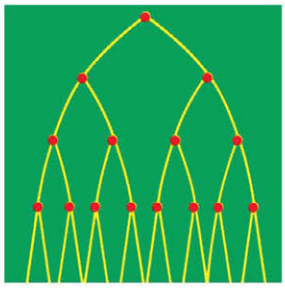
Мир программирования

Р. ХАГГАРТИ

Дискретная математика для программистов



ТЕХНОСФЕРА

МИР
программирования

Р. ХАГГАРТИ

Дискретная математика для программистов

Издание 2-е, исправленное

Перевод с английского
под редакцией С.А. Кулешова
с дополнениями А.А. Ковалева,
В.А. Головешкина, М.В. Ульянова

Допущено УМО вузов РФ по образованию в области прикладной математики в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки «Прикладная математика»

ТЕХНОСФЕРА
Москва
2012

УДК 519.854
ББК 22.176
Х13
Х13 Хаггарти Р.
Дискретная математика для программистов
Издание 2-е, исправленное

Москва: Техносфера, 2012. – 400 с., ISBN 978-5-94836-303-5

Основополагающее введение в дискретную математику, без знания которой невозможно успешно заниматься информатикой и программированием. Ни одно из многочисленных изданий по этой дисциплине, вышедших на русском языке, не читается с таким удовольствием и пользой. В доступной и весьма увлекательной форме автор рассказывает о фундаментальных понятиях дискретной математики – о логике, множествах, графах, отношениях и булевых функциях. Теория изложена кратко и иллюстрируется многочисленными простыми примерами, что делает ее доступной даже школьнику. После каждой главы (начиная со второй) рассматривается приложение описанных методов к информатике.

Дополнения в издании на русском языке посвящены актуальным задачам теории графов, рекурсивным алгоритмам, общей проблеме перебора и задачам целочисленного программирования.

Книга будет полезна студентам, изучающим курс дискретной математики, а также всем желающим проникнуть в технику написания и проверки корректности алгоритмов, включая программистов-практиков.

УДК 519.854
ББК 22.176

ISBN 978-5-94836-303-5
ISBN 0-201-73047-2 (англ.)

Содержание

Указатель обозначений

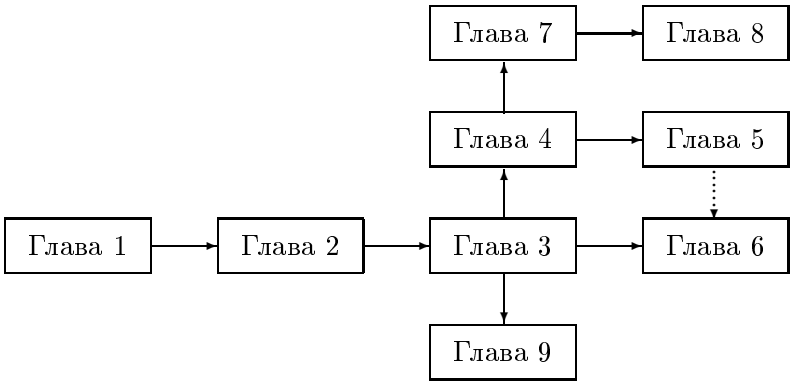
R^*	замыкание отношения R	75
E_x	класс эквивалентности элемента x	78
$x \prec y$	x — непосредственный предшественник y	80
проект	операция «проект»	87
соединение	операция «соединение»	88
выбор	операция «выбор»	89
R^{-1}	обратное отношение	91
$S \circ R$	композиция отношений R и S	92
MN	булево произведение матриц M и N	94
$f(x)$	образ элемента x	97
$f\colon A \longrightarrow B$	функция из A в B	97
$f(A)$	множество значений функции f	97
$f^{-1}\colon\longrightarrow A$	обратная функция	102
$g \circ f$	композиция функций f и g	104
$\lvert x \rvert$	модуль числа x	110
$\lfloor x \rfloor$	целая часть числа x	110
$P(n,\, k)$	число всех $(n,\, k)$-размещений без повторений	121
$C(n,\, k)$	число всех $(n,\, k)$-сочетаний без повторений	123
$O(g(n))$	класс функций, растущих не быстрее, чем $g(n)$	137
$\delta(v)$	степень вершины	143
$G = (V,\, E)$	граф с множеством вершин V и множеством ребер E	143
$c(G)$	число компонент связности	146
K_n	полный граф с n вершинами	148
МОД	минимальное остовное дерево	154
P	граф Петерсена	160
ПЕРТ	система планирования и руководства разработками	171
M^k	булево произведение k экземпляров матрицы M	176
M^*	матрица достижимости	176
$d[v]$	расстояние до вершины v	182
$\bar{p}$	отрицание булевой переменной p	195
$p \vee q$	дизъюнкция переменных p и q	195
$p \wedge q$	конъюнкция переменных p и q	195
НЕ–И	функция **НЕ–И**	200

Предисловие

Основная цель этой книги — рассказать об основной математической технике, необходимой студентам, изучающим информатику. Представленные здесь темы интересны и сами по себе, и в связи с их широкой применимостью как непосредственно в математике, так и в дисциплинах, использующих математический аппарат. В частности, формальные методы, применяемые в информатике, опираются на такие фундаментальные понятия дискретной математики, как логика, множества, отношения и функции.

Теория излагается преднамеренно кратко, а обсуждаемые здесь математические идеи вполне доступны студентам со скромной математической подготовкой. В многочисленных примерах обобщаются и развиваются ключевые идеи курса, а каждая глава, начиная со второй, снабжена приложением теории к практике. Приложения наглядно демонстрируют, как математика, о которой рассказывается в книге, решает задачи информатики. Каждая глава заканчивается набором упражнений, а чтобы поощрить читателя заниматься ими, полное решение приводится только в конце книги.

Основной материал книги появился при подготовке к чтению начального (годового) курса информатики в Оксфорде. Он рассчитан на 20 лекций. Зависимость глав друг от друга представлена на диаграмме, которая показывает, что существует некоторая свобода выбора очередности изучения материала. Это вместе с возможностью опускать отдельные приложения или заменять их альтернативными, делает книгу более гибкой и удобной для изучения.



Есть несколько доступных текстов по дискретной математике, охватывающих схожий материал. Их список для дальнейшего изучения предмета приведен в конце книги. Более продвинутые учебники по дискретной математике требуют большей математической зрелости, и я надеюсь, что читатели, успешно овладевшие содержанием настоящей книги, смогут изучать их более легко и уверенно.

Я хотел бы поблагодарить своих студентов, кто выдержал все трудности этого материала и чей рост собственных математических способностей поощрял меня писать книгу. Моя благодарность адресована также рецензентам предварительного варианта, сделавшим много полезных замечаний, и сотрудникам издательства «Pearson Education» за их усилия, предпринятые при оформлении текста. И наконец, моя признательность — супруге, за ее неизменную заботу и поддержку.

Род Хаггарти
Оксфорд
Март 2001

ГЛАВА 1

ВВЕДЕНИЕ

Дискретная математика и логика лежат в основе любого современного изучения информатики. Слово «дискретный» означает «составленный из отдельных частей», а дискретная математика имеет дело с совокупностями объектов, называемых множествами, и определенными на них структурами. Элементы этих множеств как правило изолированы друг от друга и геометрически не связаны. Действительно, большинство интересующих нас множеств конечны или, по крайней мере, счетны.

Эта область математики привлекается для решения задачи на компьютере в терминах аппаратных средств и программного обеспечения с привлечением организации символов и манипуляции данными. Современный цифровой компьютер — по существу конечная дискретная система. Понимания того, как такая машина работает, можно достигнуть, если представить машину как дискретную математическую систему. Поэтому наша главная цель при изучении дискретной математики — приобрести инструменты и технику, необходимые для понимания и проектирования компьютерных систем. Когда и как использовать эти инструменты и технику — основа раздела математики, известного как математическое моделирование.

В настоящей главе мы бросим взгляд на процесс моделирования и применим стандартный алгоритм к решению практической задачи. Выбранный пример проиллюстрирует не только вид математики, о которой идет речь в этой книге, но и ее использование при решении насущных задач. Затем мы разовьем паскалеподобный[1] псевдокод в качестве средства выражения алгоритмов, вводимых далее, для однозначной трактовки их команд.

1.1. Моделирование

Процесс математического моделирования на диаграмме можно представить так, как показано на рис. 1.1.
В качестве примера моделирования рассмотрим следующую задачу:

> *Расстояние (в милях) между шестью шотландскими городами: Абердин, Эдинбург, Форт Уильям, Глазго, Инвернесс*

[1] Pascal — язык программирования высокого уровня. — *Прим. перев.*

и Перт дано в табл. 1.1. Требуется найти дорожную сеть минимальной длины, связывающую все шесть городов.

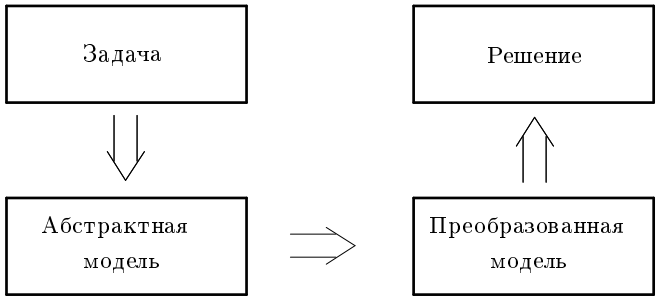


Рисунок 1.1. Схема моделирования

Сама таблица является абстрактной моделью реальной задачи. Однако для нашего решения мы преобразуем ее в геометрическую модель.

Таблица 1.1

	Абердин	Эдинбург	Форт Уильям	Глазго	Инвернесс	Перт
Абердин	—	120	147	142	107	81
Эдинбург	120	—	132	42	157	45
Форт Уильям	147	132	—	108	66	105
Глазго	142	42	108	—	168	61
Инвернесс	107	157	66	168	—	112
Перт	81	45	105	61	112	—

Мы нарисуем *граф*, чьи *вершины* обозначают города, а *ребра* — дороги их связывающие. Более подробно о графах рассказано в главе 7. Каждое ребро нашего графа, изображенного на рис. 1.2, снабжено *весом*, который означает расстояние между соответствующими городами согласно табл. 1.1.

Для решения поставленной задачи с помощью подходящего *алгоритма* (последовательности однозначных инструкций, выполнение которых влечет решение за конечное время), мы построим новый граф, имеющий минимальный общий вес, в котором все шесть городов будут соединены дорогами.

Алгоритм Прима

Шаг 1 Выберите произвольную вершину и ребро, соединяющее ее с ближайшим (по весу) соседом.

Шаг 2 Найдите не присоединенную (еще) вершину, ближе всего лежащую к одной из присоединенных, и соедините с ней.

Шаг 3 Повторяйте шаг 2 до тех пор пока все вершины не будут присоединены.

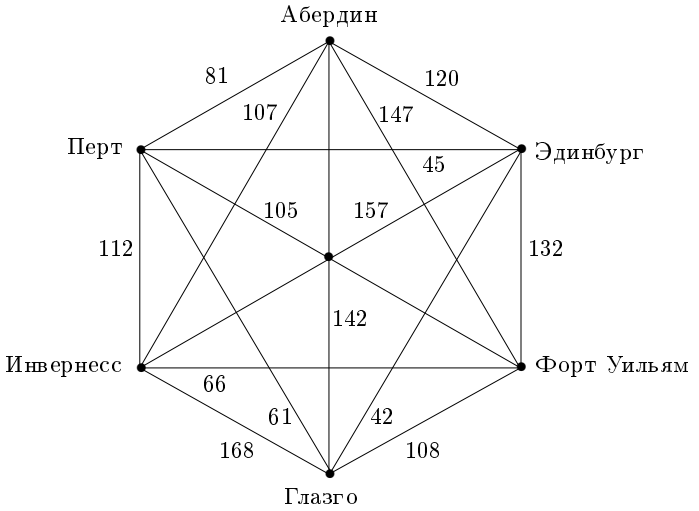


Рисунок 1.2.

На рисунках 1.3, 1.4 и 1.5 изображена последовательность графов, которая получается в результате применения алгоритма Прима, если начинать с вершины Перт. Последний граф (с общим весом 339) представляет собой минимальную сеть дорог, охватывающую все шесть городов.

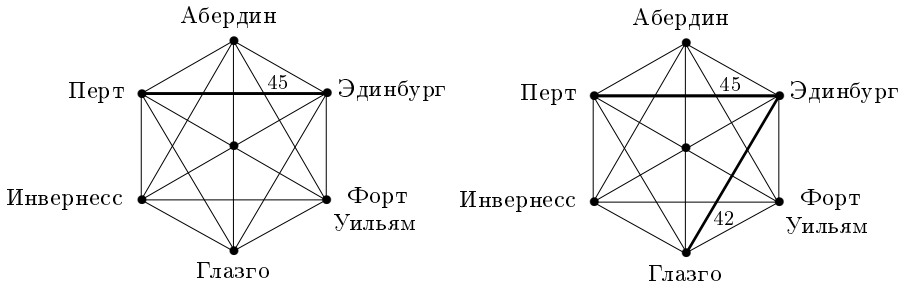


Рисунок 1.3.

Алгоритм, который мы применяли, написан на обычном русском языке. Разговорный язык может оказаться слишком многоречивым, неоднозначным и, в следствие этого, не соответствующим запутанной проблеме. Мы могли бы написать программу для компьютера,

реализующую алгоритм, но какой язык выбрать? Кроме того, язык программирования зачастую скрывает истинный смысл алгоритма от неопытного читателя! Подходящий компромисс в этой ситуации — использовать так называемый *псевдокод*, состоящий из небольшого числа структурных языковых элементов вместе с русскоподобным описанием действий реализуемого алгоритма. О нем идет речь в следующем параграфе.

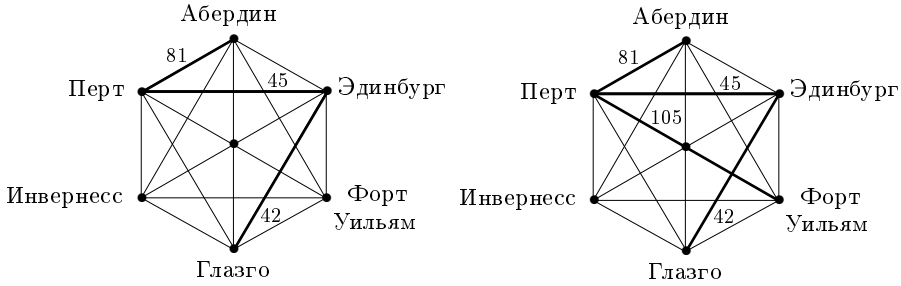


Рисунок 1.4.

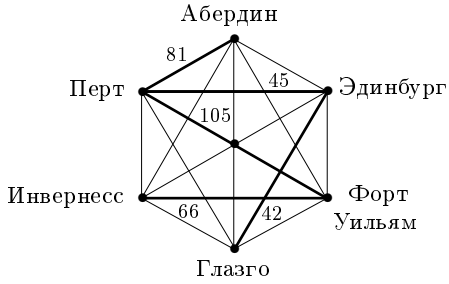


Рисунок 1.5.

1.2. Псевдокод

Мы будем использовать псевдокод, основанный на Паскале. Алгоритм в нем выглядит следующим образом.

begin
операторы исполняемых действий
операторы, управляющие порядком выполнения
end

Строительными блоками алгоритмического языка являются операторы, которые можно разбить на две категории: операторы присваивания и управляющие операторы.

Оператор присваивания приписывает переменным определенные величины и имеют такую общую форму:

имя переменной := *выражение*

Пример 1.2.1. (Алгоритм сложения двух чисел, *First* и *Second*, и присвоение результата переменной *Sum*.)

begin
Input First and Second;
Sum:= *First* + *Second;*
end

Управляющий оператор определяет порядок, в котором должны выполняться шаги алгоритма. Операторы управления бывают трех типов:

- составные операторы;
- условные операторы;
- оператор цикла.

Составные операторы представляют собой список операторов, которые должны выполняться как отдельная команда в том порядке, в котором они записаны. Составные операторы имеют следующий вид:

begin
оператор 1;
оператор 2;
.....
оператор n;
end

Пример 1.2.2. (Алгоритм обмена значений двух переменных: *One* и *Two*.)

begin
Input One and Two;
Temp := *One*;
One := *Two*;
Two := *Temp*;
end

Чтобы проследить за работой алгоритма, предположим, что начальные значения переменных *One* и *Two* равны 5 и 7 соответственно, и обратимся к табл. 1.2.

Таблица 1.2

	Temp	*One*	*Two*
Строка 1	—	5	7
Строка 2	5	5	7
Строка 3	5	7	7
Строка 4	5	7	5

Условные операторы позволяют делать выбор между двумя альтернативными ситуациями. Они записываются в виде **if-then** или **if-then-else**. На псевдокоде условные операторы изображают так:

begin
  if *условие* **then** *оператор*
end

или так:

begin
  if *условие* **then** оператор 1
    else *оператор 2*
end

Пример 1.2.3. (Алгоритм вычисления модуля числа n и присвоение результата переменной *abc*.)

begin
  Input n;
  if $n < 0$ **then** $abc := -n$
    else $abc := n$;
  Output abc;
end

В этом алгоритме оператор, стоящий во второй строке, выполняется при отрицательных значениях переменной n, а в третьей — при положительных (и нулевом). Можно написать и другой алгоритм, решающий ту же задачу, но не использующий **else**:

begin
  Input n;
  if $n < 0$ **then** $n := -n$;
  $abc := n$;
  Output abc;
end

Здесь оператор во второй строчке выполняется только при отрицательных значениях n и игнорируется при любом другом значении.

В последнем случае выполняется оператор, записанный в третьей строке.

Оператор цикла или просто *цикл* может иметь одну из форм записи:

for $X := A$ **to** Z **do** оператор; (1)

while выражение **do** оператор; (2)

repeat
оператор 1;
оператор 2;
.....
оператор n;
until условие. (3)

Здесь X — переменная, а A и Z — ее начальное и конечное значения.

В случае (1) цикл повторяется определенное число раз. Его разновидность выглядит следующим образом:

for *всех элементов множества* **do** *оператор*

В случае (2) цикл выполняется не определенное число раз, а до тех пор, пока выражение, о котором в нем идет речь, остается верным. Как только выражение становится ложным, цикл заканчивается.

И наконец, в последней ситуации (3) цикл выполняется до тех пор, пока конечное условие остается ложным. Единственное различие между (2) и (3) заключается в том, что в последнем цикл выполнится по крайней мере один раз, поскольку истинность условия в нем проверяется *после* каждого прохода цикла.

Пример 1.2.4. (Алгоритм вычисления суммы квадратов первых n натуральных чисел.)

```
begin
    sum := 0;
    for i := 1 to n do
        begin
            j := i * i;
            sum := sum + j;
        end
    Output sum;
end
```

Проследим алгоритм в случае $n = 4$, записав результаты в табл. 1.3

Таблица 1.3

	i	j	Sum
Перед выполнением цикла	—	—	0
Первый проход цикла	1	1	1
Второй проход цикла	2	4	5
Третий проход цикла	3	9	14
Четвертый проход цикла	4	16	30

Выводимый результат: $sum = 30$.

Пример 1.2.5. (Алгоритм выделения графа с минимальным общим весом, связывающего все вершины в данном связном взвешенном графе.)

```
begin
    v := произвольная вершина;
    u := ближайшая соседняя вершина;
    связать v и u;
    while остаются неприсоединенные вершины do
        begin
            u := неприсоединенная вершина, ближайшая
                 к одной из присоединенных вершин;
            соединить u с ближайшей
            из присоединенных вершин;
        end
end
```

Это — написанная на псевдокоде версия алгоритма Прима, с которым мы познакомились ранее.

Замечание. *Связным называется такой граф, в котором существует путь (по ребрам) между любыми двумя вершинами (подробнее об этом см. главу 7, стр. 146).*

Превращение алгоритма в работающую программу — дело программирования или курса структуры данных, поэтому мы не будем обсуждать этот процесс в нашей книге. Однако мы познакомимся со множеством алгоритмов, некоторые из которых представлены в форме псевдокода, а другие оформлены как математические теоремы. Доказательство истинности теорем — необходимая и далеко нетривиальная часть математического процесса. Аналогично необходимо проверять корректность написанного на псевдокоде алгоритма. Например, откуда мы можем знать, что алгоритм из примера 1.2.5 действительно дает минимальную сеть дорог?

В том случае, когда есть несколько различных алгоритмов, решающих одну и ту же задачу, возникает вопрос: какой из них является более эффективным? В упражнении 1.5 приведен еще один алгоритм, суммирующий квадраты натуральных чисел (как и в примере 1.2.4). Оба работают. Но какой это делает быстрее, использует при этом меньше памяти? Короче говоря, какой из этих алгоритмов является наилучшим?

Обе эти проблемы: корректности и эффективности алгоритмов — будут обсуждаться в последующих главах после того, как мы освоим необходимый для этого аппарат дискретной математики.

Набор упражнений 1

1.1. Граф на рисунке рис. 1.6 изображает сеть дорог, связывающих семь деревень. Расстояние между деревнями задано в милях. Используя алгоритм Прима, найдите сеть дорог минимальной общей длины, охватывающую все деревни.

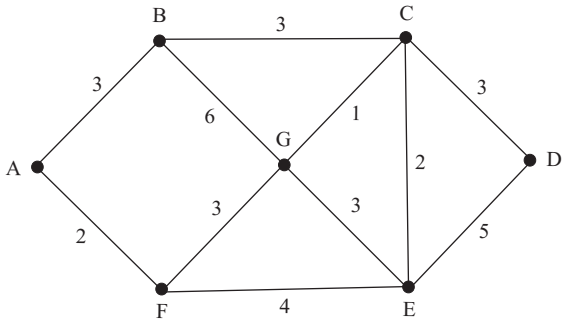


Рисунок 1.6.

1.2. Найдите результаты вычислений следующего алгоритма в случаях

(а) $n = 3$;

(б) $n = 5$.

```
begin
    f := 1;
    Input n;
    for i := 1 to n do
        f := f * i;
    Output f;
end
```