, не указывались):

	1	2	3	4	5	6
1	–	O^1	0	3	3	6
2	O^1		1	4	1	0
3	1	2	_ <	O^1	0	3
4	4	5	O^1	–	1	3
5	4	2	0	1		0
6	7	1	3	3	O^1	

Выберем максимальную из этих оценок (в примере есть несколько оценок, равных единице, выберем первую из них в клетке (1, 2)).

Итак, выбрано нулевое ребро (1,2). Разобьем все туры на два множества— включающие ребро (1,2) и не включающие ребро (1,2).

В первом случае необходимо рассмотреть матрицу C4 с вычеркнутой первой строкой и вторым столбцом:

$C_4 =$

	1	3	4	5	6
2	O¹	1	4	1	0
3	1	.–	O¹	0	3
4	4	O¹	–	1	3
5	4	0	1	-г..'	0
6	7	3	3	O¹	

Дополнительно в уменьшенной матрице *C* поставлен запрет в клетке (2, 1), так как выбрано ребро (1, 2) и замыкать преждевременно тур ребром (2,1) нельзя.

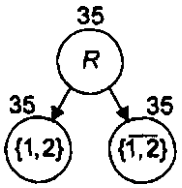


Рис 12.17. Фрагмент дерева перебора решений

Уменьшенную матрицу можно привести на 1 по первому столбцу, при этом тур, отвечающий ей, будет стоить 35. Результат ветвлений и получения оценок показан на рис. 12.17.

Во фрагменте дерева перебора решений верхняя вершина—множество всех туров; нижняя левая — множество всех туров, включающих ребро (1,2); нижняя правая — множество всех туров, не включающих ребро (1, 2).

Продолжим ветвление влево-вниз. Для этого оценим нули в матрице C5:

$C_5 =$

	1	3	4	5	6
2	O¹	1	4	1	0
3	O³		O¹	0	3
4	3	O¹	–	1	3
5	3	0	1	-г-"	0
6	6	3	3	O¹	

Максимальная оценка в клетке (3, 1) равна 3. Таким образом, оценка для правой нижней вершины на рис. 12.18 будет равна 35 + + 3 = 38. Для оценки левой нижней вершины (рис. 12.18) нужно

Рис. 12.18. Второй этап построения дерева решений

вычеркнуть из матрицы C5 строку 3 и столбец 1, получив матрицу C_6:

	3	4	5	6
2		4	1	0
4	O¹	–	1	3
5	0	1	> -г .	0
6	3	3	O¹	– ',

В эту матрицу нужно поставить запрет в клетку (2, 3), так как уже построен фрагмент тура из ребер (1,2) и (3,1) и нужно запретить преждевременное завершение тура (2, 3). Матрица приводится по столбцу на 1, таким образом, каждый тур соответствующего класса (т.е. тур, содержащий ребра (1, 2) и (3,1)) стоит 36 условных единиц. Далее

	3	4	5	6
2		3	1	0
4	O¹	–	1	3
5	0	O²		0
6	3	2		

Теперь оцениваем нули в приведенной матрице *Ст.* Нуль с максимальной оценкой 3 находится в клетке (6,5). Отрицательный вариант имеет оценку 38 + 3 = 41. Для получения оценки положительного варианта удаляем строку 6 и столбец 5, ставим запрет в клетку (5,6):

	3	4	6
2	£98	3	O³
4	O³		3
5	0	O³	

Матрица *C»* приведенная, следовательно, оценка положительного варианта не увеличивается.

Оценивая нули в матрице *C»,* получаем ветвление по выбору ребра (2,6), отрицательный вариант получает оценку 36 + 3 = 39, а для получения оценки оптимального варианта вычеркиваем вторую строку и шестой столбец, получая матрицу Cg:

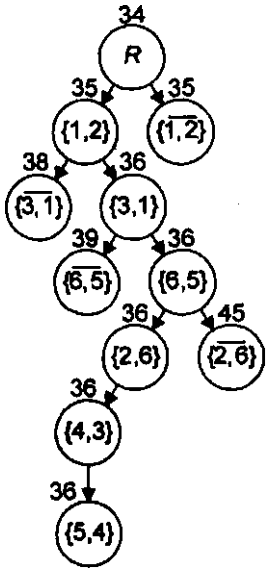


Рис. 12.19. Ограниченное дерево перебора

C_9 =

	3	4
4	0	
5		0

В матрицу надо добавить запрет в клетку (5,3), так как уже построен фрагмент тура [3,1,2,6,5] и надо запретить преждевременное завершение тура (5,3). После преобразований получилась матрица 2 х 2 с нулевыми элементами по диагонали. Элементы, помеченные нулем (ребра (4, 3) и (5,4)), автоматически включаются в тур, не меняя его длины.

Таким образом, получен тур: (1 - 2 - 6 — 5 — 4 - 3 - 1) стоимостью в 36 (рис. 12.19). При достижении низа по дереву перебора класс туров сузился до одного тура, а оценка снизу превратилась в точную стоимость.

Контрольные вопросы

1. Какое множество называется рекордом?
2. В чем сущность улучшения рекорда?
3. Укажите главный принцип жадного алгоритма.
4. В чем особенность деревянного алгоритма?
5. Как строится дерево перебора для расшифровки криптограмм?
6. Назовите основные принципы метода ветвей и границ.
7. Сформулируйте условие задачи коммивояжера.
8. Что означает «привести матрицу по строкам»?
9. Что такое функция штрафа?
10. Как исключается досрочное завершение тура?
11. Что такое нижняя граничная оценка?

Глава 13
Моделирование с использованием генераторов случайных чисел

Многие явления в природе, технике, экономике и в других областях носят случайный характер. В этом случае величина, принимающая свои значения в зависимости от исходов некоторого испытания (опыта), называется *случайной величиной.*

Пусть *X* — дискретная случайная величина, возможными значениями КОТОРОЙ ЯВЛЯЮТСЯ ЧИСЛа *X\,X2,..., Xn.*

Обозначим через *щ = P(X = xi), i* = 1, 2,..., *n,* вероятности этих значений, т.е. ***pi*** — вероятность события, состоящего в том, что *X* принимает значение ***X{.***

Закон распределения полностью задает дискретную случайную величину. Однако часто закон распределения случайной величины неизвестен. В таких ситуациях ее описывают числовыми характеристиками.

13.1. Числовые характеристики случайных величин

Случайные величины характеризуются следующими числовыми параметрами:

• *Математическое ожидание M(X)* —это статистическое среднее случайной величины:

$$M(X) = \sum_{i=1}^{n} x_i p_i,$$

где ***Xi*** — значение случайной величины, ***pi*** — вероятность появления этой величины. При этом

п

Для равновероятных событий

$$M(X) = l \pm x_i .$$
$$i=l$$

•***Дисперсия*** — это математическое ожидание квадрата отклонения случайной величины от ее математического ожидания:

$$D\{X\}^{\wedge}Y_i\{x_i\text{-}M\{X\}f_{pi}.$$

• Корень квадратный из дисперсии называется ***средним квадратичным отклонением*** *o(X)* случайной величины, т.е.

• ***Коэффициент корреляции*** определяется для двух потоков случайных величин:

$$\text{щ x ,y} > \sim \frac{M(XY)\text{-}M(X)M(Y)}{c(X)a(Y)} \quad \bullet$$

Коэффициент корреляции определен на отрезке [—1;1], т.е. —1 < *<R{X,Y)<1.*

Потоки случайных величин, для которых *R(X, Y)* — 0, называются *некоррелированными (независимыми).*

Рассмотрим алгоритмы для детерминированной выборки случайных чисел.

13.2. Метод середины квадрата

Один из ранних генераторов случайных чисел, принадлежащий Джону фон Нейману (1946), известен как *метод середины квадрата.*

Метод используется для генерации ^-разрядных псевдослучайных чисел. Должно быть задано fc-разрядное начальное число ***хо*** (для удобства предполагаем, что ***к*** четно). Это число возводится в квадрат, и получается число ***у.*** Число ***у*** должно иметь ***2к*** разрядов. Если число разрядов меньше ***2к,*** то число ***у*** дополняется слева нулями. Затем в ***у*** выделяют средние ***к*** разрядов, которые и дают новое случайное число.

Алгоритм сводится к выполнению следующих шагов.

Ш а г 0. Инициализация: ***х*** := ***XQ.***

Ш а г 1. Основной цикл: FOR *j* := 1 TO m DO шаг 2; Stop.

Ш а г 2. Генерация нового случайного числа ***ху.*** ***у*** := x^2; ***Xj :=*** (средние ***к*** разрядов числа у); х := ***Xj.*** (Число ***у*** будет иметь ***2к*** разрядов, а следующее случайное число ***XJ*** получается, если удалить по

к/2 разрядов с каждого конца ***у.*** Десятичная точка помещается перед первым разрядом числа ***Xj*** до поступления его на выход случайного генератора.)

П р и м е р 1. Получить три случайных числа методом середины квадрата. Выберем ***к*** = 4; ***XQ*** = 3167. Тогда:

XQ = 3167; ***у*** = 10029889:
xi = 0298; ***у*** = 00088804:
X2 = 0888; ***у*** = 00788544:
x_3 = 7885.

П р и м е р 2. Для ***к*** = 4 и хо = 2134 получить первые восемь псевдослучайных чисел, выработанных алгоритмом середины квадрата, в интервале (0; 1):

xl = 04553956,	Xi = 0,5539
**?* = 30680521,	x_2 = 0,6805;
x = 46308025,	*хз* = 0,3080;
Л = 09486400,	***x*** = 0,4864;
А = 23658496,	X5 = 0,6584;
4 = 43349056,	26 = 0,3490;
4 = 12180100,	X_7 = 0,1801;
4 = 03243601,	***х8*** = 0,2436.

Несмотря на видимую случайность чисел, генерируемых алгоритмом, ему свойственны недостатки. В самом деле, если в последовательности когда-нибудь появится число 0,0000, то все следующие за ним числа будут также равны 0,0000. Таким образом, многое зависит от начального выбора А; и хо-

13.3. Линейный конгруэнтный метод

Метод используется для генерации последовательности x i, X2, ..., x_m из *m* псевдослучайных чисел. Должны быть заданы следующие входные значения:

• ***Ь*** — целочисленный множитель, ***b*** > 1;
• хо — начальное случайное целое число хо > 1;

• ***к*** — шаг, ***к*** > 0 целое;

• ***m*** — целочисленный модуль, ***m*** > **xn, *Ь, к.***

Случайные числа генерируются по рекуррентной формуле:

$$Xj = (bxj\text{-}i + к) \bmod m$$

Алгоритм сводится к выполнению следующих шагов.

Ш а г 1. Основной цикл:

FOR j := 1 TO m ЕЮ шаг 2; шаг 3; Stop.

Ш а г 2. Генерация нового необработанною случайного числа:

$$Xj := (bxj\text{-}i + к) \bmod rfi.$$

Число ***XJ*** должно лежать в полуинтервале 0 < **x** < ***m.*** По определению для целых ***а*** и ***m*** (a mod тп) есть остаток от целочисленного деления ***а*** на ***m.*** Например, 5 mod 3 = 2, 7 mod 3 = 1 9 mod 3 = 0.

Ш а г 3. Генерация следующего случайного числа: ***yj := Xj/m*** ***(yj*** будет лежать в полуинтервале ***0 <yj*** < 1 и обладать требуемым распределением).

П р и м е р 1. Получить три случайных числа линейным конгруэнтным методом. Выбираем **xo** = 27; *b* = 5; ***к*** = 10; ***m*** = 40. В результате:

$$\mathbf{x}_1 = (5 \cdot 27 + 10) \bmod 40 = 145 \bmod 40 = 25;$$

$$\mathbf{x}_2 = (5 \cdot 25 + 10) \bmod 40 = 15;$$

$$\mathbf{x}_3 = (5 \cdot 15 + 10) \bmod 40 \quad 5.$$

П р и м е р 2. Для ***к = 0, m*** = 2^{10}, ***Ь*** = 101, **xo** = 432 получить первые восемь псевдослучайных чисел, выработанных линейным конгруэнтным методом, при этом

$$\mathbf{XI} = 624, \quad 2/1 = \frac{624}{1024} = 0,610;$$

$$\mathbf{x}_2 = 560, \quad 2/2 = \frac{560}{1024} = 0,546;$$

$$x_3 = 240, \quad 2/3 = \frac{240}{1024} = 0\ 234;$$

$$\mathbf{X4} = 688, \quad 2/4 = \frac{688}{1023} = 0,67\&$$

$$\mathbf{x}_5 = 880, \quad 2/5 = \frac{880}{1024} = 0,859";$$

$$\mathbf{Xб} = 816, \quad 2/6 = \frac{816}{1024} = 0,790!$$

$$\mathbf{X}_7 = 496, \quad 2/7 = \frac{496}{1024} = 0.48\&;$$

$$\mathbf{Z8} = 944, \quad 2/8 = \frac{944}{1024} = 0,923\text{-}$$

Существует такой выбор параметров ***к, Ь, m,*** **xo,** при котором алгоритм будет генерировать числа на отрезке [0; 1], представляющиеся непредсказуемыми и удовлетворяющие определенным статистическим критериям. Для всех практических целей эти числа оказываются последовательностью наблюдений равномерно распределенной случайной переменной.

13.4. Полярный метод генерации случайных ч1исел

Метод используется для генерации ДВУ^{x} независимых случайных чисел с нормальным распределением ^(0>1) ^{из} Д^{в}У^{x} независимых равномерно распределенных случайных чисел. Распределение iV(0,1) преобразуется в ***N(fi,a²)*** с использование" центральной предельной теоремы.

Алгоритм сводится к выполнению следующих - шагов.

Ш а г 1. Генерация двух равномерно распреде?$^{а\ е\ и\ и\ ы\ х}$ случайных чисел. Генерируются два независимых случайнь*^{1X} числа *щ* и u_2 с распределением *U*(0,1); *v\ :=2ui* - 1; *vi* •= $2u_2$ ^~ 1; **("1 " "2** имеют распределение *U*(-1,1)).

Ш а г 2. Вычисление и проверка *s: s := v* + *v* IF *s* > 1 THEN GOTO шаг 1.

Ш а г 3. Вычисление *щ* и n_2: *n\ := u* ^ - - $^{/\ \ \ s}$ ^; n_2 := u_2 **/ 21ns** у - —

Ш а г 4. Stop, *(щ* и n_2 распределены нормально)

Алгоритм имеет вычислительное преимущество перед другими методами: он обычно генерирует по одному числу с нормальным распределением на каждое число с равномерным распределением Правда, при этом следует убедиться, что компенсируется дополнительное время, затрачиваемое на вычисление натуральных логарифмов и квадратных корней.

Контрольные вопросы

1. Какие существуют типы числовых характеристик случайных величин!
2. Дайте определение математическому ожиданию.
3. Дайте определение дисперсии.
4. Что показывает коэффициент корреляции?
5. В чем особенность метода середины квадрата?
6. Какие исходные данные используются в линейном конгруэнтном методе?
7. Какие случайные числа получают полярным методом?

Глава 14
Машина Тьюринга

В 1937 г. английский математик Тьюринг опубликовал работу, в которой уточнил понятие алгоритма, прибегая к воображаемой вычислительной машине.

Машина Тьюринга — один из способов записи алгоритма, **ИЛР** алгоритм, записью которого является функциональная таблица ил* функциональная диаграмма, а правилом выполнения — описание е< устройства.

Машина Тьюринга так же, как и конечный автомат, являете) дискретным устройством преобразования информации. Приведем е< точное определение, а затем интерпретацию ее работы.

Машиной Тьюринга называется частичное отображение

M: {Qo, <Эь .... Q n - i } x { 0 ,

-> {Qo, ***Qi, ... Qn-i)*** x {Л, П} x {0, 1},

где Л, П — соответственно «влево», «вправо»;

{Qo, Q i , . . . , *Qn-l)* — множество состояний машины.

Тот факт, что отображение частичное, означает, что *M* может быть определено не для всех наборов аргументов. Машина Тьюринга работает с бесконечной в обе стороны лентой, разбитой на ячейки, в каждой из которых написан один из символов 0 или 1.

Считывающая (записывающая) головка машины обозревает в каждый момент времени одну из ячеек и за один такт, сменяющий два последовательных момента времени, может перемещаться влево или вправо. Машина Тьюринга в каждый момент времени находится в одном из состояний *Qo, Qi, Q*„-\, а в следующий момент времени переходит в другое состояние или остается в том же. Кроме того, машина может изменять символ, стоящий в обозреваемой ячейке. Все эти преобразования — изменения состояния, информации на ленте, направления движения полностью определяются отображением *M.* Так, если *M(i,e) = (j,* Л, 77), то в случае, когда машина находится в состоянии *Qi,* а на обозреваемой в данный момент ячейке написан символ е, машина должна записать в эту ячейку *rj* вместо е, перейти в состояние *Qj* и сдвинуться на одну ячейку влево. Если М(г,е) = *(j,* П, *г)),* те же действия будут сопровождаться сдвигом вправо.

Например, равенство М(2,1) = (1, П, 1) означает, что, находясь в состоянии *Qi* и обозревая ячейку, в которой написан символ 1, машина должна сохранить в этой ячейке символ 1, сдвинуться вправо и перейти в состояние *Q*\. Если же *M(г,е)* не определено, то машина, находясь в состоянии *Qi* и обозревая ячейку с символом *е,* прекращает свою работу, не изменяя своего состояния, информации на ленте и никуда не сдвигаясь.

Примечание. Существуют различные модификации машины Тьюринга (машина Поста, машина Минского и т.д.). Некоторые модификации предусматривают на ленте не символы 0 или 1, а буквы какого-либо конечного алфавита *A* = {ai, *a„ a„}.* В некоторых определениях разрешается не только сдвиг головки машины вправо или влево, но и оставление ее на прежней позиции. Однако различные модификации машины Тьюринга эквивалентны между собой в том смысле, что классы функций, вычисляемых на этих машинах, совпадают.

14.1. Структура машины Тьюринга

В качестве исполнителя алгоритмов Тьюринг предложил автомат, состоящий из:

• бесконечной ленты, разбитой на ячейки;

• головки (каретки), способной передвигаться над лентой, от ячейки к ячейке, считывать символы, записанные на ленте, записывать символы в ячейки.

В каждой ячейке ленты может быть записан только один из определенного множества символов, называемого алфавитом. За одно срабатывание каретка способна выполнить следующие действия:

• считать символ из ячейки, над которой она находится;

• записать символ в ячейку, над которой она находится;

• переместиться либо влево, либо вправо на следующую ячейку, либо остаться на месте;

• изменить свое внутреннее состояние.

Предполагается, что каретка может находиться в одном из состояний, определяемых ее положением на бесконечной ленте. Одним из ее действий наряду с перечисленными выше является переход из одного состояния в другое.

На рис. 14.1 представлена структурная схема машины Тьюринга, где обозначены:

B — внешняя память машины, которую можно интерпретировать, как неограниченную в обе стороны ленту, которая делится на элементарные ячейки;

Q — внутренняя память машины, определяющая состояния, в которых находится машина в любой момент времени;

G — считывающая (записывающая) головка;

Рис. 14.1. Структурная схема машины Тьюринга

Z — логический блок машины. Этот блок формирует символ, который будет записан на ленту, а также управляет переходами машины из одного состояния в другое;

P — устройство, управляющее головкой. Основные кодовые символы для управления машиной:

R — движение головки вправо;

L — движение головки влево;

H — продолжать обозревать текущую ячейку.

Символы, записанные на ленте *B,* образуют внешний или исходный алфавит. Среди символов исходного алфавита выделяется пустой, который чаще всего обозначают символом А. Тогда информация на ленте может иметь следующий вид:

{А А 0 1 1 0 0 1 Л А}.

Алфавит внутренних состояний машины задается множеством: ***{Qo> Я\, <72, Як},*** где ***qo*** — начальное состояние, определяющее начало алгоритма; $q_к$ — конечное состояние, определяющее конец алгоритма.

Головка находится в стандартном положении, если она располагается на ленте у самого левого символа, отличного от пустого.

Машина Тьюринга работает следующим образом: на ленту записывается исходная информация, и головка устанавливается в исходное положение; затем через логический блок информация считывается с ленты, и на основании закона функционирования машины определяется символ, который нужно записать на ленту.

14.2. Функциональные таблицы и диаграммы

Законы функционирования машины определяются тремя символами: *Q, a, P,* где *Q*—состояние машины; *a*—символ, который записывается на ленту; *P* — управление движением головки.

При этом получаем систему правил (команд) вида

$$q_o aj \qquad q\backslash, a), d_s,$$

т.е. после обзора символа *aj* головкой в состоянии *qi* в ячейку записывается символ aj, головка переходит в состояние *q*, а лента совершает движение aV Для каждой комбинации *qi, aj* имеется *ровно одно* правило преобразования. Это означает, что логический блок реализует функцию, сопоставляющую каждой паре входных сигналов *qi, aj* одну и только одну тройку выходных *q*, aj, *d*. — она называется *логической функцией машины* и обычно представляется в виде таблицы *(функциональной схемы машины),* строки которой обозначаются символами состояний, а столбцы — знаками внешнего алфавита. Бели знаков внешнего алфавита *n,* а число состояний логического управления (ЛУ) *m,* то, очевидно, общее число правил преобразования составит *nm.*

Конкретная машина Тьюринга задается перечислением элементов множеств *A* и *Q,* а также логической функцией, которую реализует *ЛУ,* т.е. набором правил преобразования. Ясно, что различных множеств *A, Q* и логических функций может быть бесконечно много, т.е. и машин Тьюринга также бесконечно много.

Совокупность состояний всех ячеек ленты, состояния ЛУ и положение головки называется *конфигурацией машины.*

Записать конфигурацию можно следующим образом:

Aaiqiaj ...akA,

которая означает, что в слове из ***к*** символов обозревается секция номер ***j*** и при этом управляющее устройство находится в состоянии Ясно, что конфигурация машины может содержать любое количество символов внешнего алфавита и лишь один символ внутреннего.

Работа машины Тьюринга определяется функциональной таблицей, в которой на пересечении строки *qi* и столбца *aj* указывается тройка символов *q*, aj, *dk,* где *q* — состояние, в которое переходит машина; aj —символ, который будет записан на ленту; *dk* —указатель перемещения головки.

Запись алгоритма определяется набором правил преобразования. Из табл. 14.1 видно, что исходными данными являются двоичные символы 0,1. Работа алгоритма определяется четырьмя состояниями: *qo,* 9ъ 92,9fc-

Тройка символов *qiOR,* записанная на пересечении строки 90 и столбца 0, означает: если машина находится в состоянии *qo* и с ленты

Таблица 14.1

Запись алгоритма функциональной таблицей

Состояние	Исходный алфавит	
	0	1
90 **9i** **92**	**9i** *OR* **91**ОЯ **9i** *OR*	**90***IR* **92**1Я *ЯкОЬ*

будет считан символ 0, тогда машина перейдет в состояние *q*, на ленту будет записан символ 0, и головка сдвинется на один символ вправо.

Если информация в ячейке не обновляется или машина остается в прежнем состоянии, то функциональная таблица упрощается (табл. 14.2).

Таблица 14.2

Упрощенная функциональная таблица

Состояние	Исходный алфавит	
	0	1
90 **9i** **92**	**9i** *R* *R* **91**Я	*R* **92Д** *qkL*

От функциональной таблицы можно перейти к функциональной диаграмме. Функциональная диаграмма — способ записи алгоритма в виде графа или графовой структуры (рис. 14.2).

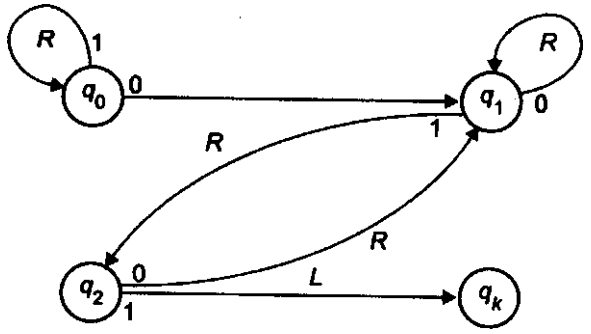


Рис. 14.2. Функциональная диаграмма

14.3.
Примеры записи алгоритмов

Пример 1. Задан двоичный код, заключенный в пустые символы А. Составить машину Тьюринга для получения копии двоичного кода на ленте. Например, исходные данные — {ААА1010011ААА}, результирующее множество — {ААА1010011*1010011ААА}.

Анализ алгоритма.

Шаг 1. Движемся вправо, пока не встретим А, ставим на место А символ *.

Шаг 2. Движемся влево до А и сдвигаемся на один символ вправо, записываем вместо 1 символ *а*.

Шаг 3. Движемся вправо до А и вместо А ставим 1.

Шаг 4. Движемся влево до любого символа, отличного от 1 и 0. Сдвигаемся на один символ вправо. Меняем 0 на /3,1 на *а* и т.д., пока не будет получена копия двоичного кода. После этого заменяем *а* на 1, /3 на 0.

Пример 2. Работа машины Тьюринга с исходным алфавитом *А* = {1А} задана следующей функциональной таблицей:

Состояние	Исходный алфавит	
	1	А
91	*1R*	*1R*

Данная функциональная таблица позволяет бесконечно заполнить всю ленту единицами вправо от выбранной точки.

Ниже приведена таблица, выполняющая те же самые операции до тех пор, пока на ленте не встретится символ 0:

Состояние	Исходное множество		
	1	А	0
9i	*1R*	*1R*	%1Я

Пример 3. Составить функциональную таблицу для инвертирования двоичного кода.

Например, исходное множество — {ААА01101ААА}, результирующее множество — {ААА10010ААА}.

Решение. Функциональная таблица будет иметь вид:

Состояние	Исходное множество		
	0	1	А
Qo	*1R*	*OR*	***ЯкН***

Пример 4. Составить алгоритм для увеличения на единицу числа, записанного в десятичной системе счисления.

Например исходное множество — {ААА7ААА}, результирующее множество —{ААА8ААА}.

Решение. Функциональная таблица будет иметь вид:

Состояние	Исходное множество символов										
	0	1	2	3	4	5	б	*7*	8	9	А
Яо	1ot	2%	39*	*Мк*	5%	6%	*7Як*	*8Як*	*9Як*	*0L*	*Як*

Примере Составить алгоритм преобразования числа из множества {1, 2, 3, 4, 5} в унарную запись.

Примечание. Унарная запись — представление числа палочками например 3 -^ || |.

Решение. Функциональная таблица будет иметь вид:

Состояние	Исходное множество символов							
	0	1	2	3	4	5	А	*
90	*R*	*R*	*R*	*R*	*R*	*R*	91 * Я	
91	*ЯкН*	*Я2R*	9зЯ	94-R	*ЯsR*	*qeR*		*L*
92	—	—		—	—		9*1#	*R*
93	—	-	-		—	—	92*1R*	*R*
94	—	—	-	-	—	-	93 1Д	*R*
95	—		-	=	—	—	94*1R*	*R*
96	—	=	-				951Д	*R*
			_	_	_	_		

14.4.
Композиция машин Тьюринга

Написать программу работы машины Тьюринга—значит составить диаграмму или таблицу ее функционирования. Программирование машины Тьюринга состоит в умении разделить процедуру решения задачи на последовательность из трех шагов: запись символа, считывание символа, перемещение головки на одну позицию.

Если требуется написать достаточно сложную программу, для упрощения процесса программирования применяются операции композиции машин Тьюринга, т.е. построение сложной машины из более простых, при этом используются две операции: умножение машин Тьюринга и итерация машин Тьюринга.

Умножение машин Тьюринга. Пусть заданы две машины Тьюринга: $T\backslash$ $mлT_2$. В начальный момент времени to на ленте имеется некоторая конфигурация, которую начинает обрабатывать машина $T\backslash$, отправляясь от некоторого начального состояния $q\backslash$, причем головка машины в момент *to* находится напротив ячейки с номером ***IQ*** (рис. 14.3).

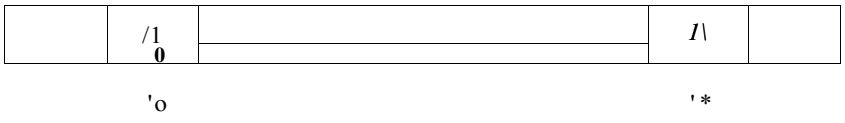


Рис. 14.3. Умножение машин Тьюринга (работа машины Ti)

Тогда к некоторому моменту $t\backslash$ машина Ti перейдет в состояние $q\backslash$, а головка машины остановится напротив ячейки $1\backslash$. В момент времени $t\backslash$ машину $T\backslash$ отключаем, и с этого момента начинает работать машина T_2, отправляясь от своего начального состояния $q^$, причем в момент времени $t\backslash$ головка будет расположена напротив ячейки $1\backslash$ (рис. 14.4).

В момент времени $t\backslash$ машина T_2 закончит работу и остановится напротив ячейки $l\backslash$.

Из рис. 14.4 видно, что результатом последующей работы машин $T\backslash$ и T_2 будет некоторая машина T, функциональная таблица которой строится по правилу: верхняя часть таблицы описывает работу машины $T\backslash$, а нижняя часть — работу машины T_2, причем состояние

Яо

Рис. 14.4. Умножение машин Тьюринга (работа машины T_2)

останова машины 7\ (состояние $q\backslash$) отождествляется с начальным состоянием машины T_2.

Пример 1. Пусть заданы две машины $T\backslash$ и T_2 (состояние *qfi):*

Состояние	Символы	
	0	1
90	9iOL	90 *1L*
91	*Я\0Ь*	» 1 Я

,

$m_2 =$

Состояние	Символы	
	0	1
90	9*0Я	90ОХ

Тогда композиция машин $T\backslash$ и T_2 будет выглядеть следующим образом:

$T = T_1T_2 =$

Состояние	Символы	
	0	1
90	*qiOL*	90 IX
91	*qiOL*	921Я
92	*ЯкОН*	9201

В том случае, когда одна из перемножаемых машин имеет несколько состояний останова, указывается, какой из остановов предыдущего сомножителя отождествляется с начальным состоянием последующего.

Система команд (таблица или диаграмма) машины T есть результат объединения системы команд машин 7\ и T_2. Из сказанного нетрудно видеть, что машина $T = T\backslash \bullet T_2$ работает так, как после завершения работы машины $T\backslash$ начала бы работать машина T_2. Очевидно, что произведение машин Тьюринга не коммутативно ,т.е. $T\backslash T_2 \phi T_2 T\backslash$.

Если машина $T\backslash$ (записываемая в произведении первой) имеет не одно, а два заключительных состояния, то появляется неоднозначность (индетерминизм) в том, с каким из этих состояний отождествлять начальное состояние машины T_2.

Пример 2. Пусть машина $T\backslash$ имеет два состояния останова. Тогда

Начальное состояние машины T_2 отождествляется с первым состоянием останова машины $T\backslash$. Таким образом, машина Т будет иметь два заключительных состояния: одно совпадает с состоянием останова машины T_2, другое — со вторым заключительным состоянием машины Ti.

Итерация машин Тьюринга. Итерация машин Тьюринга заключается в отождествлении г-го состояния останова машин Тьюринга с начальным состоянием.

Машина Т может быть задана следующим образцом:

$$T = \begin{matrix} '(1); \\ < \\ AS). \end{matrix}$$

Значения в скобках указывают на отождествление состояний останова с начальным состоянием. Если машина $T\backslash$ имеет лишь одно состояние останова, то в этом случае машина не будет иметь останова и станет бесконечной. Если итерация произведена над машиной, которая в свою очередь является результатом умножения и итерации других машин, то соответствующее число точек ставится над теми машинами, чьи состояния останова и начальные отождествляются.

Пример 3. Пусть задана машина Тьюринга

$$T = \begin{matrix} Г(1)Т_3; \\ \backslash\ (2)Т_4f_5; \\ 1\ (3)Т_6. \end{matrix}$$

Здесь показано умножение и итерация машин Тьюринга. При этом состояние останова машины Тз отождествляется с начальным состоянием машины Ti, а состояние останова машины Т5 — с начальным состоянием машины T_2.

Контрольные вопросы

1. Какова структура машины Тьюринга?
2. Чем машина Тьюринга отличается от машины Поста?
3. Что такое конфигурация машины?
4. Каким образом записывается система команд (правил)?
5. Что такое внутренняя и внешняя память машины Тьюринга?
6. В чем заключается особенность построения функциональных таблиц?
7. Какова связь функциональной таблицы с функциональной диаграммой?
8. Что такое композиция машин?
9. В чем смысл умножения машин Тьюринга?
10. Что такое итерация машин Тьюринга?

Глава 15
Элементы математической логики

В древности различные мыслители пробовали давать рецепты правильных умозаключений, которые от истинных посылок приводят только к истинным выводам. Таких мыслителей называли логиками. Наука установила общие методы правильных умозаключений, называемых формальной логикой.

Термин «логика» происходит от древнегреческого *logos,* означающего «слово, мысль, понятие, рассуждение, закон».

Понятие — это форма мышления, в которой отражены существенные (отличительные) свойства объектов.

Суждение — это форма мышления, отражающая связь понятий друг с другом.

Умозаключение—это процесс получения нового суждения-вывода из одного или нескольких данных суждений.

Высказывание — это любое предложение какого-либо языка (утверждение), содержание которого можно определить как истинное или ложное.

Предикат — высказывание, содержащее одну или несколько неизвестных.

Всякое высказывание или истинно, или ложно; быть одновременно и тем и другим оно не может. Формулировка любой теоремы является высказыванием. Высказывания могут выражаться с помощью математических, физических, химических и прочих знаков. Из двух числовых выражений можно составить высказывания, соединив их знаками равенства или неравенства. Сами числовые выражения высказываниями не являются. Не являются высказываниями равенства или неравенства, содержащие переменные. Например, предложение $X < 12$ становится высказыванием при замене переменной каким-либо конкретным значением. Такие предложения называют *высказывательныи формами.*

Примерами высказываний могут служить:

1) {Число 2 является делителем числа 7} (ложное высказывание);
2) {3 + 5 = 2 * 4} (ложное высказывание);
3) {2 + 6 > 10} (ложное высказывание);
4) {II + VI > VIII} (ложное высказывание); ,
5) {Сумма чисел 2 и 6 больше числа 8} (ложное высказывание);
6) {Two plus six is eight} (истинное высказывание);
7) {Студент *X* — лучший по информатике} (предикат).

Высказывание называется *простым (элементарным),* если никакая его часть сама не является высказыванием. Если условие не выполняется, высказывание называется *сложным.*

В алгебре логики, как и в обычной алгебре, вводится ряд операций.

15.1. Алгебра высказываний

На сегодняшний день в логике не существует унифицированной символики для обозначения логических операций. Во избежание путаницы полезно иметь в виду следующую таблицу (табл. 15.1).

Действия и преобразования, применяемые в обычной алгебре, в которой буквами обозначаются числа, основываются на небольшом числе определений и формул.

• Существует арифметическое действие, называемое сложением и обозначаемое знаком +. Устанавливается, что для каждой пары данных двух чисел *а* и *Ь* существует единственное определенное число с, называемое суммой чисел *а* и *Ь.* Действие сложения обладает переместительным и сочетательным свойствами. Отсюда формулы:

Таблица 15.1

Символика логических операций

Обыденная речь	Символика			
	Шредера—Пирса	Пеано — Рассела	Гильберта	Лукасевича
нер	*p'*	~ р	***P***	***Np***
если р, то ***q***	***p-*q***	***pDq***	***p->q***	***Cpq***
р тогда и только тогда, когда ***q***	***p = q***	***p = q***	***p~q***	***Epq***
pvimq	***p + q***	***pVq***	***p\lq***	***Apq***
pwq	***pq***	рЛд	***p&cq***	***Kpq***

1) *a + b = c* (существование единственной суммы чисел *а* и *Ь);*

2) а + Ь = Ь + а (переместительное свойство);

3) *а + (Ь + с) = (а + Ь) + с* (сочетательное свойство).

• Существует второе арифметическое действие, называемое умножением и обозначаемое знаком х или • (последний знак при употреблении буквенных обозначений обычно не ставится). Действие умножения обладает теми же свойствами, что и сложение: для каждых двух чисел *а* и *Ь* существует определенное единственное произведение *аЬ,* и действие умножения обладает переместительным и сочетательным свойствами, которые дают формулы:

4) *ab = d* (существование произведения);

5) *ab = Ьа* (переместительное свойство);

6) *а(Ьс) = (аЬ)с* (сочетательное свойство).

• Сложение и умножение обладают распределительным свойством: чтобы умножить сумму двух слагаемых на третье число, можно умножить каждое слагаемое отдельно на это число и полученные произведения сложить:

7) *(а + Ь)с = ас + bc* (распределительное свойство).

• Существует такое число, обозначаемое знаком 0 (нуль), при сложении которого с любым числом *а* получается в сумме то же число о, а при перемножении его (т.е. нуля) с любым числом *а* получается в произведении 0. Отсюда формулы:

8) *а + 0 = 0 + о = а;*

9) о • 0 = 0 • *а* = 0.

• Существует еще число, обозначаемое знаком 1 и называемое единицей, при перемножении с которым любого числа *а* получается в произведении то же число а:

10)а-1 = 1-а = а.

Отмеченные десять формул являются основными законами обычной арифметики и алгебры.

Примечание. Джордж Буль, давший в 1847 г. первое изложение алгебры логики, сделал предположение, что буквы в записанных десяти формулах обозначают не числа, а высказывания, и показал, что можно выбрать такие определения действий сложения и умножения, при которых все десять формул остаются в силе.

Не должен удивлять тот факт, что надо выбрать новые определения сложения и умножения. Это делается и в обычной алгебре. Действия сложения и умножения для дробных, отрицательных или комплексных чисел имеют иной смысл, чем для натуральных чисел. При переходе от одной числовой области к другой надо определять, что будет называться суммой или произведением новых чисел.

Может показаться бессмысленным понятие арифметических действий над высказываниями. В алгебре логики высказывание рассматривается не по его содержанию или смыслу, а только в отношении того, истинно оно или ложно. Принимается, что каждое высказывание может быть только истинно или ложно.

Условимся истинность высказывания обозначать единицей, а ложность — нулем. Тогда каждое высказывание может быть охарактеризовано цифрами 1 или 0, которые являются мерами или функциями истинности высказывания *а.* Для любого высказывания *а* либо *а* — 1, либо *а* = 0.

Сумма двух высказываний о и *Ь,* т.е. *а* + *Ь,* является сложным высказыванием, которое, как всякое высказывание, может быть истинно или ложно. Сумма двух высказываний считается истинной, т.е. равной единице, если хоть одно из складываемых высказываний истинно: *а* + *b* = 1, если или *а* = 1 или 6=1, что согласно с обычной арифметикой: 1+0 = 0 + 1 = 1.

Если оба складываемых высказывания истинны, то сумма считается также истинной, поэтому в алгебре логики (1) + (1) = 1.

Скобки поставлены для того, чтобы подчеркнуть условный, необычный смысл этого сложения.

Сумма двух высказываний считается ложной и равной нулю тогда и только тогда, когда оба слагаемых ложны, т.е. 0 + 0 = 0.

В алгебре логики «сумма» *а* + *Ь* часто называется *дизъюнкцией* и знак + обозначается знаком V (первой буквой латинского слова «vel» — или).

Произведение *а* • *b* двух высказываний *а* и *b* является также сложным высказыванием. Оно считается истинным (равным единице) тогда и только тогда, когда оба сомножителя истинны, и ложным (равным нулю), если хоть один из сомножителей ложен. Это определение произведения соответствует обычной арифметике: 1-1 = 1; 1 0 = 0 1 = 0; 0-0 = 0.

Первое равенство читается так: если и *а,* и *b* истинны, то произведение *ab* истинно. Значит, знак умножения х или • заменяет союз «и».

В алгебре логики «умножение» называется *конъюнкцией* и обозначается особым знаком Л.

Установив данные определения сложения и умножения, легко доказать, что все десять правил обычной алгебры остаются верными и в алгебре логики.

Пример 1. Докажем переместительные законы. Для этого составим соответствующие таблицы, давая буквам *а* и *b* по порядку значения 1 или 0, комбинируя их и находя по данным правилам меру истинности суммы и произведения:

а	*Ь*	*а* + *Ь*	*Ъ* + *а*	*ab*	*ba*
1	1	1	1	1	1
1	0	1	1	0	0
0	1	1	1	0	0
0	0	0	0	0	0

Меры истинности сумм *а* + *b* и *b* + *а* (третий и четвертый столбцы) при всех возможных комбинациях мер истинности *а* и *b* совпадают, следовательно *а* + *bub* + *а в* отношении истинности всегда «равны». То же происходит с произведениями *ab* и *Ьа* (пятый и шестой столбцы таблицы).

Пример 2. Рассмотрим доказательство распределительного закона (о + 6)с = *ас* + *bc,* составив соответствующую таблицу:

a	b	$a + b$	с	$(a + b)c$	ac	be	$ac + be$
1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0
1	0	1	1	1	1	0	1
1	0	1	0	0	0	0	0
0	1	1	1	1	0	1	1
0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Некоторые особенности алгебры высказываний. В алгебре высказываний вводится действие, которого нет в обычной алгебре — отрицание данного высказывания. Для каждого высказывания *а* существует его отрицание «не-а», которое будем обозначать символом *а*.

Если высказывание о истинно (и), то его отрицание *а* ложно (л); если *а* ложно, то его отрицание *а* истинно. Это можно выразить таблицами:

a	a
и	л
л	и

или

a	a
1	0
0	1

Из определения смысла действия отрицания, именно из того положения, что из противоположных высказываний а и а всегда истинно одно и только одно, следуют новые формулы алгебры логики:

11) $a + a = 1$;

12) $aa = 0$.

В алгебре логики существуют еще другие упрощающие формулы, которых нет в обычной алгебре, например $a + a = a, a + a + a = a, \ldots, a + \ldots + a =$ о.

Точно так же $aa = a^2 \quad a, \quad aa \ldots a = a^n = a,$ где n — целое положительное число.

Кроме распределительного закона обычной алгебры $(a + Ь)c = ac + bc$ в алгебре логики имеется еще другой распределительный закон:

$$(a + c)(b + c) = ab + c.$$

Пример 3. Докажем справедливость этого закона в алгебре логики обычным методом применения таблицы истинности:

a	ъ	с	ab	$a + c$	$Ь + c$	$(a+c)(Ь+c)$	$ab + c$
1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	0	1	0	1	1	1	1
1	0	0	0	1	0	0	0
0	1	1	0	1	1	1	1
0	1	0	0	0	1	0	0
0	0	1	0	1	1	1	1
0	0	0	0	0	0	0	0

Рассмотрим примеры формул поглощения:

$$a + ab = \text{о}(1 + b) = a \cdot 1 = \text{а};$$

$$\text{а}(\text{а} + b) = a^2 + ab = a + ab = \text{а}(1 + Ь) = \text{а} \cdot 1 = \text{а}.$$

Все изложенное приводит к заключению: так как все основные формулы обычной алгебры верны и для алгебры высказываний, то все преобразования, употребляемые в обычной алгебре при решении уравнений, остаются верными и в алгебре высказываний.

Логические операции. В алгебре логики рассматриваются переменные, которые могут принимать только два значения 0 и 1. Переменные будем обозначать латинскими буквами х, *у, z*. . . . а также x_i, хь . . . , x_{ij}, уо, Ш. . . . .Уг»и т.д.

Отношение эквивалентности (равенства) удовлетворяет следующим свойствам:

• рефлексивность: х = х;

• симметричность: если х = *у,* то *у = х;*

• транзитивность: *х = уиу = г,тох = г;* отсюда следует принцип: если х = *у,* то в любой формуле, содержащей х, вместо *х* можно подставить у, и в результате будет получена эквивалентная формула.

Пусть х i , 12 и 1з — переменные булевского типа (логические переменные), способные принимать лишь два значения *(True* и *False),* которые для удобства мы будем обозначать соответственно 1 и 0.

Над логическими переменными определен ряд простейших логических функций, значения которых определяются по правилам (табл. 15.2).

Таблица 15.2

Логические функции

	x_2	$X1\ AX2$	x 1 V X2	$X\backslash$ —¥ X2	Xl ф x 2	$Xl \sim X2$	Xl
0	0	0	0	1	0	1	1
1	0	0	1	0	1	0	0
0	1	0	1	1	1	0	
1	1	1	1	1	0	1	

Функции имеют следующие названия и обозначения:

- *конъюнкция* (логическое умножение) — Л (И);
- *дизъюнкция* (логическое сложение) — V (ИЛИ);
- *импликация* ———>;
- *сумма по модулю 2* — ф;
- *функция эквивалентности* ———;
- *отрицания* (логическое отрицание) — -> (НЕ).

Справедливы следующие *соотношения эквивалентности* между простейшими функциями:

$$x\backslash\ -+x_2 = xTVx_2;$$

$$xi@x2 = (xi\ Лхг)\ V\ (xi\ Лхг);$$

$$xi \sim xг = (x"i\ Л\ xг)\ V\ (xi\ Л\ xг).$$

1. Конъюнкция: *A A B* или *A • B* (логическое умножение, читается как союз «и»). Таблица истинности для конъюнкции:

A	*B*	*АлВ*
0	0	0
0	1	0
1	0	0
1	1	1

2. Дизъюнкция: *A* V *B* или *A + B* (логическое сложение, читается как союз «или»). Таблица истинности для дизъюнкции:

A	*B*	*АУВ*
0	0	0
0	1	1
1	0	1
1	1	1

3. Отрицание: *A* или -<*A*. Иногда отрицание называют функцией Вебба или функцией Даггера.

4. Импликация, или *логическое следование* (читается: «если *A*, то *B*»). Обозначается *A* —* *B*. Таблица истинности для импликации:

A	*B*	*A ^ B*
0	0	1
0	1	1
1	0	0
1	1	1

5. Эквиваленция, или *тождественность: A +> B (A ~ B)* (читается: *«А* тогда и только тогда, когда В»). Таблица истинности для эквиваленции:

A	*B*	*A ^ B*
0	0	1
0	1	0
1	0	0
1	1	1

6. *Штрих Шеффера* (антиконъюнкция; читается: «неверно, что *A* и *B*»): *A\B = ~A~XB.*

7. *Стрелка Пирса* (антидизъюнкция; читается: «неверно, что *A* или В»): *A | B = AУB.*

15.2. Законы математической логики

В алгебре логики выполняются следующие основные законы (табл. 15.3), позволяющие производить тождественные преобразования логических выражений.

Если левая и правая части одновременно, т.е. при одинаковых наборах значений входящих в них переменных, принимают одинаковые значения, то закон доказан.

Для логических операций рассмотрим ряд теорем и законов.

Коммутативные законы:

$$x + y = y + x, \quad x\text{-}y = y\text{-}x.$$

Ассоциативные законы:

$$(x + y) + z = x + (y + z), \quad (x\text{-}y)\text{-}z = x\text{-}(yz).$$

Дистрибутивные законы:

$$x \cdot (y + z) = x \cdot y + X \cdot Z, \quad x + y \cdot z = (x + y) \cdot (x + z).$$

Таблица 1S.3

Основные законы алгебры логики

Закон	Для дизъюнкции	Для конъюнкции
Переместительный	* V У = У V x	t-y = y-x
Сочетательный	x V (y V z) = (x V y) V z	X-(y • z) = (xy)z
Распределительный	x(yVz) = (x-y)V(x-z)	xV(yi) = (xVy)-(xVz)
Правила де Моргана	xУy=yx	аГ^у = у Vx"
Идемпотенции	x Ух = x	x · x = x
Поглощения	x V (x • y) = x	X • (x V y) = x
Склеивания	(x-y)V(xy) = y	y)(x V y) = y
Операция переменной с ее инверсией	x Vx = 1	(xV x • x = 0
Операция с константами	xV0 = x, xV1 = 1	= x, x • 0 = 0
Двойного отрицания	x = x	x 1

Законы склеивания:

$$xy + xy = x, \quad (x + y)(x + y) = x, \quad x + xy = x + y, \quad x(x + y) = xy.$$

Большинство законов алгебры логики записаны для конъюнкции и дизъюнкции. При внимательном изучении пар можно вывести принцип двойственности: если в логическом тождестве произвести взаимные замены операций дизъюнкции и конъюнкции, то получим двойственное тождество. Такое свойство называется *принципом двойственности.*

Законы алгебры логики могут быть доказаны аналитически или методом перебора (по таблице истинности).

Например, докажем тождество $x + xy = x + y$ аналитическим методом:

$$x + xy = ж(1 + y) + xy = x + xy + xy =$$
$$= XX + Xy + Xy + XX = (X + X)X + (X;+- x)y = X + y.$$

15.3. Решение логических задач

При решении логических задач используют приемы:

- построение таблицы истинности;
- решение логических уравнений или систем логических уравнений.

Для решения многих логических задач необходимо:

1) внимательно изучить условие задачи;

2) выделить элементарные (простые) высказывания и обозначить их буквами;

3) записать условие задачи на языке алгебры логики, соединив простые высказывания в сложные с помощью логических операций;

4) составить единое логическое выражение для всех требований задачи;

5) используя законы алгебры логики, попытаться упростить полученное выражение и вычислить все его значения либо построить таблицу истинности для рассматриваемого выражения;

6) выбрать решение—набор значений простых высказываний, при котором логическое выражение является истинным;

7) проверить, удовлетворяет ли полученное решение условию задачи.

Построение таблиц истинности. После того, как будет составлено логическое выражение, удовлетворяющее всем условиям, можно заполнить для него таблицу истинности. Анализ полученной таблицы истинности позволит получить требуемый результат.

Задача 1. Составим таблицу истинности для формулы

$$x - yV\,xV\,yV\,x,$$

которая содержит две переменные x и y. В первых двух столбцах таблицы запишем четыре возможные пары значений этих переменных, в последующих столбцах — значения промежуточных формул и в последнем столбце — значение формулы:

Переменные		Промежуточные логические формулы					Формула			
X	У	X	x-y	xУy	xУy	x-y У xУy	x-y	Ух	Уy	Ух
0	0	1	0	0	1	1	1			
0	1	1	1	1	0	1	1			
1	0	0	0	1	0	0	1			
1	1	0	0	1	0	0	1			

Из таблицы видно, что при всех наборах значений переменных x и y формула $x\text{-}yУ\,x\,У\,y\,У\,x$ принимает значение 1, т.е. является тождественно истинной.

Задача 2. Построить таблицу истинности для выражения *хУух* х *(х-у):*

Переменные		Промежуточные логические формулы				Формула
X	*у*	*х У у*	*х У у*	*у*	*х-у*	*х* V *у* • *(х • у)*
0	0	0	1	1	0	0
0	1	1	0	0	0	0
1	0	1	0	1	1	0
1	1	1	0	0	0	0

Из таблицы видно, что при всех наборах значений переменных х и у формула *хУу • (х-у)* принимает значение 0, т.е. является тождественно ложной.

Задача 3. По обвинению в ограблении перед судом предстали Иванов, Петров, Сидоров. Следствием установлено следующее:

1) если Иванов не виновен или Петров виновен, то Сидоров виновен;

2) если Иванов не виновен, то Сидоров не виновен.

Виновен ли Иванов?

Решение. Рассмотрим простые высказывания:

А = {Иванов виновен};

В = {Петров виновен};

С = {Сидоров виновен}.

Запишем на языке алгебры логики факты, установленные следствием:

(АУВ)->С и *А->С.*

Пусть *F = (А У В)* -» С) Л *(А ->• С).* Составим для данного высказывания таблицу истинности:

А	*В*	*с*	*F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Решить данную задачу — значит указать, при каких значениях *А* полученное сложное высказывание истинно. Необходимо проанализировать все строки таблицы истинности, где *F* = 1, и если хотя бы в одном случае *F* = 1 при *А* = 0 (Иванов не виновен), то у следствия достаточно фактов для того, чтобы обвинить Иванова в преступлении.

Анализ таблицы показывает, что сложное высказывание истинно во всех случаях, когда *А* — истинно, т.е. Иванов виновен в ограблении.

Задача 4 (финансовый прогноз). Три подразделения *А, В, С* торговой фирмы стремились получить по итогам года максимальную прибыль. Экономисты высказали следующие предположения:

1) если *А* получит максимальную прибыль, то максимальную прибыль получат также *В* и С;

2) либо Л и С получат максимальную прибыль одновременно, либо не получат;

3) для того чтобы *С* получило максимальную прибыль, необходимо, чтобы и *В* получило максимальную прибыль.

По завершении года оказалось, что одно из трех предположений ложно. Какие из названых подразделений получили максимальную прибыль?

Решение. Рассмотрим простые высказывания:

А — {А получает максимальную прибыль};

В = {В получает максимальную прибыль};

С = {С получает максимальную прибыль}.

Запишем на языке алгебры логики прогнозы, высказанные экономистами:

1) F_1 = А - > (В А С) ;

2) F_2 = *(ААС)уААС;*

3) F_3 = *С ->• В.*

Составим таблицу истинности для *Fi,* F_2, F_3:

А	*В*	*с*	*Fi*	F_2	*Fs*
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Теперь вспомним, что один из прогнозов F_1: F_2, F_3 оказался ложным. Эта ситуация соответствует четвертой стро ке таблицы.

Ответ: *В к С* получат максимальную прибыль.!

Задача 5. Три подруги — Джуди, Айрис и J инда приобрели известность в разных видах искусств -— пении, **багете** и кино. Все они живут в разных городах: в Париже, Риме и Чика/ Известно, что:

1) Джуди живет не в Париже, а Линда — н е в Рч ме;

2) парижанка не снимается в кино;

3) та, которая живет в Риме, — певица;

4) Линда равнодушна к балету.

Где живет Айрис и какова ее профессия?

Решение. Составим таблицу и отразим в не? условия 1 и 4, заполнив клетки цифрами 0 и 1 в зависимости от гого, ложно или истинно соответствующее высказывание:

Париж	Рим	Чикаго	Имя	Пение	—1	>алет	Кино
0			Джуди				
			Айрис				
	0		Линда			0	

Далее рассуждаем следующим образом. Так как инда живет не в Риме, то, согласно условию 3, она не певица. В клетнк соответствующую строке «Линда» и столбцу «Пение», ставим 0. *t* з таблицы сразу видно, что Линда киноактриса, а Джуди и Айрис не *a* имаются в кино:

Париж	Рим	Чикаго	Имя	Пение		!алет	Кино
0			Джуди				0
			Айрис				0
	0		Линда	0		0	1

Согласно условию 2 парижанка не снимается кино, следовательно, Линда живет не в Париже. Но она живёт и не в Риме, Следовательно, Линда живет в Чикаго. Так как Линра и Джуди живут не в Париже, там живет Айрис. Джуди живет в име и согласно условию 3 является певицей. А так как Линда киноа»|триса, то Айрис балерина.

В результате постепенного заполнения получаем следующую таблицу:

Париж	Рим	Чикаго	Имя	Пение	Балет	Кино
0	1	0	Джуди	1	0	0
1	0	0	Айрис	0	1	0
0	0	1	Линда	0	0	1

Ответ: Айрис балерина. Она живет в Париже.

Задача 6. На вопрос, кто из трех учащихся *А, В* и *С* изучал математику, были даны ответы:

- если изучал *А,* то изучал и *В;*
- не верно, что если изучал *С,* то изучал и *В.*

Кто из них изучал математику?

Решение. Запишем условие задачи, используя язык алгебры *(А -> В) А (С -+ В).* Для этой формулы имеем таблицу истинности:

А	*В*	*с*	*А->В*	*с ^ в*	(А -> *В) А (С -> В)*
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	0	0

Ответ: математику изучал *С.*

Задача 7 **(на конгрессе).** На конгрессе встретились четверо ученых: физик, биолог, историк и математик. Национальности их были различны, и, хотя каждый ученый владел двумя языками из четырех (русским, английским, французским или итальянским), не было такого языка, на котором они бы могли разговаривать вчетвером. Есть только один язык, на котором могли вести беседу сразу трое. Никто из ученых не владеет французским и русским языками одновременно. Хотя физик не говорит по-английски, он может служить переводчиком, если историк и биолог захотят поговорить друг с другом. Историк говорит по-русски и может говорить с математиком, хотя тот не знает ни одного

русского слова. Физик, биолог и математик могут баредовать на одном языке.

Какими двумя языками владеет каждый ученый!

Решение. Эту задачу удобнее всего решить, заш лняя следующую таблицу:

	Русский	Английский	Французски		Итальянский
Математик					
Биолог					
Физик					
Историк					

Будем анализировать условия задачи и постепе! но выяснять, ка-кими языками владеет тот или иной ученый, ставя шак «плюс» или «минус» в ячейках, соответствующих тем языкам, оторые он знает или не знает.

Известно, что математик не знает русского, физЦ: — английского, историк — французского (он говорит по-русски, но жкто не говорит на русском и французском одновременно).

Физик служит переводчиком в беседах историЛа и биолога (он владеет двумя языками, на которых историк и биолог i е могут говорить одновременно). Так как историк и биолог не владею общим языком, то, следовательно, биолог не знает русского языка.

Значит, русский — общий язык для физика и юторика; физик не владеет французским (он говорит по-русски, но икто не говорит на русском и французском одновременно). Второй язык физика — итальянский; итальянским владеет и биолог, истомик итальянским не владеет. Тогда второй язык историка — английский, а биолог английским не владеет. Значит, второй язык биолога - французский.

Историк может беседовать с математиком, хо> тот не знает русского. Следовательно, математик владеет английс им.

Так как только трое ученых одновременно вл! цеют одним из языков, то этот язык — итальянский. Заполним итоге »ую таблицу:

	Русский	Английский	Французский!	Итальянский
Математик		+	–	+
Биолог	−		+ j	+
Физик	∓	−	−	
Историк	+	∓		
			−	−

Ответ: математик владеет английским и итальянским; биолог — французским и итальянским; физик — русским и итальянским; историк — русским и английским.

Задача 8. Вадим, Сергей и Михаил изучают различные иностранные языки: китайский, японский и арабский. На вопрос, какой язык изучает каждый из них, один ответил: «Вадим изучает китайский, Сергей не изучает китайский, а Михаил не изучает арабский». Впоследствии выяснилось, что в этом ответе только одно утверждение верно, а два других ложны. Какой язык изучает каждый из молодых людей?

Решение. Имеются три утверждения:

1) Вадим изучает китайский;

2) Сергей не изучает китайский;

3) Михаил не изучает арабский.

Если верно первое утверждение, то верно и второе, так как юноши изучают разные языки. Это противоречит условию задачи, поэтому первое утверждение ложно. Если верно второе утверждение, то первое и третье должны быть ложны. При этом получается, что никто не изучает китайский. Это противоречит условию, поэтому второе утверждение тоже ложно. Остается считать верным третье утверждение, а первое и второе — ложными. Следовательно, Вадим не изучает китайский язык, китайский изучает Сергей.

Ответ: Сергей изучает китайский язык, Михаил — японский, Вадим — арабский.

Задача 9. На заводе работают три друга: слесарь, токарь и шлифовщик. Их фамилии Борисов, Иванов и Семенов. Даны утверждения:

1) у слесаря нет ни братьев, ни сестер;

2) слесарь самый младший из друзей;

3) Семенов женат на сестре Борисова;

4) Семенов старше токаря.

У кого какая специальность?

Решение. При решении задачи необходимо привести множества фамилий и специальностей во взаимно однозначное соответствие, т.е. осуществить биекцию множеств. Осуществить биекцию множеств A и B — значит привести эти множества во взаимно однозначное соответствие, т.е. каждому элементу из A поставить один единственный элемент из B или наоборот.

В множестве фамилии обозначены буквами: С1 - Семенов; И — Иванов ; Б — Борисов. В множестве специальн|сти обозначены буквами: С — слесарь; Т — токарь; Ш — шлифовщ] к.

Биекция множеств в данном случае показывав' (рис. 15.1), что Иванов — столяр, Борисов — токарь и Семенов — i1lшифовщик.

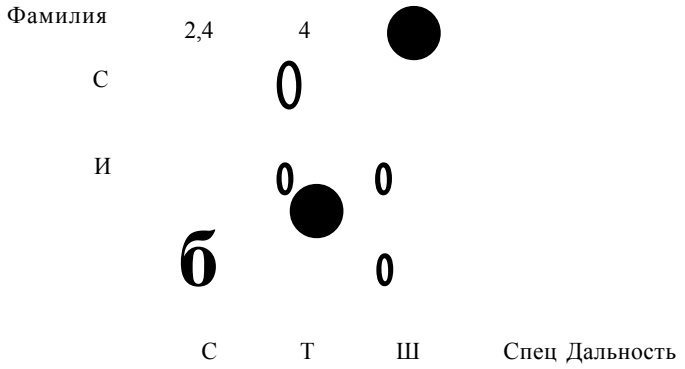


Рис. 15.1. Биекция множеств, цифрами отмечены утЦерждения

Примеры упрощения логических формул. Г Ькажем на примерах некоторые приемы и способы, применяемые при упрощении логических формул:

1) *хУу - (ху) = х-у-(х-у) = х-ху-у = 0-у* = 0 у = 0

(законы алгебры логики применяются в следующе последовательности: правило де Моргана, сочетательный закон, п||авило операций переменной с ее инверсией и правило операций с константами);

2) *х• уУхУуУх = х-уУх-уУх = х-(уУу)Уш= хУх* = 1

(применяется правило де Моргана, выносится за см жи общий множитель, используется правило операций переменной! ее инверсией);

3) (х V») • (х *У у) • (хУу) = (х У у) • (хУу) • {хУу) сУу) = у-х*

(повторяется второй сомножитель, что разрешено зак ном идемпотенции; затем комбинируются два первых и два послед! IX сомножителя и используется закон склеивания);

А) х-уУх-у- гУ х- г = х-уУх-у- гУ х- г(уУ) =*

= х- уУх-у-гУху-гУх-у-г=>

= (х-уУх-у-г)У(х-у-гУх-у- = х • *у У у • z*

(вводится вспомогательный логический сомножитель *у У у;* затем комбинируются два крайних и два средних логических слагаемых и используется закон поглощения);

5) х • *у У z = х^у • z =* (х V *у) • z*

(сначала добиваемся, чтобы знак отрицания стоял только перед отдельными переменными, а не перед их комбинациями, для этого дважды применяем правило де Моргана; затем используем закон двойного отрицания);

6) х • *у У х • у • гУ х • z • р = х • (у •* (I V *z)у z • р)* = х • *(у У z • р)*

(выносятся за скобки общие множители; применяется правило операций с константами);

7) *хУу-гУхУуУг* = х V | / V f V х - у - f =

= *хУуУгУх-у-г* = *хУгУ(уУх-у-г)* = *хУгУу*

(к отрицаниям неэлементарных формул применяется правило де Моргана; используются законы двойного отрицания и склеивания).

Решение логических уравнений. При решении логических задач методом составления логических уравнений используют следующие правила:

• если после преобразований получили уравнение, у которого в левой части многочлен, а в правой части стоит единица, то это означает, что по крайней мере одно слагаемое равно единице;

• если сумма нескольких слагаемых равна нулю, то все они являются ложными высказываниями;

• знак «+» в логическом выражении означает «или» (верно то или другое);

• знак «*» обозначает союз «и» (имеет место и то, и то);

• если получилось уравнение вида: *А • В • С* = 1, то все высказывания истинны;

• если получается уравнение вида: *А • В С* = 0, то по крайней мере одно из них ложно;

• для каждого высказывания *А* существует Х При этом *А + ~А~=1 иАА = 0;*

• *А + А +... + А = А* для любого количества слагаемых;

• если в логическом выражении какое-нибудь слагаемое содержит противоречие, то такое слагаемое считается равным 0 и удаляется;

• применяйте, где возможно, формулы, вытекающие из определений сложения и умножения: о + *а* = 1, *аа* = 0.

Задача **10.** Четыре ученицы: Мария (М), Нина (ЛГ), Ольга **(о)** и Полина (Р) участвовали в соревновании и заняли первые четыре места. На вопрос, кто из них какое место занял, три девушки ответили:

1) Ольга была вторая, Полина — третья;

2) Ольга была первая, Нина — вторая;

3) Мария была вторая, Полина — четвертая.

В каждом из этих ответов одна часть верна, другая неверна.

Какое место заняла каждая из четырех учениц?

Решение. Введем обозначения для высказываний: «Ольга была первая» — Ох.

«Мария была вторая» — M_2 и т.д.

Приступая к решению, по первому условному ответу нельзя сказать, будет ли **02** = 1 или **02** = 0, Рз = 1 или Рз = 0.

Но одна часть ответа верна, т.е. или **02** = 1, или Рз = 1, поэтому 0_2 + Рз = 1.

Из второго и третьего условных ответов имеем:

$$0\,i + i\,V_2 = 1,$$
$$M_2 + P_4 = 1.$$

Итак, имеем три логических уравнения, которые все должны удовлетворяться одновременно, т.е. имеем систему уравнений:

$$\begin{cases} 0_2 + Рз = 1, \\ \mathbf{01} + ЛГ_2 = 1, \\ M_2 + P_4 = 1. \end{cases}$$

Перемножив почленно два первых уравнения системы, получим:

$$\mathbf{02\ 01} + \mathbf{02\text{-}V2} + Рз\,O_2 + P_2N_2 = 1.$$

Из этого уравнения следует, что по крайней мере одно из слагаемых в левой части уравнения истинно и равняется единице. При этом 02 0i = 0, так как это утверждение противоречиво (Ольга заняла второе и первое место), и $0_2H_2 = 0$, так как *О и N* обе не могли быть вторыми. Остается:

$$P_3\,0\,i + P_3\,N_2 = 1.$$

Помножив это уравнение почленно на третье уравнение системы, получим

$$Рз\,Oi\,M_2 + P_3\,Oi\,P_4 + P_2N_2M_2 + \mathbf{P3N2P4} = 1.$$

В этом уравнении **Р3О1Р4** = 0 и **P3N2P4 =** 0 (Полина не может занимать третье и четвертое место); $P_2N_2M_2 = 0$ (Нина и Мария не могут обе занимать второе место).

Остается **Рз01М**$_2$ = 1 или **01М2Р3** = 1.

Ответ: Ольга заняла первое, Мария — второе, Полина — третье место. Если так, то Нина заняла четвертое место, и ответ будет

$$01M_2P_3Щ = 1.$$

Задача 11. В велогонке участвовали пять учащихся и заняли первые пять мест. На вопрос, кто из них какое место занял, ребята ответили:

1) Сережа *(S)* занял второе место, Коля *(К)* — третье;

2) Толя (Т) — пятое место, Надя *(N)* — третье;

3) Толя (*Т)* — первое место, Надя *(N)* — второе;

4) Толя (*Т)* — первое место, Надя *(N)* — второе;

5) Толя (Г) — пятое место, Надя *(N)* — третье;

6) Сережа *(S)* — второе место, Ваня (V) — четвертое;

7) Коля *(К)* — первое место, Ваня *(V)* — четвертое.

В каждом ответе одна часть верна, другая неверна. Найти, кто какое место занял.

Решение без уравнений. Если верно, что Сережа занял второе место, то в третьем ответе неверно, что Надя заняла второе, а верно, что Толя занял первое место. Но тогда по четвертому и пятому ответам Коля занял первое место, что невозможно, так как это место занял при сделанном предположении Толя. Предположение, что Сережа занял второе место, невозможно.

Если это предположение невозможно, то по первому ответу, верно, что Коля занял третье место. По пятому ответу утверждение, что Коля занял первое место, неверно; значит, Ваня занял четвертое; по второму ответу предположение, что Надя заняла третье место, также неверно; значит, Толя занял пятое место, а по третьему ответу Надя — второе. Итак, при сделанном предположении заняли места: Коля — третье,

Ваня — четвертое, Толя — пятое и Надя — второе. Следовательно, Сережа занял первое место.

Решение при помощи уравнений. При употреблении введенного нами в предыдущих задачах способа обозначать высказывания символами, имеем:

$$S_2 + K_3 = l; \quad S_2K_3 = 0; \tag{1}$$

$$N_3 + T_5 = l; \quad N_3T_5 = 0; \tag{2}$$

$$Ti + N_2 = l; \quad T_xN_2 = 0; \tag{3}$$

$$S_2 + V_4 = l; \quad S_2V_4 = 0; \tag{4}$$

$$K_x + V_4 = l- \quad K_xV_4 = 0. \tag{5}$$

Почленное перемножение уравнений (2) и (3) дает:

$$N_3T_x + N_3N_2 + T_5T\backslash + T_bN_2 = ЛГЗГ1 + T_bN_2 = 1 \tag{6}$$

($N_3N_2 = 0$ и Т5Т1 = 0, как выражающие невозможные высказывания).

Почленное перемножение уравнений (4) и (5) дает:

$$S_2K_1 + S_2V_4 + V_4Ki + V_4 = S_2Ki + V_4 = l, \tag{7}$$

так как $S_2K\backslash = 1$ и $V_4K_x = 0$.

Почленное перемножение уравнений (1) и (6) дает:

$$S_2N_3T_x + S_2T_3N_2 + K_3N_3T\backslash + K_3T_5N_2 = 1,$$

Отсюда с учетом $S_2N_2 = 0$ и $K_3N_3 = 0$ имеем:

$$S_2N_3Ti+K_3T_5N_2 = l. \tag{8}$$

Почленное перемножение результатов (7) и (8) дает:

$$S_2N_3TiKi + K_3T_bN_3S_2Ki + V_4S_2N_3T_x + V_4K_3T_5N_2 = 1.$$

Первое, второе и третье слагаемые равны нулю, как содержащие

$$XiTi = 0, \quad K_2K_3=0, \quad V_4S_2 = 0.$$

Остается $N_2K_3V_4T_b = 1$ или $S_2N_2K_3V_4T_b = 1$.

Ответ: Сережа был первым, Надя — второй, Коля — третьим, Ваня — четвертым, Толя — пятым.

Задача 12. Три девушки: Аня *(А),* Валя *(В)* и Клава *(К)* ходили в гости. Одна из них была в красном платье, другая — в белом, третья — в синем. На вопрос, какое на каждой из девушек было платье, они дали ответ: Аня была в красном, Валя — не в красном, Клава — не в синем.

В этом условном ответе из трех частей одна верна, две неверны. В каком платье была каждая из девушек?

Решение с помощью логических уравнений. Введем обозначения высказываний:

•Ак — Аня в красном;

$\underline{B_к}$ — Валя в некрасном,

$K_с$ — Клава в несинем платье.

Условный ответ девушек выражается тройкой высказываний $A_кB_кK_с$.

Верной может быть лишь одна из трех частей этого ответа, причем остальные две части неверны. Возможны три предположения:

1. Если верно, что Аня была в красном платье *(А),* то неверны утверждения: Валя не в красном *(В~Ї)* и Клава — не в синем *(К)* и верны: Валя — в красном *(В)* и Клава — синем *(К).* Этот вывод выражается произведением высказываний $A_кB_кK_с$.

2. Если верно, что Валя — не в красном *(В),* то неверны утверждения: Аня — в красном ($А_к$) и Клава — не в синем *(К),* а верны: Аня — не в красном ($Л_к$) и Клава — в синем *(К),* что выражается произведением $A_к \quad B_кK_с$.

3. Если верно, что Клава — не в синем *(К),* то неверны утверждения: Аня — в красном *{А)* и Валя — не в красном *(В),* а верны: Аня — не в красном *(А)* и Валя — в красном ($Б_к$). Это предположение дает произведение $A_кB_кK_с$.

Одно из трех предположений верно. Поэтому имеем

$$A_кB_кK_с + A_к \quad B_кK_с + A_кB_кK_с = 1.$$

В уравнении $A_кB_кK_с = 0$, так как *А* и *В* не могли быть обе в красном платье, и $A_к \; B_кK_с = 0$, так как при таком предположении синее платье занято Клавой, и для Ани и Вали не остается двух некрасных платьев.

Можно было бы рассуждать и так: $A_к \; B_к \; K_с$ можно записать в виде *JQ* $B_к \; K_с$; это предположение невозможно, так как по нему ни одна из

девушек не одета в красное, значит $A_к\ B_к\ K_к = 0$ и >авнозначное ему $A_к\ B_к K_с = 0$.

Следовательно, остается $A_к B_к K_с = 1$. Это юзможно, когда $A_с B_к Кб = 1$.

Ответ: Аня — в синем платье, Валя — в кр« ном, Клава — в белом.

Задача 13. Шесть школьников *C, D, H, I,* , *T* ходили на олимпиаду. Двое из них решили все задачи. На прос, кто решил **BI** все задачи, они ответили: 1) *C* и *G;* 2) *D* и Т; 3) и *C;* 4) *D* и /; 5) Я и С.

В четырех из ответов одна часть верна, другая [еверна; в одном из ответов обе части неверны. Кто из учеников ешил задачи на олимпиаде?

Решение с помощью уравнений. Условия задач! можно записать так:

$$CG = DT = TC = DI = HC = 0; \qquad (9)$$

$$(C + G)(D + T)(T + C)(D + I)(H + C = 0. \qquad (10)$$

Равенства нулю в (9) следуют из того, что в кажд м произведении по крайней мере один сомножитель равен нулю. В п; оизведении одна из скобок равна нулю (оба слагаемых ее нули), q :тальные четыре скобки равны единице.

Раскроем скобки в (10) и уничтожим на осноі (9) отдельные слагаемые: перемножим первый сомножитель на втоj ой, результат — на третий, новый результат — на четвертый, третий результат — на пятый сомножитель:

$$(C + G)\ (D + T) = CD + CT + GD + GT = CD \quad GD + GT,$$

так как $CT = 0$ по (9);

$$(CD + GD + GT)(T + C) = CDT + GDT +$$

$$+ GT^2 + C^2D + CGD + CG \quad = GT + CD,$$

так как по (9) $DT = CG = CT = 0$,а $GT^2 + C^2D$ $T + CD;$

$$(GT + CD)(D + I) = GTD + CD^2 + GTI + C \quad U = CD,$$

так как по (9) $DT = DI = 0$, а $GTI = 0$ потому, что задачу не решили трое, что неверно;

$$CD(H + C) = CDH + C^2D = CD.$$

Итак, левая часть выражения (10) равна CD и согласно условию (9) $CD = 0$.

В уравнении (10) правая часть есть нуль, потому что четыре скобки равны каждая единице, одна скобка равна нулю. Однако не известно, какая из скобок дает нуль. Если из пяти множителей выражения (10) составим все произведения по четыре множителя в каждом произведении, то таких произведений будет пять:

$$(C + G)\{D + T)(T + C)(D + I);$$

$$(C + G)(D + T)(T + C)(H + C);$$

$$(C + G)(D + T)(D + I)(H + C);$$

$$(C + G)(T + C)(D + 7)\ (Я + С);$$

$$(D + T)(T + C)(D + I)(H + C).$$

Четыре из пяти скобок равны единице, одна скобка равна нулю, так как в ней оба слагаемых равны нулю. Такая скобка войдет в четыре произведения и обращает их в нуль; в одно из пяти произведений она не войдет, и это произведение равно единице (например, если $C + G = 0$, то пятое произведение равно единице, если $D + T = 0$, то четвертое произведение равно единице и т.д.).

Следовательно, одно из пяти произведений равно единице, и сумма

$$(C+G)\{D+T)(T+C)(D+I)+(C+G)(D+T)(T+C)(H+C) \quad +$$

$$+ \quad \{C+G)\{D+T)(D+I)\{H+C) \quad + \quad \{C+G)\{T+C)(D+I)\{H+Q) \quad +$$

$$+ (D + T)\{T + C)(D + I)(H + C) = 1. \qquad \textbf{(11)}$$

Предполагая, что в каждом из слагаемых раскрыты скобки, выпишем в получаемых многочленах слагаемые, состоящие из двух букв (такие из трех или четырех букв равны нулю).

Имеем из первого слагаемого $C^2D^2 = CD = 0$; из второго слагаемого $C^2D = CD = 0$ и $С^3Т = CT = 0$; из третьего слагаемого $C^2D^2 = CD = 0$ и $С^3/ = CI$ (равенства нулю CI имеющиеся уравнения не дают); из пятого слагаемого $C^2D^2 = CD$ — 0.

Итак, из левой части уравнения (11) осталось і получим *CI* = 1.

Ответ: Задачи решили С и / .

Задача 14. При решении одной задачи ученик

1) *X* есть число иррациональное, равное пл! треугольника, у которого сторона *а* = 2;

2) *X* — число кратное 4 и равно радиусу < которой 2;

3) *X* < 3 и равно диагонали квадрата, сторона;

В каждом из ответов одна часть верна, друп равно X?

Замечание. Упростим выражение вторых частей

1) площадь правильного треугольника, сторона ко

| N / 3 = N/3;
4

2) радиус окружности, длина которой 2, дает *2пг* =

3) диагональ квадрата, сторона которого равна 2, ее

После этого условные ответы задачи можно за!

- *X* — число иррациональное, равное л/3;
- X кратно четырем и равно -;
- *X* < 3 и равно *2у/2.*

Решение без применения уравнений.

1. *X ф* л/3, так как при предположении *X* = і первом условном ответе обе части верные, а имени является иррациональным числом. Первый отве условию.

2. Третий ответ также невозможен, так как ложении, что *X* = 2v^2 в этом ответе, обе части вер

3. Второй ответ удовлетворяет условию задачи нии, что *X* — -, первая часть условного ответа *"X* так как кратным числу 4 может быть только целое ч* ответа верна.

Ответ: X = —.

Ответ удовлетворяет всем трем условиям:

1) число - иррациональное, но не равно -\/3. Значит, в первом условном ответе первая часть верна, вторая неверна;

2) во втором условном ответе вторая часть верна, первая неверна: *X* = -, но - не является кратным 4;

3) в третьем условном ответе первая часть верна: — = —-— < 3, 7г 3,14...

но вторая часть неверна: - *ф* 2\/2.

Решение при помощи уравнений. Введем обозначения для выска-зываний: *[J]* — *X* число иррациональное;

л/3] — *X* равно *V3;*

кратно 4] — *X* кратно 4;

равно!;

< 3] — *X* меньше 3;

[2V2] — *X* равно 2ч/2.

Как и при решении уравнений предыдущих задач, имеем систему уравнений:

[./] + Ь/3] = 1;

[кратно 4] + = 1;

[< 3] + [2\/2] = 1.

Почленное перемножение первых двух уравнений дает:

И[кратно 4] + *[J]* **[I]** + [^[кратно 4] + [л/3] [£] = 1.

Здесь:

[7][кратно 4] = 0, так как число *X* не может быть одновременно иррациональным и целым;

[J] [-1 возможно, так как - иррациональное число;
L7TJ 7г

[\/3] [кратно 4] = 0, так как \/3 не кратно 4;

[v/3]fl] = 0, так как - *ф* \/3.

Уравнение получит вид: *[J]* **^j** = 1.

Таким образом, получено, что ***X*** = -, которое есть иррациональное число. Проверим, не противоречит ли этому третье условное уравнение. Перемножим почленно полученный результат с третьим условным уравнением:

$$[J][I][\hat{}\} + [J][\pm][2V2\} = 1.$$

В первом слагаемом нет противоречий: — — число иррациональное и меньше 3, второе слагаемое ***[J]*** **Jij** ***[2\/2]*** = О, так как оно утверждает, что ***X*** = -= ^ и в то же время равно 2\/2. Последнее неверно.

Итак, имеем для значения ***X*** уравнение: ***[J]*** **j** [< 3] = 1, отсюда

Jf = I.

Задача 15. В один из осенних дней четыре друга Альберт, Карл, Дидрих и Фридрих впервые переступили порог школы. Учительница сказала им, что с этого дня она будет называть их по имени и фамилии. Оказалось, что у друзей фамилии те же, что и имена, только так, что ни у одного из них имя и фамилия не были одинаковы. Кроме того, фамилия Дидриха не была Альберт. Определить фамилию каждого из мальчиков, если дано, что имя мальчика, у которого фамилия Фридрих, есть фамилия того мальчика, имя которого — фамилия Карла.

Решение при помощи уравнений. Условимся обозначать имена первыми буквами и фамилии указателями при обозначении имени.

Так, если имя мальчика *A,* а фамилия С, то это запишется символом *Ac* и ***AQ*** = 1; если же фамилия мальчика с именем *A* не ***C,*** то *Ac* = 0. При такой записи по условиям задачи:

$$A_A = C_c = D_D = F_F = D_A = 0.$$

По последнему условию задачи мальчик по имени ***X*** с фамилией *F,* мальчик по имени ***Y*** с фамилией ***X*** и мальчик с фамилией ***C*** с фамилией ***Y,*** иными словами: *XpYxCy* = 1.

Непосредственный путь для нахождения значений ***X*** *Y* состоит в подстановке вместо ***X*** и *Y* по порядку всех четырех букв *A,* ***C,*** *D, F,* т.е. в вычислении суммы (суммирование обозначается символом YJ) ***Y,X,Y,C,.***

После подстановки имеем:

$$X\phi C, \qquad X\hat{}F, \qquad Y\hat{}C, \qquad Y\hat{}F,$$

так как нет двух мальчиков с одинаковыми именами или одинаковыми фамилиями.

Остается в $\hat{}X_FY_xCy$ вместо *X* и У подставлять только *A* и *D.* Начнем с подстановки в первый множитель *X* = ***A*** и *X* — *D:*

$$Y,XFY_xC_Y = 53\ A_FY_xCY + Y_fD_FY_xC_Y =$$

$$= A_FJ2YXCY + D_FJ2\ Y_xC_Y = (A_F + D_F)\ Y,YxC_Y.$$

(A_F и *Dp* как постоянные множители можно вынести за знак суммы).

Аналогично делаем подстановку в последнем множителе:

$$\{A_x+D_x)YYX\{CA + C_B) = (AP+D_x)(C_A + C_B)\ Y\ Y_x =$$

$$(ApC_A + A_FC_D + D_FC_A + D_FC_D)\ \mathbf{53}\ Yx\text{-}$$

Подставляя в *Yx* значения *A»D* для *Y* и *X,* имеем:

$$(A_AC_A+ApC_D+DpC_A+D_FCA+DpC_D)(A_A+A_D+D_A+D_D)=$$

$$= \{ApC_A + A_FC_D + D_FC_A + D_FC_D)A_D =$$

$$= (\text{так как } A_A = D_A = D_D = 0) = A_FC_AA_D +$$

$$+ A_FC_DA_D + D_FC_AA_D + D_FC_DA_D = D_FC_AA_D.$$

Поскольку $A_ACAA_B = 0$, $A_AC_BAD = 0$ (у двух мальчиков не может быть одна и та же фамилия *D),* тогда D_FAvC_A — 1.

Ответ: D имеет фамилию *F; A* имеет фамилию *D; C* имеет фамилию *A; F* имеет фамилию *C.*

Второе решение. Надо решить уравнение *XpYxCy* = 1, в котором видно, что *X* и У могут получить только значения *A* и *D.*

Так как произведение $X_AYXCY = 1$, то все множители равны единице и *Yx* — 1. Подстановка вместо *X* и *Y* букв *A* и *D* может дать только два результата: ***АЦ*** и D_A (D_A = 0 по условию). Остается единственная возможность *Ap* = 1, т.е. в уравнении $Y_x = 1$, *Y = A, X = D.*

Задача 16. Семья, состоящая из отца *A,* матери *B* и трех дочерей *C, D* и *E,* купила телевизор. Условились, что в первый вечер будут смотреть передачи в таком порядке:

1) когда отец *A* смотрит передачу, то мать *B, я* гает то же;

2) дочери ***D*** и ***E,*** обе или одна из них, смотря| [ередачу;

3) из двух членов семьи — мать *B* и дочь ***C*** *л* смотрит передачу одна и только одна;

4) дочери ***C*** и ***D*** или обе смотрят, или обе не с |отрят;

5) если дочь ***E*** смотрит передачу, то и отец J и дочь ***D*** делают то же.

Кто из членов семьи в этот вечер смотрел пера **1чу?**

Решение. Обозначим буквами *A, B,* С, *Щ* (предположения): что *A, B* и т.д. смотрят перед! ***E*** высказывания ***D, E*** означают, что *A* и *B* и т.д. не смотрят. *г,* тогда ***A, B, C,***

Имеем:

1. На основе первого условия имеет место одна **13** возможностей: либо отец и мать вдвоем смотрят, либо отец не <отрит, либо оба не_смотрят, т.е. $AB + AB + \sim ХБ = 1$; но $AB + A\sim B = AB +$, $+ A(B + B) = AB + 1 = 1$. Так как В + 7J =1, им! 1м уравнение

$$A\sim + AB = 1. \tag{12}$$

2. Из второго условия имеем:

$$DE + DE + DE = 1; \qquad E\{D + Ë) + DE = i|p + DE,$$

т.е.

$$E + DE \text{ -1.} \tag{13}$$

3. Третье условие дает уравнение

$$BC + BC = 1. \tag{14}$$

4. Четвертое условие:

$$DC + D\sim U = 1. \tag{15}$$

5. Из пятого условия вывод: если дочь ***E*** смо1 ит передачу, то имеем $E\ A\ D = 1$; если же ***E*** не смотрит, то остаются **13МОЖНОСТИ,** что ***A*** и ***D*** оба или смотрят, или оба не смотрят, или см >ит только отец ***A,*** или только сестра ***D.***

Имеем:

$$EAD + EAD + EAD + EATJ + EAD - EAD + Ë\ A + A) =$$

$$= EAO + Ш + Щ = EAD + Ë,$$

отсюда

$$\overline{EAD + E} = 1. \tag{16}$$

Использование второго распределительного закона дает возможность упростить уравнения (12) и (13).

Все пять уравнений должны удовлетворяться одновременно, поэтому

$$(3 + AB)(E + DE)(BC + BC)\{DC + DC)(EAD + Ë) = 1.$$

Раскрытие скобок можно произвести в любом порядке:

а) перемножаем почленно (13) и (16):

$$E + D\,E = 1, \qquad Ë + EAD = 1, \qquad EË + DE + EAD + EADE = 1;$$

если учесть, что $EE = 0$, то остается

$$DE + EAD = 1; \tag{17}$$

б) перемножаем почленно (14) и (15):

$$BC\sim + BC = 1, \qquad DC + DC = 1,$$

$$BVDC + BVD + BCD + BCDC = 1,$$

или

$$\overline{BCË + BCD = 1}; \tag{18}$$

в) перемножив (17) и (12), получим:

$$DE + EAD = 1, \qquad A + AB = 1.$$

Имеем

$$DEA + DEAB + EAD\sim A + EADB = 1; \qquad EAD\sim A\sim = 0;$$

$$ADE + ABDE + ABDE = ADE + ABD(E + E) = ADE + ABD.$$

Отсюда

$$\overline{ABD + ADE} = 1; \tag{19}$$

г) перемножив почленно (18) и (19), получим:

$$BCD + BCD = 1, \qquad ABD + ADE = 1,$$

имеем

$$BCËAD + BCDAB + BVEADE + BCD'JE = 1,$$

или $\sim KBCDE = 1$.

Ответ: Передачу смотрели только дочери ***C H D.***

15.4. Логические основы ПЭЕ М

Логическая функция — это логическо1 выражение, состоящее из логических переменных, связанных межД собой с помощью операций алгебры логики.

Функция может принимать в зависимости от значений переменных x_p только два значения 0 и 1. Для функции га пер менных $x_{n\_i}, \ldots,$ хо будем использовать общее обозначение $F(v) = x_{n\_i}, \ldots, xo)$, где $v = (x_{n\_i}, \ldots, xo)$, каждая переменная x_p $(p = (., 2, \ldots, n)$ может принимать только два значения 0 и 1. Поэтом число всех возможных комбинаций значений $x_{n\_}$ хо ко' 1ечно и равно 2^n. Областью определения функции га переменных : $\_1, \ldots,$ Хо является совокупность точек га-мерного пространств **1,** причем каждая из точек задается определенной комбинацией зн нений этих переменных $x_{„\_i} = e_{n\_i}$, **Хэт — 6**р. хо = ео где e_p — О или 1 $(p = 0, 1, 2, \ldots, n-1)$.

Функции п переменных могут зависеть не с всех переменных $x_{„\_i}, \ldots,$ хо. Такие функции называются *вырожд ннными.*

Также функция может быть задана как во всех т< чках определения, так и не во всех:

• функция га переменных *f(v)* называется *полн устью определенной,* если ее значения *f(vj)* = О или 1 заданы во jcex 2^n точках V* области определения;

• если же значение функции не задано хотя бь в одной точке *Vi,* то она называется *не полностью определенной.* Это означает, что функция в этой точке может иметь значение 1 или такое значение будем называть коэффициентом *с;*

• если значения функции не заданы во всех •очках *Vi,* то она называется *полностью неопределенной.*

Аппарат алгебры логики широко используется п)и описании работы контактных схем и цифровых машин. При проектировании таких схем на основе анализа условий работы схемы coci, авляются логические функции, описывающие работу схемы. Нали позволяет изучить разнообразные свойства самой сэ|емы и в ряде случаев заменить ее более простой эквивалентной схек >й

Введем следующие обозначения для высказываг ий:

x — контакт *x* замкнут;

x — контакт *x* не замкнут.

Рассмотрим участок цепи (рис. 15.2) с последовательно расположенными контактами *хну.*

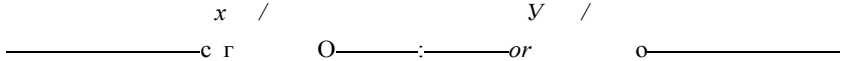


Рис. 15.2. Последовательное расположение контактов

Этот участок цепи будет замкнут тогда и только тогда, когда одновременно замкнуты контакты х и *у.* Эта ситуация в алгебре логики описывается конъюнкцией высказываний, т.е. *х А у.*

В случае когда контакты х и *у* подключены параллельно друг другу (рис. 15.3), участок цепи будет замкнут, когда по крайней мере, один из контактов замкнут. Эта ситуация в алгебре логики описывается дизъюнкцией высказываний, т.е. х V *у.*

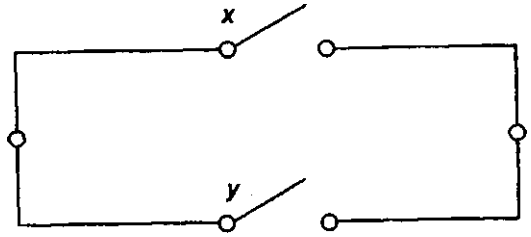


Рис. 15.3. Параллельное расположение контактов

П р и м е р . Составить функцию, соответствующую контактной схеме, изображенной на рис. 15.4.

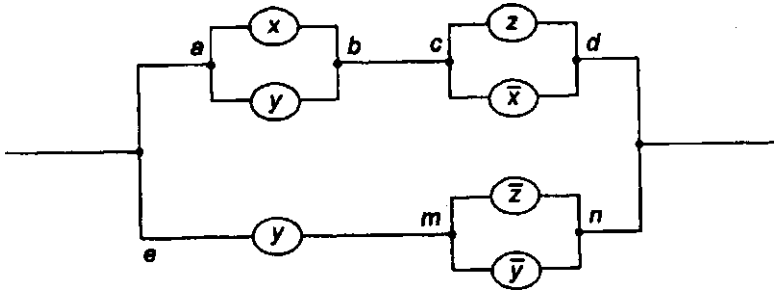


Рис 15.4. Контактная схема

Решение. На участках *ab, cd, mn* контакты включены параллельно друг другу, следовательно, эти участки описываются соответствующими дизъюнкциями: *xVy,zVx,zVy.* Так как *ab* и *cd* соединены последовательно, то участку *ad* соответствует конъюнкция (*x* V *y)* Л *(z* V *x).* На участке *en* имеем последовательно соединенные контакт *y* и участок *mn,* следовательно, весь участок *en* описываете) конъюнкцией *y* Л *(z* V *y).* Так как участок *ad* параллельно соединен с, участком *en,* то всей схеме соответствует дизъюнкция высказываний, описывающих эти участки, т.е. (ж V *y)* Л *(z* V *x)* V *y* Л *(z* V *y).* .

Последнее выражение и представляет собой логич< скую функцию /, описывающую работу данной схемы. ,

Ответ: f(x,y,z) = (xV y) A(zVx)V y Л (zVy). ,

Функции и аргументы в алгебре логики определены на множестве {О, 1} и, следовательно, могут принимать только два значения. Различные комбинации значений аргументов называются ,*наборами.* Для каждого набора аргументов можно задать два значения' функции алгебры логики (ФАЛ), следовательно для *n* аргументов можно получить 2^n различных функций. С целью получения новых функций можно использовать принцип суперпозиции, позволяющий по; цставлять одни функции вместо аргументов в другие функции. Система ФАЛ, позволяющая получать любые, сколь угодно сложные функции, называется *функционально полной системой,* а набор элементов} реализующих данные функции, — *функционально полным набором,* или *базисом.* При построении дискретных устройств наибольшее распространение получили функции, реализующие следующие операции (рис. 15.5).

Пь Операция инверсия (отрицание>

& — Операция конъюнкция

— Операция дизъюнкция

Операция отрицание дизъюнкции

Рис. 15.5. Логические операции

15.5. Логический синтез вычислительных схем

Различают несколько способов задания ФАЛ, основными из которых являются: табличный, аналитический, координатный, графический, цифровой. При табличном способе ФАЛ задается таблицей истинности, в которой указывается, какое из двух возможных значений 0 или 1 принимает функция на каждом наборе аргументов.

Рассмотрим логический синтез (создание) вычислительных схем на примере одноразрядного двоичного сумматора, имеющего два входа (о и *Ь)* и два выхода (5" и *Р)* и выполняющего операцию сложения. Пусть работа сумматора отображается следующей таблицей:

а	*Ь*	*h(a,b) = S*	/₂(а,Ь) = Р
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

В этой таблице *fi(a,b) = S* — значение цифры суммы в данном разряде; /г(а,Ь) = *Р* — цифра переноса в следующий (старший) разряд.

Согласно правилам упрощения логических формул, можно записать:

$$5 = /\text{i}(\text{а},6) = 0\ a\ b + l\ a\ b + l\ -\ a\ b + 0\ a\ b = ab + ab,$$

$$P = /_2(\text{а},\text{Ь}) = l\ a\ b + 0a\text{-}b + 0\ a\ b + 0\text{-}ab = ab.$$

Логическая блок-схема устройства, реализующего полученную функцию, приведена на рис. 15.6.

Примечание. В ряде случаев перед построением логической блок-схемы устройства по логической функции последнюю, пользуясь соотношениями алгебры логики, следует преобразовать к более простому виду (минимизировать). Для логических схем «ИЛИ», «И» и «НЕ» существуют типовые технические схемы, реализующие их на интегральных схемах. Для построения современных компьютеров обычно применяются системы интегральных элементов, у которых с

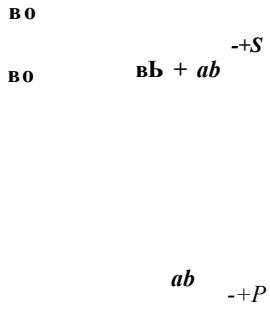


Рис. 15.6. Логическая схема устройства

целью большей унификации в качестве базовой логи ческой схемы используется всего одна из схем: «И — НЕ» (штрих Шеффера), «ИЛИ — НЕ» (стрелка Пирса) или «И — ИЛИ — НЕ» базис».

Пример 1. Для данной схемы (рис. 15.7) запишите логическое выражение.

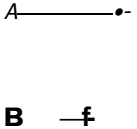


Рис. 15.7. Логическая схема

Решение. $F = (A \vee B \vee C) \wedge (B \vee A \vee C)$.

Пример 2. Для данной схемы (рис. 15.8) запишите логическое выражение.

Рис. 15.8. Логическая схема

Решение. $F = (A \vee B \vee C) \wedge (A \vee B \vee C)$.

15.6. Представление логической функции в виде графа

На практике бывает удобно изображать логическую функцию, описывающую работу какого-либо устройства, в виде дерева, где висячим вершинам поставлены в соответствие булевы переменные или их отрицания, а во внутренних вершинах указаны операции, которые надлежит выполнить над переменными или над формулами.

Пусть с некоторого устройства поступают сигналы $x, y, z, \bar{x}, \bar{y}, \bar{z}$, ($\bar{x}, \bar{y}, \bar{z}$ — инвертированные сигналы, т.е. сигналы противоположного содержания) на вычислительное устройство (ВУ), логика работы которого описывается булевой функцией $f(x,y,z) = ((x \wedge y) \vee (x \wedge y) \wedge z) \vee (y \wedge z) = xy \vee xyz \vee y\bar{z}$, причем блоки суммирования и умножения ВУ имеют только два входа.

Изображение булевой функции в виде дерева представлено на рис. 15.9.

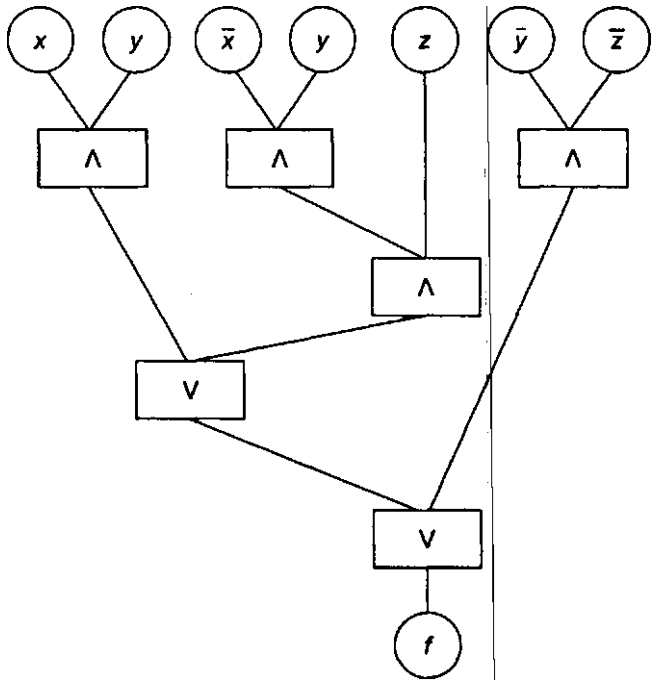


Рис. 15.9. Изображение булевой функции в виде дерева

На выходе, у основания дерева имеем значение логической функции /, которая является корнем дерева.

15.7. Проверка истинности заключений из серии посылок

Терм — это переменная или инверсия переменной.

Конъюнктивным термом (контермом) называется конъюнкция любого числа первичных термов, если каждые первичный терм с индексом *p* входит в него не более одного раза.

Дизъюнктивным термом *{диэтермом)* называется дизъюнкция любого числа первичных термов, если каждый первичный терм с индексом *p* входит в нее не более одного раза.

Пусть имеется серия посылок *A, B, C,.. .*и заключение *Z.*

Необходимо проверить, истинно или ложно заключение, сделанное из серии посылок.

В методе Вонга посылки объединяются в строку: *A,B,C,...,N=$ Z,* где знак ⇒• следует отличать от знака импликации -к

Первоначально строка приводится к правильно построенной форме (ППФ). *Правильно построенная форма* — строка, которая включает только логические операции: конъюнкцию, дизъюнкцию и отрицание.

Затем над ППФ осуществляются следующие преобразования.

• Если слева от знака стрелки стоит конъюнкция, а справа стоит дизъюнкция, то эти знаки заменяются запятыми.

Пример: *mAAB=>P, Mv Z* получаем *A, B P, M, Z.*

• Если слева от знака стрелки =Φ стоит дизъюнкция, а справа конъюнкция, то ППФ разбивается на несколько строк.

Пример: из *P* V *Q* =>• *M* получаем *P* =>• *M,Q* => *M.*

• Знак отрицания опускается, если выражение переместили в противоположную сторону от знака стрелки

Пример: из *P, Q M* получаем *Q* =$> *M,P.*

• Каждая строка должна быть доказана. Строка считается доказанной, если хотя бы одно высказывание встречается в данной строке слева и справа, т.е. если в строке имеется тавтология.

Пример: *P,M=>Z, P.*

• Если все строки доказаны, то следствие из серии посылок будет истинно, если хотя бы одна строка не доказана, то следствие ложно.

П р и м е р . Доказать истинность или ложь заключения из серии посылок: «Если курс логики не труден, то он полезен. Курс логики неинтересен или он бесполезен. Курс логики интересен. Следовательно, курс логики труден.»

Решение. Для обозначения высказываний введем переменные:

A — курс логики труден;

B — курс логики полезен;

C — курс логики интересен.

Запишем условие задачи в терминах математической логики:

$$A \to B,\quad \frac{C \vee B \to C}{A}$$

где ~*A -> B, U* V B, *C* — посылки, а *A* — заключение._

Теперь составляем строку Вонга: *A -> B, C У B, C => A.* Для приведения строки Вонга к ППФ воспользуемся следующими соотношениями эквивалентности между простейшими функциями:

$$x_i \to x_j = \bar{x}_i \vee x_j, \quad x_i \sim x_j = (x_i \wedge x_j) \vee (\bar{x}_i \wedge \bar{x}_j).$$

Отсюда получим *АУ B, C У B, C* =• А

Если слева от знака стрелки =Ф> стоит дизъюнкция, то ППФ разбивается на несколько строк:

A, C У B, C => А (доказано), *B,CУB,C=>A.*

Рассмотрим вторую недоказанную строку, в которой дизъюнкция разбивает ППФ на две строки:

B, C, C => A, B,B,C=> A.

Отсюда *B, C =>• A, C* — доказано; *B,C=$A,B* — доказано. Следовательно, из серии посылок следует, что заключение *A* является логически верным.

Контрольные вопросы

1. Что такое высказывание?
2. Чем отличается высказывание от предиката?
3. Каковы особенности алгебры высказываний?
4. Перечислите аксиомы алгебры логики.
5. Приведите примеры таблиц истинности для логических операций.
6. Что такое биекция множеств?
7. Назовите логические операции, используемые при синтезе вычислительных схем.
8. Приведите правила, используемые при решении логических уравнений.
9. Приведите правила, используемые при решении систем логических уравнений.
10. В чем заключается основная идея метода Вонга?

Библиографический список

1. **Колдаев** В.**Д.** Основы алгоритмизации и программирования: учеб. пособ. / В.Д. Колдаев; под ред. Л.Г. Гагариной. - М.: ИД "ФОРУМ": ИНФРА-М, 2006.
2. **Колдаев** В.**Д.** Сборник задач и упражнений по информатике: учеб.пособ. / В.Д. Колдаев, Е.Ю. Павлова; под ред.Л.Г.Гагариной. -М.: ИД "ФОРУМ": ИНФРА-М, 2007.
3. **Шень А.** Программирование: Теоремы и задачи / А. Шень. - М.: МЦНМО, 2004.
4. Кнут **Д.** Искусство программирования для ЭВМ / Д. Кнут. - Т. 3. Сортировка и поиск. - М.: Мир, 2000.
5. **Кормен Т.** Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. - М.: МЦНМО, 2000.
6. Внрт Н. Алгоритмы и структуры данных / Н. Вирт; пер. с англ. - М.: Мир, 2001.
7. Хусаинов **Б.С.** Структуры и алгоритмы обработки данных: примеры на языке Си: учеб. пособ. / Б.С. Хусаинов. - М.: финансы и статистика, 2004.
8. **Мейн М.** Структуры данных и другие объекты в С++ / М. Мейн, У. Савитч; пер. с англ. - М.: Издательский дом "Вильяме", 2002.
9. **Ахо А.** Структуры данных и алгоритмы: учеб. пособ. / А. Ахо, Д.Э. Хопкрофт, Д. Ульман; пер. с англ. - М.: Издательский дом "Вильяме", 2000.
10. **Бадд Т.** Объектно-ориентированное программирование в действии / Т. Бадд; пер. с англ. - СПб.: Питер, 1997.
11. **Марченко А.И.** Программирование на языке Object Pascal 2.0 / А.И. Марченко. - Киев: Юниор, 1998.
12. **Епашенников А.М.** Программирование в среде Delphi: учеб. пособ.: В 4-х ч. / А.М. Епашенников, В.А. Епашенников. - М.: Диалог-МИФИ, 1998.
13. **Подбельский** В.В. Практикум по программированию на языке Си: учеб. пособ. / В.В. Подбельский. - М.: Финансы и статистика, 2004.

Приложение 1

Системы счисления

Человеку издревле приходилось считать различные предметы и записывать их количество. Для этих целей возникла *унарная* система записи, при которой числа обозначались соответствующим количеством черточек (или засечек). Например, число 5 представлялось как 11111.

Унарная запись очень громоздкая и неудобная, поэтому были найдены более компактные способы обозначения больших чисел. Появились разные условные обозначения чисел, где использовали в качестве цифр буквы, к которым добавляли специальные значки. В табл. П1.1 указаны наиболее известные нумерации.

Таблица П1.1

Наиболее известные нумерации мира

Древнеегипетская нумерация	Древнегреческая нумерация	Вавилонская нумерация	Нумерация индейцев Майя
Старо-китайская нумерация	Славянская кириллическая нумерация	Славянская глаголическая нумерация	Латинская нумерация
Современная арабская нумерация			

Египетская нумерация. Египтяне придумали эту систему около 5000 лет тому назад. Это одна из древнейших систем записи чисел (табл. П1.2), известная человеку.

Таблица П1.2

Египетская нумерация

1 II III	1. Если палочек нужно изобразить несколько, то их изображали в два ряда, причем в нижнем должно быть столько же палочек сколько и в верхнем, или на одну больше
n n n n	10. Если нужно изобразить несколько десятков, то иероглиф повторяли нужное количество раз
	100.

Древняя греческая нумерация. В древнейшее время в Греции была распространена так называемая аттическая нумерация. В этой нумерации числа 1, 2, 3, 4 изображались соответствующим количеством вертикальных полосок. Число 5 записывалось знаком П (древнее начертание буквы «Пи», с которой начиналось слово «пять» - «пенте»).

Числа 6, 7, 8, 9 обозначались сочетаниями этих знаков: Р|, Р||, РЦ|, ГШ|. Примерно в третьем веке до нашей эры аттическая нумерация в Греции была вытеснена ионийской системой. В ней числа 1 - 9 обозначаются первыми буквами греческого алфавита:

a=1, /3 = 2, 7 = 3, *6 = 4,* e = 5, c = 6, C = 7, т? = 8, tf = 9.

Вавилонская нумерация. В древнем Вавилоне примерно за 40 веков до нашего времени создалась позиционная нумерация, т.е. такой способ записи чисел, при котором одна и та же цифра может обозначать разные числа, смотря по месту, занимаемому этой цифрой. В вавилонской поместной нумерации ту роль, которую у нас играет число 10, играет число 60, и потому эту нумерацию называют шестидесятеричной. Числа менее 60 обозначались с помощью двух знаков: У для единицы и ^ для десятка. Эти знаки повторялись нужное число раз, например

Т У Т - 3, « - 20, « < У У - 32.

Нумерация индейцев Майя. Эта нумерация очень интересна тем, что на ее развитие не повлияла ни одна из цивилизаций Старого Света. Сначала эта нумерация использовала пятеричную систему счисления, а потом ее приспособили для двадцатеричной (табл. П1.3).

Таблица П1.3

Нумерация индейцев Майя

•	1	••••	9
••	2		10
•••	3	•	11
	4	••	12
	5	•••	13

Китайская нумерация. Эта нумерация—одна из старейших и самых прогрессивных, поскольку в нее заложены такие же принципы, как и в

современную арабскую. Возникла эта нумерация около 4 тысяч лет тому назад в Китае (табл. П1.4).

Таблица П1.4

Китайская нумерация

	1	А	6
_	2	-t	7
=	3	Л	8
	4	%	9
2	5	о	0

Славянская кириллическая нумерация. Эта нумерация была создана вместе со славянской алфавитной системой для переписки священных книг для славян греческими монахами братьями Кириллом (Константином) и Мефодием в IX в. Эта форма записи чисел получила большое распространение в связи с тем, что имела полное сходство с греческой записью чисел (табл. П1.5). Православные церковные книги использовали эту нумерацию.

Таблица П1.5

Славянская кириллическая нумерация

А	1	**I**	10	Р	100
В	2	К	20	С	200
Г	3	Л	30	т	300
А	4	м	40	Y	400
6	5	н	50	Ф	500
3	6	а	60	**X**	600
3	7	0	70	+	700
н	8	п	80	*w*	800
4»	9	ч	90	Ц	900

Записывались цифры числа начиная с больших значений и заканчивая меньшими, слева направо. Запись числа, использованная славянами, аддитивная, т.е. в ней используется только сложение:

^ = 863 (800 + 60 + 3).

Для того чтобы не перепутать буквы и цифры, использовались титла— горизонтальные черточки над числами. В России славянская нумерация

сохранилась до конца XVII в. При Петре I возобладала так называемая «арабская нумерация».

Славянская глаголическая нумерация. Эта нумерация использовалась с VIII по XIII в. и была создана для записи чисел в священных книгах западных славян (табл. П1.6).

Таблица П1.6

Славянская глаголическая нумерация

	1	*V*	10	ь	100
	2	**5**	20	*о.*	200
я ?	3	«Р	30	с ю	300

Латинская (Римская) нумерация. Эта самая известная нумерация, после арабской. С ней мы достаточно часто сталкиваемся в повседневной жизни. Это номера глав в книгах, указание века, числа на циферблате часов, и т. д. Возникла эта нумерация в древнем Риме (табл. П1.7).

Таблица П1.7

Латинская (Римская) нумерация

I	1	C	100
V	5	D	500
X	10	M	1000
L	50		

Записывались цифры числа, начиная с больших значений и заканчивая меньшими, слева направо. Если цифра с меньшим значением записывалась перед цифрой с большим значением, то происходило ее вычитание. Например

CCXXXVII = 100 +100 +10 +10 + 10 + 5 +1 +1 = 237,

но

XXXIX = 10 +10+ 10+ 10 - 1 = 39.

Арабская нумерация. Это самая распространенная на сегодняшний день нумерация. Название «арабская» для нее не совсем верно, поскольку хоть и завезли ее в Европу из арабских стран, но там она тоже была не родной. Настоящая родина этой нумерации - Индия. В различных районах Индии существовали разнообразные системы нумерации, но в какой-то момент среди них выделилась одна. В ней цифры имели вид

начальных букв соответствующих числительных на древнеиндийском языке - санскрите, использующем алфавит «Деванагари»:

0 1 2 3 4 5 6 7 8 9

Системы счисления. *Система счисления* — это способ наименования и представления чисел с помощью символов, имеющих определенное количественное значение. Такие символы называются цифрами.

Алфавит системы счисления — совокупность символов, используемых в данной системе счисления.

Основание системы счисления — количество цифр, используемых в данной системе счисления.

Разряд — номер позиции в числе (нумеруются с нуля справа налево).

Вес разряда—число, равное основанию системы счисления в степени номера разряда.

Позиционные и непозиционные системы счисления

Все системы счисления подразделяются на два класса—позиционные и непозицонные.

В непозиционных системах счисления от положения цифры в записи числа не зависит величина, которую она обозначает, например римская система.

В позиционных системах счисления величина, обозначаемая цифрой в записи числа, зависит от ее позиции. Например, число 444 записано тремя одинаковыми цифрами, но каждая из них имеет свое значение: четыре сотни, четыре десятка и четыре единицы, т.е. его можно представить, как $444 = 4 \cdot 100 + 4 \cdot 10 + 4 \cdot 1$ или $444 = 4 \cdot 10^2 + 4 \cdot 10^1 + 4 \cdot 10^0$. Для записи чисел в позиционной системе счисления с основанием p нужно иметь алфавит из p цифр. Обычно для этого при $p < 10$ используют p первых арабских цифр, при $p > 10$ к десяти арабским цифрам добавляют латинские буквы.

Если требуется указать основание системы, к которой относится число, то оно записывается как нижний индекс этого числа. В системе счисления с основанием p (р-ичная система счисления) единицами разрядов служат последовательные степени числа р.

В системе счисления с основанием p в одном разряде могут стоять символы от 0 до p - 1. Так в шестнадцатеричной системе счисления в одном разряде могут стоять символы от 0 до 15 (табл. П1.8).

Примечание. К нетрадиционным системам счисления относится *фибоначчиева система счисления.* Базисом фибоначчиевой системы счисления является последовательность 1, 2, 3, 5, 8, 13, 21, 34, 55, ..., т.е. подряд идущие числа Фибоначчи. В качестве цифр в этой системе счисления используются только 0 и 1. Например: 37ю = 34 + 3 = 100000100$,; 25ю = 21 + 3 + 1 = 1000101ф.

Таблица П1.8

Системы счисления

Двоичная (основание 2)	Восьмеричная (основание 8)		Десятичная (основание 10)	Шестнадцатеричная (основание 16)	
		триады			тетрады
0	0	000	0	0	0000
1	1	001	1	1	0001
	2	010	2	2	0010
	3	011	3	3	0011
	4	100	4	4	0100
	5	101	5	5	0101
	6	110	6	6	0110
	7	111	7	7	0111
			8	8	1000
			9	9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
				F	1111

Для перевода числа из произвольной системы счисления в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа p (основания системы), и вычислить по правилам десятичной арифметики:

$$X_p = A_n P^{n-1} + A_{n-1} P^{n-2} + A_{n-2} P^{n-3} + \bullet\bullet\bullet + A_2 P^1 + A_1 P^0.$$

Пример 1. а) Число ПЮЮОО2 переведем в десятичную систему счисления:

$$111010002 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 232_{10}.$$

б) Число 75013в переведем в десятичную систему счисления:

$$75013_8 = 7 \cdot 8^4 + 5 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 + 3 \cdot 8^0 = 31243_{10}.$$

в) Число FDA1ie переведем в десятичную систему счисления:

$$FDA1ie = 15 \cdot 16^3 + 13 \cdot 16^2 + 10 \cdot 16^1 + 1 \cdot 16^0 = 64929ю.$$

г) Переведем число 703,04s из восьмеричной системы счисления в десятичную:

$$7 \cdot 8^2 + 0 \cdot 8^1 + 3 \cdot 8^0 + 0 \cdot 8^{-1} + 4 \cdot 8^{-2} = 451{,}0625_{10}.$$

д) Переведем число B2E,4ie в десятичную систему счисления:

$$B2E,4_{16} = 11 \cdot 16^2 + 2 \cdot 16^1 + 14 \cdot 16^0 + 4 \cdot 16^{-1} = 2862{,}25\text{ю}.$$

е) Переведем число 10101101,1012 в десятичную систему счисления:

$$10101101{,}101_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 173{,}625\text{ю}.$$

Перевод целых чисел из одной системы счисления в другую. Сформулируем алгоритм перевода целых чисел из системы с основанием *p* в систему с основанием *q*.

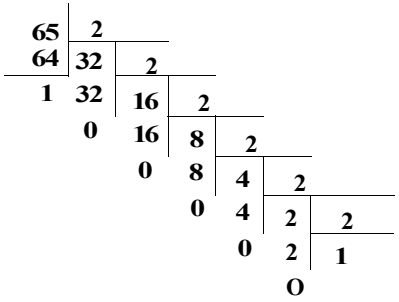
1. Основание новой системы счисления выразить цифрами исходной системы счисления и все последующие действия производить в исходной системе счисления.

2. Последовательно выполнять деление данного числа и получаемых целых частных на основание новой системы счисления до тех пор, пока не получим частное, меньшее делителя.

3. Полученные остатки, являющиеся цифрами числа в новой системе счисления, привести в соответствие с алфавитом новой системы счисления.

4. Составить число в новой системе счисления, записывая его, начиная с последнего остатка.

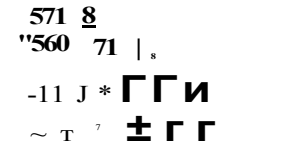
Пример 2. Переведем десятичное число 65ю в двоичную систему счисления:



Порядок записи разрядов
начиная с последнего остатка

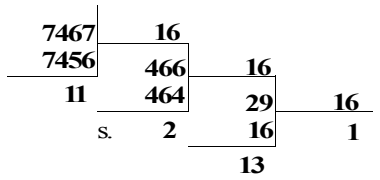
Таким образом, получаем: 65ю = 1000001_2 = 01000001_2. Обратите внимание на то, что ведущий ноль в двоичном представлении числа может быть добавлен для дополнения двоичного представления до восьми бит.

Пример 3. Переведем десятичное число 571_{10} в восьмеричную систему счисления:



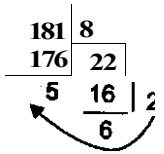
Получаем: 571ю = 1073_8.

Пример 4. Переведем десятичное число 7467_{10} в шестнадцатеричную систему счисления:



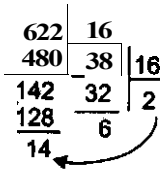
Результат: 7467ю = $1D2B_{16}$.

Пример 5. Переведем число 181ю в восьмеричную систему счисления:



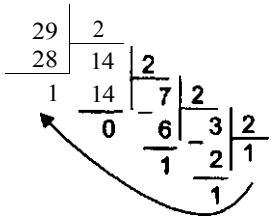
Результат: 181ю = 265_8.

Пример 6. Переведем число 622ю в шестнадцатеричную систему счисления:



Результат: 622ю = $26E_{16}$.

Пример 7. Переведем число 29ю в двоичную систему счисления:



Результат: 29ю = Ш 0 1$_2$.

Пример 8. Переведем число ЗбЗю в двоичную систему счисления. Иногда более удобно записать алгоритм перевода в форме таблицы:

Делимое	363	181	90	45	22	11	5	2	1
Делитель	2	2	2	2	2	2	2	2	2
Остаток	1	1	0	1	0	1	1	0	1

Результат: $363_{10} = 101101011_2$.

Перевод дробных чисел из одной системы счисления в другую. Сформулируем алгоритм перевода правильной дроби с основанием р в дробь с основанием *q:*

1. Основание новой системы счисления выразить цифрами исходной системы счисления и все последующие действия производить в исходной системе счисления.

2. Последовательно умножать данное число и получаемые дробные части произведений на основание новой системы до тех пор, пока дробная часть произведения не станет равной нулю или будет достигнута требуемая точность представления числа.

3. Полученные целые части произведений, являющиеся цифрами числа в новой системе счисления, привести в соответствие с алфавитом новой системы счисления.

4. Составить дробную часть числа в новой системе счисления, начиная с целой части первого произведения.

Пример 9. Переведем число 0,65625ю в восьмеричную систему счисления:

0,	65625
	V 8
5	25000
	x 8
2	00000

Результат: 0,65625ю = $0,52_8$.

Пример 10. Переведем число 0,65625ю в шестнадцатеричную систему счисления:

0,	v 65625
	16
10	v 50000
	16
8	00000

Результат: 0,65625ю = 0,A8i$_6$.

Пример 11. Переведем число 0,65ю в двоичную систему счисления:

0,	v 65
	x 2
1	*30
	x 2
0	x 60
	x 2
1	*20
	x 2
0	x 40
	x 2
0	x 80
	x 2
1	60

Результат, полученный за шесть шагов: 0,65ю = 0,101001г.

Очевидно, что этот процесс может продолжаться бесконечно, давая все новые и новые знаки в изображении двоичного эквивалента числа 0,65ю. Так, за четыре шага получаем число 0,1010г. а за пять шагов число ОДОЮОг и т.д. Такой бесконечный процесс обрывают на некотором шаге, когда считают, что получена требуемая точность представления числа.

Пример 12. Переведем десятичную дробь $0,5625_{10}$ в двоичную систему счисления.

0,	v 5625
	x 2
1	*1250
	x 2
0	v 2500
	x 2
0	v 5000
	x 2
1	0000

Результат: $0,5625_{10} = 0,1001_2$.

Пример 13. Переведем число 0,3125ю в восьмеричную систему счисления:

0,	3125 × 8	
2	5000 × 8	
4	0000	

Результат: $0,3125ю = 0,24_8$.

Перевод произвольных чисел из одной системы счисления в другую. Перевод произвольных чисел, т.е. чисел, содержащих целую и дробную части, осуществляется в два этапа. Отдельно переводится целая часть, отдельно — дробная. В итоговой записи полученного числа целая часть отделяется от дробной запятой (точкой).

Пример **14.** Переведем число 23,125ю в двоичную систему счисления:

Переводим целую часть:

23	
1	

Переводим дробную часть:

0	125 × 2
0	25 × 2
0	5 × 2
1	0

Таким образом, 23ю = Ю111_2; $0,125_{10}$: $0,001_2$, а значит, 23,125ю = = $10111,001_2$.

Пример 15. Переведем число 17,25ю в двоичную систему:

Переводим целую часть:

17	2	
1	8	2
0	4	2
0	2	2
0	1	

Переводим дробную часть:

0	25 × 2
0	5 × 2
1	0

Результат: $17,25_{10} = 10001,01_2$.

Пример 16. Переведем число 124,25ю в восьмеричную систему:

Переводим целую часть:

124	8	
4	15	8
7	1	

Переводим дробную часть:

0	25 × 8
2	0

Результат: $124,25ю = 174,2_8$.

Пример 17. Перевести восьмеричное или шестнадцатеричное число в двоичную систему счисления.

В некоторых случаях можно использовать более простые правила перевода. В трех двоичных разрядах (триаде) можно представить любую восьмеричную цифру от 0 (000) до 7 (111). Аналогично и любую шестнадцатеричную цифру можно представить четырьмя двоичными разрядами (тетрадой): от 0 (0000) до F (1111). Поэтому для перевода восьмеричного (шестнадцатеричного) числа в двоичную систему достаточно заменить каждую цифру соответствующей триадой (тетрадой):

$537,1_8 = 101\ 011\ 111,\ 001_2$; $1A3,F_{16} = 1\ 1010\ 0011,\ 1111_2$;

5 3 7 1 **1 A 3 F**

Пример **18.** Перевести двоичное число в восьмеричную и шестнадцатеричную системы счисления.

Чтобы перевести число из двоичной системы в восьмеричную, его нужно разбить на триады (тройки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую триаду нулями, и каждую триаду заменить соответствующей восьмеричной цифрой. Например, число $1101111001,1101_2$ переводится следующим образом:

$$001,101\ 011,001,110,100, = 1571,64_8.$$

1 5 7 1 6 4

Чтобы перевести число из двоичной системы в шестнадцатеричную, его нужно разбить на тетрады (четверки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую тетраду нулями, и каждую тетраду заменить соответствующей восьмеричной цифрой. Например, число $11111111011,100111_2$ переводится следующим образом:

$$0111\ 1111\ 1011,10011100 = 7FB,9C.$$

7 F B 9 C

Переведем число $10101001,10111_2$ в восьмеричную и шестнадцатеричную системы счисления:

$$10101Ш,10111_2 = \wedge\ \wedge\ \wedge\ ,\ \wedge\ П\ 0_2 = 251,56\mathbf{B};$$

2 5 1 5 6

$$10101001,10111_3 = 10101001,10111000_2 = A9,B8i_6.$$

A 9 B 8

Пример 19. Перевести число из восьмеричной в шестнадцатеричную систему.

При этом можно перевести число из восьмеричной системы в двоичную систему, а затем в шестнадцатеричную. Например, для числа $175{,}24_8$ получаем:

s ^ s ^ v ^ - " ^ v ^ x $^{-1111}$101,0101a = 01111101, 0101_2 = $7D{,}5i_6$.
001 111 101 010 100 7 D ^~5~^

Результат: $175{,}24_8 = 7D{,}5i_6$.

Двоичная арифметика

Арифметические операции над двоичными числами осуществляются при помощи алгоритма под названием «сложение в столбик». Правила выполнения арифметических действий над двоичными числами задаются таблицами двоичных сложения, вычитания и умножения (табл. П1.9). Образующийся дополнительный бит называется *битом переноса.*

Таблица П1.9

Арифметические операции над двоичными числами

Двоичное сложение	Двоичное вычитание	Двоичное умножение
0 + 0 = 0 0 + 1 = 1 1+0 = 1 1 + 1 = 10	0 - 0 = 0 1 - 0 = 1 1 - 1 = 0 10 - 1 = 1	0 0 = 0 0 1 = 0 1 0 = 0 1 1 = 1

Пример **20.** Произведем сложение чисел 55,25 и 19,5 в десятичной и в двоичной системах счисления:

Переносы	10_0	1101110_0
Первое слагаемое	55,25	110111,01
Второе слагаемое	19,50	10011,10
Сумма	74,75	1001010,11

Образующийся дополнительный бит называется *битом переноса.*

Пример **21.** Произведем сложение чисел 65 и 42 в двоичной системе счисления. Переведем эти числа в двоичную систему:

$$65ю = 01000001_2; \quad 42_{10} = 00101010_2.$$

Выполним сложение этих чисел:

Первое слагаемое	01000001
Второе слагаемое	+ 00101010
Результат	01101011

Можно убедиться в том, что ОПОЮПг = Ю7ю'.

$$0 \cdot 2^7 \pm 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$$
$$= 64 + 32 + 8 + 2 + 1: \quad 107.$$

Пример **22.** Выполнить сложение двоичных чисел.

Для X = 1101; Y = 101 имеем:

, единицы переноса

	1 *1″*
X	1101
Y	+ 101
$X + Y$	10010

Результат: 1101 + 101 = 10010.

Для X = 1101; Y = 101; Z = 111 имеем:

единицы переноса

	1 ^
	111
X	1101
Y	+ 101
Z	111
$X + Y + Z$	11001

Результат: 1101 + 101 + 111 = 11001.

Вычитание чисел в двоичной системе счисления выполняется так же, как и в десятичной. При необходимости занимается единица из следующего старшего разряда. Причем занимаемая единица равна двум единицам данного разряда. Заем единицы производится каждый раз, когда цифра в разряде вычитаемого количественно больше цифры в том же разряде уменьшаемого.

Для выполнения операции вычитания оно заменяется сложением, а в качестве второго слагаемого берется инвертированное (противоположное) число. Например, пусть надо выполнить вычитание 65 - 42. Заменим его сложением: 65 + (-42).

Все представленные выше двоичные числа имеют положительные значения, что обозначается нулевым значением самого левого (старшего) разряда. Отрицательные двоичные числа содержат *единичный бит в старшем разряде.* Для получения отрицательного двоичного числа можно использовать следующий алгоритм:

- взять соответствующее положительное число и инвертировать его биты (1 заменить на 0 и наоборот);
- к полученному числу прибавить 1.

Пример 23. Заданы двоичные числа $X = 10010$ и $Y = 101$. Вычислить $X - Y$.

При вычитании двоичных чисел в данном разряде при необходимости занимается 1 из старшего разряда. Эта занимаемая 1 равна двум 1 данного разряда. Получаем:

```
_  ioooo
     101
   01101
```

Результат: 10010 - 101 = 1101.

Пример 24. Вычесть из 65 число 42. Двоичное представление Для 42 — это 00101010, а для (-42) двоичное представление будет следующим: 11010110. Используя эти представления, получаем:

```
  65  =    01000000
(-42) =    11010110.
  23       (1)00010111
```

Пример 25. Выполнить умножение двоичных чисел 1001 • 101.

Умножение двоичных чисел производится по тем же правилам, что и десятичных с помощью таблиц двоичного умножения и сложения:

```
      101
     1001
  "*"1001
   101101
```

Результат: 1001 • 101 = 101101.

Каждое частичное произведение или равно нулю, если в соответствующем разряде множителя стоит нуль, или равно множимому, сдвинутому на один разряд влево (по отношению к предыдущему частичному произведению), если в соответствующем разряде множителя стоит единица.

Пример 26. Выполнить деление двоичных чисел: 1100,011:10,01.

Деление чисел в двоичной системе счисления производится аналогично делению десятичных чисел. При делении нецелых чисел они могут быть приведены к целым путем переноса запятой в делимом и делителе на

одинаковое число разрядов и дописывания нулей в недостающие разряды справа. В рассматриваемом случае имеем:

```
  110001,1 1001
 "1001      101,1
 _1101
  1001
   1001
  "1001
     0
```

Результат: 1100,011:10,01 = 101,1.

Прямой и обратный код

Использование кодов позволяет свести операцию вычитания чисел к арифметическому сложению кодов этих чисел. Применяются прямой, обратный и дополнительный коды чисел. Прямой код используется для представления отрицательных чисел в запоминающем устройстве ПЭВМ, а также при умножении и делении. Обратный и дополнительный коды используются для замены операции вычитания операцией сложения.

К кодам выдвигаются следующие требования:

1) разряды числа в коде жестко связаны с определенной разрядной сеткой;

2) для записи кода знака в разрядной сетке отводится фиксированный, строго определенный разряд.

Например, если за основу представления кода взят один байт, то для представления числа будет отведено 7 разрядов, а для записи кода знака один разряд.

Прямой код. Прямой код двоичного числа совпадает по изображению с записью самого числа. Значение знакового разряда для положительных чисел равно 0, а для отрицательных чисел 1.

Знаковым разрядом обычно является крайний разряд в разрядной сетке. В дальнейшем при записи кода знаковый разряд от цифровых условимся отделять запятой. Если количество разрядов кода не указано, будем предполагать, что под запись кода выделен *один байт.*

Обратный код. Обратный код для положительного числа совпадает с прямым кодом. Для отрицательного числа все цифры числа заменяются на противоположные (инвертируются), а в знаковый разряд заносится единица.

Дополнительный код. Дополнительный код положительного числа совпадает с прямым кодом. Для отрицательного числа дополнительный код образуется путем получения обратного кода и добавлением к младшему разряду единицы.

Пример 27. Пусть для записи кода выделен один байт, тогда
для числа +1101 прямой код 0,0001101;
для числа -1101 прямой код 1,0001101.

Пример 28. Пусть для записи кода выделен один байт, тогда
для числа +1101 прямой код 0,0001101; обратный код 0,0001101;
для числа -1101 прямой код 1,0001101; обратный код 1,1110010.

Пример 29. Для числа +1101 определим прямой, обратный и дополнительный коды:

Прямой код	Обратный код	Дополнительный код
0,0001101	0,0001101	0,0001101

Для числа -1101 определим прямой, обратный и дополнительный коды:

Прямой код	Обратный код	Дополнительный код
1,0001101	1,1110010	1,1110011

Экономичность системы счисления

Число в системе счисления *рек* разрядами, очевидно, будет иметь наибольшее значение в том случае, если все цифры числа окажутся максимальными, т.е. равными *р — 1.* Тогда

$$\pounds_{p}^{max} = \underbrace{(p-1)\ldots(p-1)}_{к\ цифр} = p^{к}-1.$$

Количество разрядов числа при переходе от одной системы счисления к другой в общем случае меняется. Очевидно, если *p = q" (a* необязательно целое), то $Z^{max} = ph - 1 = q''^{к} - 1$, т.е. количество разрядов числа в системах счисления *p* и *q* будут различаться в *а* раз. Очевидно соотношение:

$$a = \frac{\log p}{\log q'}$$

При этом основание логарифма никакого значения не имеет, поскольку *а* определяется отношением логарифмов,

i, Под *экономичностью* системы счисления будем понимать то количество чисел, которое можно записать в данной системе с помощью определенного количества цифр. Речь в данном случае идет не о количестве разрядов, а об общем количестве сочетаний цифр, которые интерпретируются как различные числа.

Например, пусть в нашем распоряжении имеется 12 цифр. Мы можем разбить их на 6 групп по 2 цифры (0 и 1) и получить шестиразрядное двоичное число; общее количество таких чисел равно 2^6.

Можно разбить заданное количество цифр на четыре группы по три цифры и воспользоваться троичной системой счисления - в этом случае общее количество различных их сочетаний составит 3^4. Аналогично можно произвести другие разбиения; при этом число групп определит разрядность числа, а количество цифр в группе - основание системы счисления. Результаты различных разбиений можно проиллюстрировать табл. П1.10.

Таблица П1.10

Результаты разбиений

Основание системы счисления р	Разрядность числа к	Общее количество различных чисел *N*
2	6	$2^6 = 64$
3	4	$3^4 = 81$
4	3	$4^3 = 64$
6	2	$6^2 = 36$

Из приведенных оценок видно, что наиболее экономичной оказывается троичная система счисления, причем, результат будет тем же, если исследовать случаи с другим исходным количеством цифр.

Пусть имеется *n* знаков для записи чисел, а основание системы счисления *p.* Тогда количество разрядов числа *к = n/p,* а общее количество чисел *N,* которые могут быть составлены, равно: $N = p^{n/p}$.

Если считать *N(p)* непрерывной функцией, то можно найти то значение$_{m}$, при котором *N* принимает максимальное значение. Функция имеет вид, представленный на рис. П1.1.

После преобразований получаем $\ln p = 1$, или $p_m = e$, где e = = 2,71828... — основание натурального логарифма. Ближайшее к e целое число, очевидно, 3; по этой причине троичная система счисления оказывается самой экономичной для представления чисел. В 60-х гг. в

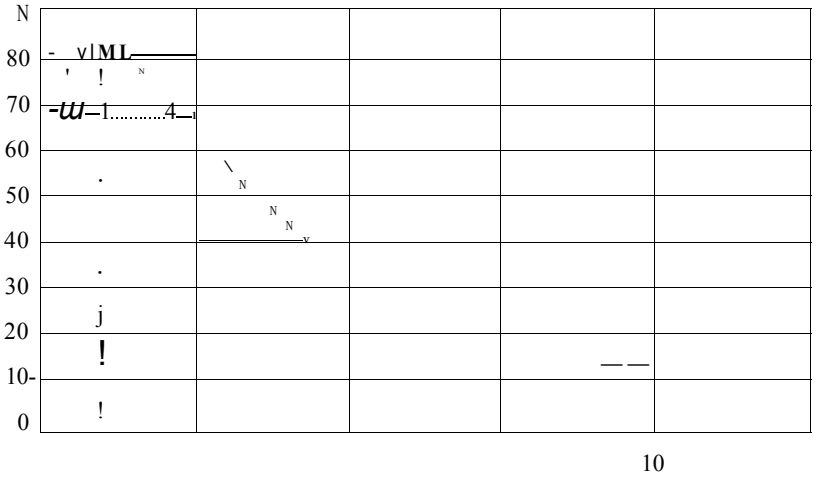


Рис. П1.1. Зависимость количества чисел от основания системы счисления

нашей стране была построена вычислительная машина «Сетунь», которая работала в троичной системе счисления. Предпочтение все же отдается двоичной системе, поскольку по экономичности она оказывается второй за троичной, а технически она реализуется гораздо проще остальных. Таким образом, простота технических решений оказывается не единственным аргументом в пользу применения двоичной системы в компьютерах.

Измерение количества информации

Информацию можно измерить количественно, т.е. подсчитать. При подобных вычислениях абстрагируются от смысла сообщения, как отрешаются от конкретности в привычных для всех нас арифметических действиях (как от сложения двух яблок и трех яблок переходят к сложению чисел вообще: 2+3).

Оценка количества информации основывается на законах теории вероятностей, точнее, определяется через вероятности событий. Сообщение имеет ценность, несет информацию только тогда, когда мы узнаем из него об исходе события, имеющего случайный характер, когда оно в какой-то мере неожиданно. Чем больше интересующее нас событие имеет случайных исходов, тем ценнее сообщение о его результате, тем больше информации.

Рассмотрим простейший случай получения информации. Вы задаете только один вопрос: «Идет ли дождь?». При этом условимся, что с одинаковой вероятностью ожидаете ответ: «Да» или «Нет». Легко увидеть, что любой из этих ответов несет самую малую порцию информации. Эта порция определяет единицу измерения информации, называемую *битом.*

Выбор единицы информации не случаен. Он связан с наиболее распространенным двоичным способом ее кодирования при передаче и обработке. Если событие имеет два равновероятных исхода, это означает, что вероятность каждого исхода равна 1/2. Такова вероятность выпадения «орла» или «решки» при бросании монеты. Информация о таком событии равна одному биту. *Бит* — минимальная порция информации, он может принимать два значения: 0 или 1.

Если событие имеет три равновероятных исхода, то вероятность каждого равна 1/3. Сумма вероятностей всех исходов всегда равна единице: ведь какой-нибудь из всех возможных исходов обязательно наступит.

Событие может иметь и неравновероятные исходы. Так, при футбольном матче между сильной и слабой командами вероятность победы сильной команды велика, например 4/5. Вероятность ничьей намного меньше, например 3/20. Вероятность же поражения совсем мала.

Количество информации — это мера уменьшения неопределенности некоторой ситуации.

Кодирование информации

Информация — произвольная последовательность символов, т.е. любое слово, каждый новый символ увеличивает количество информации. Для измерения количества информации нужен эталон. Эталоном считается слово, состоящее из одного символа двухсимвольного алфавита (цифры О или 1). Количество информации, содержащееся в этом слове, принимают за единицу, названную битом. Имея эталон количества информации, можно сравнить любое слово с эталоном. Проще сравнивать те слова, которые записаны в том же двухсимвольном алфавите.

Для определения количества информации нужно найти способ представить любую ее форму (символьную, текстовую, графическую) в едином виде. Иначе говоря, надо суметь эти формы информации преобразовать так, чтобы она получила единый стандартный вид. Таким видом стала так называемая двоичная форма представления информации. Она заключается в записи любой информации в виде последовательности только двух символов.

Благодаря введению понятия единицы информации появилась возможность определения размера любой информации числом битов. Образно говоря, если, например, объем фунта определяют в кубометрах, то объем информации — в битах. Условимся каждый положительный ответ на заданный вопрос представлять цифрой 1, а отрицательный — цифрой 0. Тогда запись всех ответов образует многозначную последовательность цифр, состоящую из нулей и единиц, например 0100.

Например, если лекция состоится, вешаем табличку с цифрой 1, если нет — с цифрой 0. В одном бите можно закодировать одно событие — свершилось или нет — или совершение одного из двух событий: есть лекция или нет лекции. Так как $2 = 2^1$, значит, для кодировки двух событий нужна одна ячейка. Рассмотрим четыре варианта:

00 — лекции нет;

01 — лекция есть;

10 — лабораторная работа;

11 — контрольная работа.

Мы видим, что для кодировки четырех событий нужны две ячейки.

Когда известно, сколько будет событий, можно выбрать необходимое количество ячеек для их хранения. Для восьми событий надо три ячейки, так как $2^3 = 8$. Для 16 событий надо четыре ячейки, так как $2^4 = 16$. В одном байте, т.е. в восьми ячейках может храниться 256 событий, так как 1 байт = 8 бит.

• Процесс получения двоичной информации об объектах исследования *пришлют кодированием информации.* Кодирование информации перечислением всех возможных событий очень трудоемко. Поэтому на практике кодирование осуществляется более простым способом. Он основан на том, что один разряд последовательности двоичных цифр имеет уже вдвое больше различных значений — 00, 01,10,11, чем одноразрядная последовательность (0 и 1). Трехразрядная последовательность имеет также вдвое больше значений — 000, 001, 010, 011, 100, 101, ПО, 111, чем двухразрядная, и т.д. Добавление одного разряда увеличивает число значений вдвое, это позволяет составить табл, П2.1 информационной емкости чисел.

Таблица П2.1

Информационная емкость чисел

Число разрядов	1	2	3	4	5	6	7	8
Количество различных значений	2	4	8	16	32	64	128	256
Число разрядов	9	10	11	12	13	14	15	16
Количество различных значений	512	1024	2048	4096	8192	16384	32768	65536

Например, для того чтобы закодировать 32 буквы русского алфавита, достаточно взять пять разрядов, потому что пятиразрядная последовательность имеет 32 различных значения. Например, русские буквы представляются восьмиразрядными последовательностями следующим образом: А —11000001, И —11001011, Я —11011101.

Перед тем как кодировать любую информацию, нужно договориться о том, какие используются коды, в каком порядке они записываются, хранятся и передаются. Это называется *языком представления информации.*

На практике довольно часто случается, что код, удобный и экономный, может исказить сообщение из-за помех, которые всегда, к сожалению, бывают в каналах связи: искажения звука в телефоне, атмосферные помехи в радио, искажение или затемнение изображения в телевидении, ошибки при передаче в телеграфе. Эти помехи, или, как их называют, шумы, обрушиваются на информацию. Поэтому для повышения надежности в передаче и обработке информации приходится вводить лишние символы — своеобразную защиту от искажений. Они — эти лишние символы, не несут действительного содержания в сообщении, они *избыточны.*

С точки зрения теории информации все то, что делает язык красочным, гибким, богатым оттенками, многоплановым, многозначным, — избыточность. Как избыточно с таких позиций письмо Татьяны к Онегину! Сколько в нем информационных излишеств для краткого и всем понятного сообщения «Я вас люблю»! И как информационно точны рисованные обозначения, понятные всем и каждому, кто входит сегодня в метро, где вместо слов и фраз объявлений висят лаконичные символьные знаки, указывающие: «Вход», «Выход».

При этом уместно вспомнить рассказ о шляпочнике, пригласившем своих друзей для обсуждения проекта вывески. Предполагалось нарисовать на вывеске шляпу и написать: «Джон Томпсон, шляпочник, делает и продает шляпы за наличные деньги». Один из друзей заметил, что слова «за наличные деньги» являются излишними — такое напоминание будет оскорбительным для покупателя. Другой нашел также лишним слово «продает», так как само собой понятно, что шляпочник продает шляпы, а не раздает их даром. Третьему показалось, что слова «шляпочник» и «делает шляпы» представляют собой ненужную тавтологию. Четвертый предложил выкинуть и слово «шляпочник» — нарисованная шляпа ясно говорит, кто такой Джон Томпсон. Наконец, пятый уверял, что для покупателя совершенно безразлично, будет ли шляпочник называться Джоном Томпсоном или иначе, и предложил обойтись без этого указания. Таким образом, в конце концов на вывеске не осталось ничего, кроме рисунка шляпы.

Конечно, если бы люди пользовались только такого рода кодами, без избыточности в сообщениях, то все «информационные формы» — книги, доклады, статьи — были бы предельно краткими, но проиграли бы в доходчивости и красоте.

Количество информации как мера уменьшения неопределенности знания

Информационные процессы — это процессы, связанные с получением, хранением, обработкой и передачей информации. В информатике рассматриваются информационные процессы, поэтому важен вопрос об определении количества информации. Количественно измерить информацию позволит подход к информации как к мере уменьшения неопределенности знания.

В окружающем нас мире существует множество явлений, которые каждый раз происходят несколько по-иному, приводят к неожиданному результату. Эти явления называют случайными. Случай играет не последнюю роль в жизни человека. Издавна существует понятие «Его Величество Случай».

Случайный эксперимент, или *опыт,* есть процесс, при котором возможны различные исходы, так что заранее нельзя предсказать, каков будет результат. Опыт характеризуется тем, что его в принципе можно повторить сколько угодно раз. Особое значение имеет множество возможных, взаимно исключающих друг друга исходов опыта (элементарных событий).

Если опыт подразделяется только на конечное число элементарных событий, которые являются к тому же равновероятными, то говорят, что речь идет о классическом случае. Примерами таких опытов являются бросания монеты, бросание игральной кости. Для опытов такого типа еще Лаплас разработал теорию вероятности (вероятность события $P(A)$ — это отношение числа элементарных событий, благоприятных для A, к числу всех возможных элементарных событий).

Пусть имеется шестигранный кубик, который будем бросать на ровную поверхность. С равной вероятностью произойдет одно из шести возможных событий — кубик окажется в одном из шести положений: выпадет одна из шести граней. Можно говорить о равновероятных событиях, если при возрастающем количестве экспериментов число выпадений каждой из граней постепенно будут сближаться. Перед самим броском возможны шесть событий, т.е. существует неопределенность нашего знания, мы не можем предсказать, сколько очков выпадет. После того как событие произошло, наступает полная определенность, так как мы получаем зрительное сообщение, что кубик в данный момент находится в определенном состоянии. Неопределенность нашего знания уменьшилась, одно из шести равновероятных событий произошло.

Начальная неопределенность нашего знания зависит от начального числа возможных равновероятных событий. Чем оно больше, тем большее количество информации будет содержать сообщение о результатах опыта.

За единицу количества информации принято такое количество информации, которое содержит сообщение, уменьшающее неопределенность знания в два раза. Такая единица названа *бит* (двоичная цифра).

Пример 1. На примере игры «Угадай число» рассмотрим уменьшение неопределенности. Один из участников загадывает целое число (например, 30) из заданного интервала (например, от 1 до 32), цель второго — «угадать» число первого участника. Для второго игрока начальная неопределенность знания составляет 32 возможных события. Чтобы найти число, необходимо получить определенное количество информации.

Первый участник может отвечать только «да» и «нет». Второй должен выбрать следующую стратегию: последовательно, на каждом шаге уменьшать неопределенность знания в два раза. Для этого он должен делить числовой интервал пополам, задавая свои вопросы (табл. П2.2).

Таблица П2.2

Вопрос второго игрока	Ответ первого игрока	Количество возможных событий (32) (неопределенность знаний)	Полученное количество информации, битов
Число больше 16?	Да	16	1
Число больше 24?	Да	8	1
Число больше 28?	Да	4	1
Число больше 30?	Нет	2	1
Число 30?	Да	1	1

Для того чтобы угадать число из интервала от 1 до 32 потребовалось пять вопросов. Количество информации, необходимое для определения одного из 32 чисел, составило 5 бит.

В 1948 г. американский математик К. Шеннон предложил формулу для вычисления количества информации для событий с разными вероятностями:

*

«=1

где / — количество информации;
к — количество возможных событий;
Pi — вероятности отдельных событий.

Как частный случай формулы Шеннона можно рассматривать формулу Хартли:

$$/ = \log_2 K = \log_2 \frac{}{P} = -\log_2 p,$$

или $K = 2^7$, где каждое из K событий имеет равновероятный исход $V = УК$.

Пример 2. Определить количество информации, получаемое при реализации события, когда бросают несимметричную четырехгранную пирамидку.

Пусть вероятность отдельных событий будет такова: $p\backslash = 1/2$; рг = = 1/4; $Рз = 1/8$; $Р4 = 1/8$. Тогда количество информации, получаемой

после реализации одного из этих событий, рассчитывается по формуле Шеннона:

$$7 = -(/Л\log_2 \frac{1}{2} + \frac{1}{4}\log_2 \frac{1}{4} + \frac{1}{8}\log_2 \frac{1}{8} + \frac{1}{8}\log_2 \frac{1}{8}) -$$

$$\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{3}{8} = \frac{14}{8}, \text{ пгів}.$$

$$- 2 + 4 + 8 + 8 - ¥ - 1,75(\text{бит}) -$$

Пример 3. В корзине лежит 16 шаров, все разного цвета. Сколько информации несет сообщение о том, что из корзины достали белый шар?

Согласно формуле Хартли, сообщение содержит / = $\log_2 16$ = 4 бита информации.

Пример 4. Определить стратегию угадывания одной карты из колоды в 32 игральные карты (без шестерок), предполагая, что на любой вопрос будет дан ответ «да» или «нет»:

Вопрос второго игрока	Ответ первого игрока	Количество возможных событий (32)	Полученное количество информации, битов
Задумана карта красной масти?	Нет	16	
Задумана карта крестовой масти?	Да	8	1
Задумана карта-картинка?	Да	4	1
Задумана дама или туз крестовой масти?	Нет	2	1
Задуман валет крестовой масти?	Нет	1	1

Ответ: задуман король крестовой масти.

Пример 5. При угадывании целого числа в диапазоне от 1 до А" было получено 7 бит информации. Чему равно *К?*

По формуле Хартли $K = 2^7 = 2^7 = 128$.

Любая информация (числа, команды, записи и т.п.) представляется в компьютере в виде двоичных кодов фиксированной или переменной длины. Отдельные элементы двоичного кода, имеющие значение 0 или 1, называют разрядами или битами. Двоичный код, состоящий из 8 разрядов, носит название байта. Для записи чисел также используют 32-разрядный формат (машинное слово), 16-разрядный формат (полуслово) и 64-разрядный формат (двойное слово).

Единицы измерения информации

1 бит — минимальная единица информации (1 бит — «О» ИЛИ «1»);
1 байт = 8 бит (1 байт — один символ);
1 килобайт (Кбайт, Kb, К) = 2^{10} байт = 1024 байга;
Одна страница машинописного текста = 2 Кбайта.
1 Мегабайт (Мбайт, Mb, М) = 2^{10} Кбайт = 1024 Кбайта = 2^{20} байт;
1 Гигабайт (Гбайт, Gb, Г) = 2^{10} Мбайт = 1024 *Ц* байта = 2^{30} байт;
1 Терабайт (Тб, Tb, Т) = 2^{10} Гбайт = 1024 Гбай1а = 2^{40} байт;
1 Петабайт = 2^{10} Тбайт = 1024 Тбайта = 2^{50} байт;
1 Эксабайт = 2^{10} Пбайт = 1024 Пбайта = 2^{60} байт.
Например:

1984 546 281 байт = 1 гигабайт 984 мегабайта 546 килобайт 281 байт

Словарь терминов

Абстрагирование — метод решения задач, при котором объекты разного рода объединяются общим понятием (концепцией). Затем сгруппированные сущности рассматриваются как элементы единой категории.

Адаптер — устройство связи компьютера с периферийными устройствами.

Адрес — номер конкретного байта оперативной памяти компьютера.

Алгебра логики (булева алгебра) — математический аппарат, с помощью которого записывают (кодируют), упрощают, вычисляют и преобразовывают логические высказывания.

Алгоритм — понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий для достижения поставленной цели за конечное число шагов.

Алгоритмический язык — совокупность символов, соглашений и правил, используемых для однозначного описания алгоритмов и обычно являющихся частью языка программирования.

Алфавит — набор знаков, в котором установлен порядок их следования (лексикографический порядок).

Анализ — метод исследования, основанный на выделении отдельных компонентов системы и рассмотрении их свойств и связей.

Аналоговая форма представления информации — представление сообщения, содержащего информацию, посредством сигналов, информационный параметр которых является непрерывной функцией времени.

Анимация — способ организации графической информации, позволяющий отображать динамические процессы.

Антивирусные программы — программы, предотвращающие заражение компьютерным вирусом и ликвидирующие последствия заражения.

Аудиоадаптер (звуковая плата) — специальная электронная плата, которая позволяет записывать звук, воспроизводить его и создавать программными средствами с помощью микрофона, наушников, динамиков, встроенного синтезатора и другого оборудования.

База данных — один или несколько файлов данных, предназначенных для хранения, изменения и обработки больших объемов взаимосвязанной информации.

База знаний — массив информационных сообщений, организованный специальным образом (в виде гипертекстовой структуры с описанием метаданных), и позволяющий ускорить поиск необходимой информации.

Базовый класс — класс, из которого порождается другой класс. Синонимы: класс-предок, надкласс, родительский класс.

Байт — группа из восьми битов, рассматриваемая при хранении данных как единое целое.

Банк данных — сумма информационных данных (как правило, мультимедийных), касающихся определенной области знаний и организованных в виде электронной библиотеки.

Банк изображений — банк данных, предназначенный для хранения фиксированных (фото, рисунки) или подвижных (видео, кино, мультипликация) изображений.

Бит — единица информации, воспринимаемая компьютером как О или 1 (единица измерения энтропии при двух возможных равновероятных исходах опыта).

Браузер — специальное программное обеспечение для просмотра информационных ресурсов в Интернете.

Вес кодовой комбинации — число ненулевых (единичных) разрядов в данной кодовой комбинации.

Видеоадаптер — электронная плата, которая обрабатывает видеоданные (текст и графику) и управляет работой монитора. Содержит видеопамять, регистры ввода-вывода и модуль BIOS.

Внешние запоминающие устройства (ВЗУ) — устройства, выполняющие операции, связанные с сохранением и считыванием данных на материальном носителе.

Гипертекст — способ непоследовательного объединения текстовой информации.

Графический язык — язык, предназначенный для написания программ машинной графики и пользования ими.

Данные — информация, представленная в определенной форме, закодированная для того, чтобы упростить ее обработку, запись или передачу.

Декодер — устройство, обеспечивающее выполнение операции декодирования, т.е. восстановления информации в первичном алфавите по полученной последовательности кодов.

Дидактические подходы — принципы и способы обучения, преподавания.

Дизайн — способ представления (описания, демонстрации) учебного материала.

Дисковод — устройство, управляющее вращением магнитного диска, чтением и записью данных на нем.

Дискретная форма представления **информации —** представление сообщения, содержащего информацию, посредством конечного числа знаков (алфавита).

Дистанционное обучение — способ организации учебного процесса с использованием образовательной среды, основанный на современных информационных и телекоммуникационных технологиях, позволяющих осуществлять обучение на расстоянии без непосредственного контакта между преподавателем и учащимся.

Документ — продукт, сформированный в результате исполнения некоторой программы.

Драйверы—программы, расширяющие возможности операционной системы по управлению устройствами ввода-вывода, оперативной памятью и т.д. С помощью драйверов возможно подключение к компьютеру новых устройств или нестандартное использование имеющихся устройств.

Знак — элемент некоторого конечного множества отличных друг от друга сущностей, используемого для представления дискретных сигналов.

Иерархия — структура, упорядоченная по подчиненности в соответствии с некоторым набором правил. **В** объектно-ориентированном программировании иерархия обычно образуется связями "класс-подкласс".

Икона — изобразительное представление, часто символическое, одной иЛи нескольких функций компьютера. Максимальный размер примерно 1 х 1,2 см.

Инкапсуляция — скрытие внутренней структуры данных и реализации методов объекта от остальной программы. Доступен только интерфейс объекта, через который осуществляется все взаимодействие с ним.

Интерактивный — качество оборудования, программ или условий эксплуатации, которое позволяет действовать в форме, приближающейся к диалогу с пользователями, или в реальном времени с компьютерами.

Интернет—всемирная сеть, состоящая из региональных компьютерных сетей. Связывает более пятидесяти миллионов человек.

Интернет-технологии — множество способов, методов, правил и протоколов для передачи данных по сетям Интернета.

Интерфейс — сопряжение двух устройств, обменивающихся информацией.

Информатика — фундаментальная естественная наука, изучающая общие свойства информации, процессы, методы и средства ее обработки (сбор, хранение, преобразование, перемещение, выдача).

Информация — сведения о лицах, предметах, фактах, событиях, явлениях и процессах независимо от формы их представления, передаваемые и хранимые с помощью условных сигналов (знаков).

Искусственный интеллект — дисциплина, относящаяся к переработке информатикой знаний и выводов.

Исполнитель алгоритма — это субъект или устройство, способные правильно интерпретировать описание алгоритма и выполнить содержащийся в нем перечень действий.

Источник информации — это субъект или объект, порождающий информацию и представляющий ее в виде сообщения.

Канал связи — это материальная среда, а также физический или иной процесс, посредством которого осуществляется передача сообщения, т.е. распространение сигналов в пространстве с течением времени.

Каталог (директория, папка) — оглавление файлов. Доступен пользователю через командный язык операционной системы. Его можно просматривать, переименовывать зарегистрированные в нем файлы, переносить их содержимое на новое место и удалять. Часто имеет иерархическую структуру.

Класс — это множество объектов, обладающих одним или несколькими одинаковыми атрибутами; эти атрибуты называются *полем свойств класса.*

Классификация — это распределение однотипных объектов в соответствии с выделенными свойствами (признаками, категориями, классами).

Код — (1) правило, описывающее соответствие знаков или их сочетаний одного алфавита знакам или их сочетаниям другого алфавита;

(2) знаки вторичного алфавита, используемые для представления знаков или их сочетаний первичного алфавита.

Кодирование перевод информации, представленной посредством первичного алфавита, в последовательность кодов.

Компрессия/декомпрессия — техника, используемая для сохранения объемов данных. Основываясь на алгоритмах, она состоит в определении излишней информации и описании ее в сравнении с эквивалентной, похожей информацией, иногда ценой потери качества. Наиболее известные нормы: JPEG для неподвижных изображений и MPEG для видео.

Компьютерная графика — оборудование и программное обеспечение для графического представления и трансформации изображений.

Контент — содержание курса. Все учебные материалы, пособия, документы, задания, тесты и контрольные мероприятия курса.

Контроллер—устройство, которое связывает периферийное оборудование или каналы связи с центральным процессором, освобождая процессор от непосредственного управления функционированием данного оборудования.

Массив — упорядоченная линейная совокупность однородных данных.

Машинный язык — совокупность машинных команд компьютера, отличающаяся количеством адресов в команде, назначением информации, задаваемой в адресах, набором операций и др.

Метаданные — короткие информационные идентификаторы (индексы, ключевые слова), предназначенные для описания смысла информационного сообщения (текста, картинки, таблицы и проч.).

Моделирование — построение упрощенного варианта прототипа, обеспечивающего приемлемую для данной задачи точность описания его строения или поведения.

Моделирование имитационное — метод исследования, основанный на том, что изучаемый прототип заменяется его имитатором (натурной или информационной моделью), с которым и проводятся эксперименты с целью получения информации об особенностях прототипа.

Модель—это объединение составных частей (элементов) и связей между ними, отражающая существенные для данной задачи свойства прототипа.

Модем — сокращения слов модулятор + демодулятор. Это оборудование передает и получает данные непосредственно по телесронным линиям.

Мультимедиа — общий термин, описывающий сумму технологий, программного обеспечения и методов, которые позволяют интегрировать данные различного происхождения (тексты, изображения, звуки и т.д.)

Навигация — набор инструментов и индикаторов системы дистанционного обучения для упрощения процесса изучения учебных материалов.

Объект — сложная структура данных, обладающая свойствами наследования, инкапсуляции и полиморфизма. Объединяет в себе данные и операции над ними (методы). Структуры данных и реализация методов объекта невидима для других объектов в системе. Объекты взаимодействуют между собой, посылая друг другу сообщения, вызывающие один из его методов. Объекты с общими свойствами и методами объединены в классы.

Оперативная память (ОЗУ) — быстрое запоминающее устройство не очень большого объема, непосредственно связанное с процессором и предназначенное для записи, считывания и хранения выполняемых программ и данных, обрабатываемых этими программами.

Операционная система — комплекс взаимосвязанных программ, предназначенных для автоматизации планирования и организации процесса обработки программ, ввода-вывода и управления данными, распределения ресурсов, подготовки и отладки программ, других вспомогательных операций обслуживания.

Основание системы счисления — количество различных знаков, используемых для изображения цифр в данной системе.

Парадигма — базовая модель конкретного способа организации информации. Объектно-ориентированная парадигма делает упор на поведении и обязанностях.

Периферия — детали и агрегаты, позволяющие вводить информацию в компьютер (клавиатура) или выводить ее (принтер).

Пиксель — наименьшая единица графики, с которой может работать компьютерная программа.

Полиморфизм — центральное понятие в объектно-ориентированном программировании, обозначающее способность объекта выбирать правильный метод (внутреннюю процедуру объекта) в зависимости от типа данных, полученных в сообщении. Благодаря полиморфизму объект выполняет нужные действия, даже если содержимое сообщения было неизвестно во время написания программы.

Порты устройств — электронные схемы, содержащие один или несколько регистров ввода-вывода и позволяющие подключать периферийные устройства компьютера к внешним шинам микропроцессора. Последовательный порт обменивается данными с процессором побайтно, а с внешними устройствами — побитно. Параллельный порт получает и посылает данные побайтно.

Постоянная память (ПЗУ) — используется для хранения данных, не требующих изменения. Содержание памяти специальным образом "зашивается" в ПЗУ при изготовлении. В ПЗУ находятся программа управления работой самого процессора, программы управления дисплеем, клавиатурой, принтером, внешней памятью, программы запуска и остановки компьютера, тестирования устройств.

Программное обеспечение — совокупность программ, выполняемых компьютером, а также вся область деятельности по проектированию и разработке программ.

Программный объект — это совокупность некоторого набора данных и процедур, определяющих возможности их изменения.

Пропускная способность канала связи — максимальное количество информации, передаваемое по каналу за единицу времени.

Процессор — центральная часть компьютера. Его задача преобразовывать, обрабатывать информацию (операции подсчета, сравнения, чтения, записи в памяти) и обеспечивать управление периферическими устройствами.

Псевдокод—система обозначений и правил, предназначенная для единообразной записи алгоритмов. Занимает промежуточное место между естественным и формальным языками.

Сайт — "место" в Интернете. Речь идет об информации в сервере, доступной **для** подключенных к глобальной сети пользователей. "Сайт" иногда заменяют словом "страница", хотя эти понятия не всегда совпадают.

Сверхоперативная память (кэш-память) — очень быстрое ЗУ малого объема. Используется для компенсации разницы в скорости обработки информации процессором и несколько менее быстродействующей оперативной памятью.

Свойство (атрибут) — качество объекта, для которого установлена мера.

Семантика — система правил истолкования отдельных языковых конструкций. Определяет смысловое значение предложений языка. Устанавливает, какие последовательности действий описываются теми или иными фразами языка и какой алгоритм определен данным текстом на алгоритмическом языке.

Сервер — высокопроизводительный компьютер с большим объемом внешней памяти, который обеспечивает обслуживание других компьютеров путем управления распределением дорогостоящих ресурсов совместного пользования (программ, данных и периферийного оборудования).

Сертификат — ин(рормационное сообщение (запись), подтверждающее успешное завершение изучения курса.

Сигнал — изменение характеристики материального носителя, которое используется для представления информации.

Синтез — (1) метод исследования (изучения) системы в целом (т.е. компонентов в их взаимосвязи), сведение в единое целое данных, полученных в результате анализа;

(2) создание системы путем соединения отдельных компонентов на основании законов, определяющих их взаимосвязь.

Система — совокупность взаимодействующих компонентов, каждый из которых в отдельности не обладает свойствами системы **в** целом, но является ее неотъемлемой частью.

Система счисления — способ записи чисел с помощью заданного набора специальных знаков (цифр).

Скрин-шот — рисунок, представляющий собой копию изображения на экране.

Сообщение — последовательность сигналов.

Средства связи — совокупность устройств, обеспечивающих преобразование первичного сообщения от источника информации в сигналы заданной физической природы, их передачу и прием.

Структура данных — перечень объединяемых одиночных данных, их характеристики, а также особенности связей между ними.

Схема — это комбинация базисных элементов, в которой выходы одних элементов присоединяются к входам других.

Телематика — группа обслуживающих технологий, имеющая характеристики информатики, с одной стороны, и обычных телеграфии и телефонии с другой. Телематика объединяет такие услуги, как телекс, видеографику, факс, телетревогу и т.д.

Телепорт — место, где принимают со спутников и передают на спутники радиоэлектронные сигналы, а затем распределяют информацию среди абонентов через кабельные сети.

Терминал — аппарат, обеспечивающий доступ на расстоянии к информационной системе с помощью передающей линии.

Файл — определенным образом оформленная совокупность физических записей, рассматриваемая как единое целое и имеющая описание в системе хранения информации.

Формальная грамматика — система правил, описывающая множество конечных последовательностей символов формального алфавита.

Форум — это инструмент для общения на сайте. Сообщения в форуме в чем-то похожи на почтовые: каждое из них имеет автора, тему и собственно содержание.

Чат — общение в интернете, когда разговор ведется в реальном времени.

Черный ящик — это система, строение которой неизвестно пользователю, однако известна ее реакция на определенные внешние воздействия.

Чип (блоха) — фамильярное название микропроцессора, так как он часто представляет собой коробочку с многочисленными "ножками". Плоские микросхемы, содержащие один контур или несколько контуров, в основном используются для банковских карточек и других подобных изделий.

Электронная почта (E-mail) — сетевая служба, позволяющая пользователям обмениваться сообщениями или документами без применения бумажных носителей.

Энтропия — мера неопределенности опыта, в котором проявляются случайные события, равная средней неопределенности всех возможных его исходов.

Эргономика — изучает условия работы и отношений между человеком и машиной.

Язык — система символов и правил, предназначенная определять задачи, которые компьютер должен решать.

ASCII — американский стандартный код обмена информацией. Широко используется для кодирования в виде байта букв, цифр, знаков операций и других компьютерных символов.

Учебное издан, ie

Гагарина Лариса Ге| надьевна
Колдаев Виктор Дм1 **[триевич**

АЛГОРИТМ! I
И СТРУКТУРЫ ***ДА*** ИНЫХ

Заведующая редакцией *Л, Табакова*
Ведущий редактор *Л.Д. 1 ригорьева*
Младший редактор *Н.А.• Редорова*
Художественный редактор *Л).И. Артюхов*
Технический редактор *Т.С, Маринина*
Корректоры *Г.В. Хлощева, Г\ м. Васильева*
Компьютерная верстка Л. *Н. (анатникова*
Обложка художника *Н.М. киксентееea*

ИБ№5248

Подписано в печать 15.04.2009 г. формат 60 х 90 У„
Печать офсетная. Гарнитур «Тайме»
Усл. п. л. 19,0. Уч.-изд. 18,32
Тираж 1500 экз. Заказ № !37/«С»026

Издательство «Финансы и статистика»
101000, Москва, ул. Пок| вка, 7
Телефон (495)625-35-02, факс 95)625-09-57
E-mail: mail@finstat.ru http:// f'ww.finstat.ru

Издательский Дом «ИН<| 1РА-М»
127282, Москва, ул. Поляр» ш, д. 31 в
Тел.: 380-05-40, 380-05-43. Фак< (495)363-92-12
E-mail: books@infra-m.ru http:j vww.infra-m.ru

Отпечатано с готовых диапозитивов в ОАС ордена «Знак Почета»
«Смоленская областная типография *т* В. И. Смирнова».
214000, г. Смоленск, проспект им. О. Гагарина, 2.