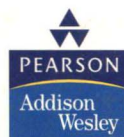
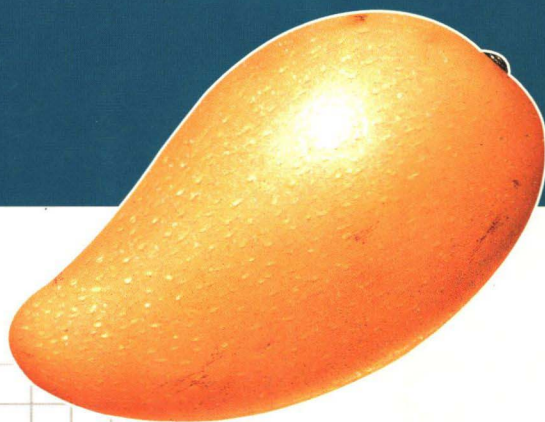


大学计算机教育国外著名教材系列 (影印版)



DATA STRUCTURES AND ALGORITHMS

数据结构与算法



Alfred V. Aho
John E. Hopcroft 著
Jeffrey D. Ullman

清华大学出版社

DATA STRUCTURES AND ALGORITHMS 数据结构与算法

内 容 简 介

本书是由计算机科学研究和教学的三位大师编写的，主要阐释了数据结构和算法两大部分，内容包括数据结构的各种基本概念，如数组、列表、栈、队列、映射、迭代、树、有向图与无向图等，以及各种算法的概念与方法，如排序、搜索、外存与内存管理等。对各种算法都给出了详细的示例和插图。本书出版20多年以来，仍然是国内外数据结构与算法课程中推荐使用最广的教材，是一本经受了时间考验的经典之作。

本书概念讲解清楚，逻辑性强，可作为相关课程的教材或参考书，也可供从事计算机工程的技术人员参考。

For sale and distribution in the People's Republic of China
exclusively(except Taiwan, Hong Kong SAR and Macao SAR).

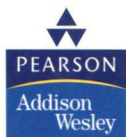
仅限于中华人民共和国境内（不包括中国香港、
澳门特别行政区和中国台湾地区）销售发行。

ISBN 7-302-07564-6

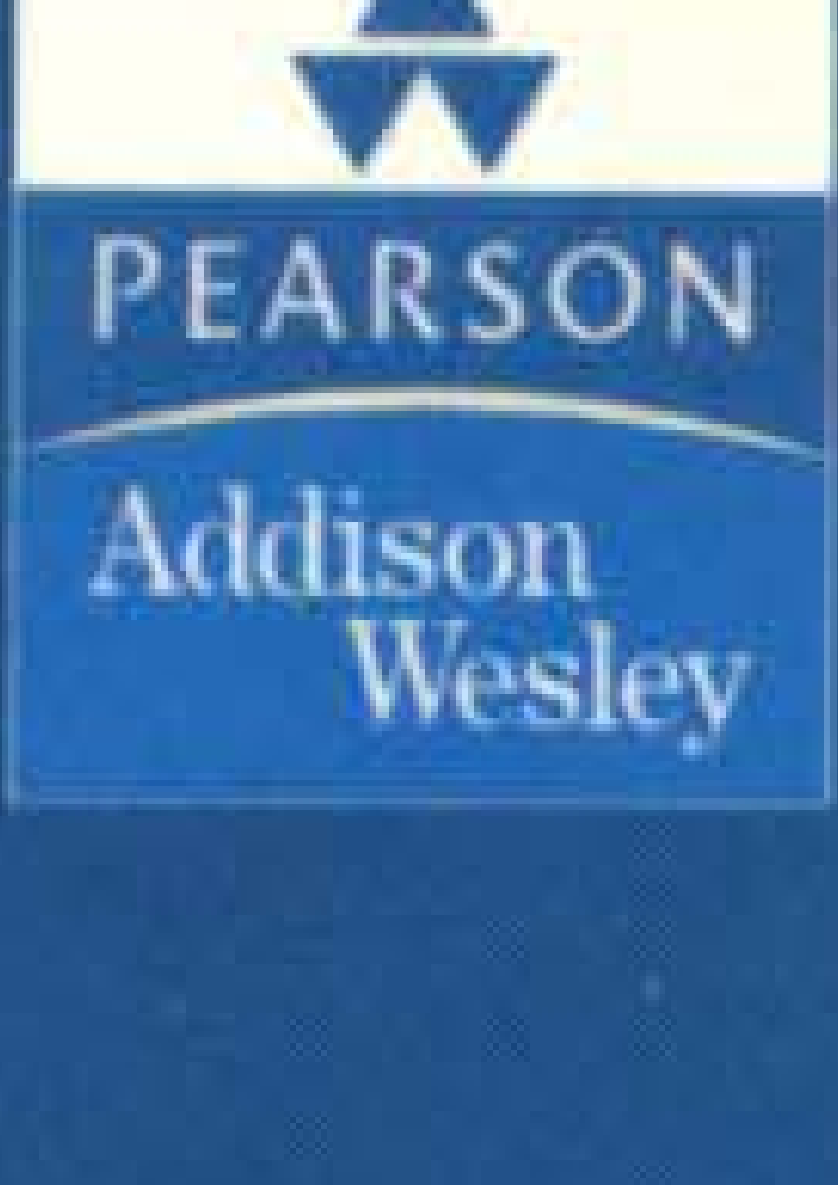


9 787302 075646 >

定价：40.00元



<http://www.pearsoned.com>



影印版

DATA STRUCTURES
AND ALGORITHMS

数据
结构
与
算法

清华大学

大学计算机教育国外著名教材系列（影印版）

Data Structures and Algorithms

数据结构与算法

Alfred V. Aho

Bell Laboratories Murray Hill, New Jersey

John E. Hopcroft

Cornell University Ithaca, New York

Jeffrey D. Ullman

Stanford University Stanford, California

清华大学出版社

北 京

English reprint edition copyright © 2003 by PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.

Original English language title from Proprietor's edition of the Work.

Original English language title: Data Structures and Algorithms, by Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, Copyright © 1983

All Rights Reserved.

Published by arrangement with the original publisher, Addison-Wesley, publishing as Addison-Wesley.

This edition is authorized for sale and distribution only in the People's Republic of China (excluding the Special Administrative Region of Hong Kong, Macao SAR and Taiwan).

本书影印版由 Pearson Education (培生教育出版集团) 授权给清华大学出版社出版发行。

**For sale and distribution in the People's Republic of China exclusively
(except Taiwan, Hong Kong SAR and Macao SAR).**

仅限于中华人民共和国境内(不包括中国香港、澳门特别行政区和中国台湾地区)销售发行。

北京市版权局著作权合同登记号 图字: 01-2003-3539

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签, 无标签者不得销售。

图书在版编目(CIP)数据

数据结构与算法 = Data Structures and Algorithms / [美]阿霍 (Aho, A. V.), [美]霍普克罗夫特 (Hopcroft, J. E.), [美]厄尔曼 (Ullman, J. D.) 著. —影印本. —北京: 清华大学出版社, 2003

(大学计算机教育国外著名教材系列)

ISBN 7-302-07564-6

I. 数… II. ①阿… ②霍… ③厄… III. ①数据结构—高等学校—教材—英文 ②算法分析—高等学校—教材—英文 IV. TP311.12

中国版本图书馆 CIP 数据核字 (2003) 第 100712 号

出版者: 清华大学出版社
http://www.tup.com.cn
社总机: (010) 6277 0175

地 址: 北京清华大学学研大厦
邮 编: 100084
客户服务: (010) 6277 6969

责任编辑: 周维焜

印刷者: 清华大学印刷厂

装订者: 三河市新茂装订有限公司

发行者: 新华书店总店北京发行所

开 本: 185×230 印张: 27.5 插页: 1

版 次: 2003 年 12 月第 1 版 2003 年 12 月第 1 次印刷

书 号: ISBN 7-302-07564-6/TP · 5563

印 数: 1~5000

定 价: 40.00 元

出版说明

进入 21 世纪, 世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的争夺。谁拥有大量高素质的人才, 谁就能在竞争中取得优势。高等教育, 作为培养高素质人才的事业, 必然受到高度重视。目前我国高等教育的教材更新较慢, 为了加快教材的更新频率, 教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始, 与国外著名出版公司合作, 影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书, 受到了国内读者的欢迎和支持。跨入 21 世纪, 我们本着为我国高等教育教材建设服务的初衷, 在已有的基础上, 进一步扩大选题内容, 改变图书开本尺寸, 一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材, 组成本套“大学计算机教育国外著名教材系列(影印版)”, 以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材, 以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好, 更适合高校师生的需要。

清华大学出版社
2002 年 10 月

Preface

This book presents the data structures and algorithms that underpin much of today's computer programming. The basis of this book is the material contained in the first six chapters of our earlier work, *The Design and Analysis of Computer Algorithms*. We have expanded that coverage and have added material on algorithms for external storage and memory management. As a consequence, this book should be suitable as a text for a first course on data structures and algorithms. The only prerequisite we assume is familiarity with some high-level programming language such as Pascal.

We have attempted to cover data structures and algorithms in the broader context of solving problems using computers. We use abstract data types informally in the description and implementation of algorithms. Although abstract data types are only starting to appear in widely available programming languages, we feel they are a useful tool in designing programs, no matter what the language.

We also introduce the ideas of step counting and time complexity as an integral part of the problem solving process. This decision reflects our long-held belief that programmers are going to continue to tackle problems of progressively larger size as machines get faster, and that consequently the time complexity of algorithms will become of even greater importance, rather than of less importance, as new generations of hardware become available.

The Presentation of Algorithms

We have used the conventions of Pascal to describe our algorithms and data structures primarily because Pascal is so widely known. Initially we present several of our algorithms both abstractly and as Pascal programs, because we feel it is important to run the gamut of the problem solving process from problem formulation to a running program. The algorithms we present, however, can be readily implemented in any high-level programming language.

Use of the Book

Chapter 1 contains introductory remarks, including an explanation of our view of the problem-to-program process and the role of abstract data types in that process. Also appearing is an introduction to step counting and "big-oh" and "big-omega" notation.

Chapter 2 introduces the traditional list, stack and queue structures, and the mapping, which is an abstract data type based on the mathematical notion of a function. The third chapter introduces trees and the basic data structures

that can be used to support various operations on trees efficiently.

Chapters 4 and 5 introduce a number of important abstract data types that are based on the mathematical model of a set. Dictionaries and priority queues are covered in depth. Standard implementations for these concepts, including hash tables, binary search trees, partially ordered trees, tries, and 2-3 trees are covered, with the more advanced material clustered in Chapter 5.

Chapters 6 and 7 cover graphs, with directed graphs in Chapter 6 and undirected graphs in 7. These chapters begin a section of the book devoted more to issues of algorithms than data structures, although we do discuss the basics of data structures suitable for representing graphs. A number of important graph algorithms are presented, including depth-first search, finding minimal spanning trees, shortest paths, and maximal matchings.

Chapter 8 is devoted to the principal internal sorting algorithms: quick-sort, heapsort, binsort, and the simpler, less efficient methods such as insertion sort. In this chapter we also cover the linear-time algorithms for finding medians and other order statistics.

Chapter 9 discusses the asymptotic analysis of recursive procedures, including, of course, recurrence relations and techniques for solving them.

Chapter 10 outlines the important techniques for designing algorithms, including divide-and-conquer, dynamic programming, local search algorithms, and various forms of organized tree searching.

The last two chapters are devoted to external storage organization and memory management. Chapter 11 covers external sorting and large-scale storage organization, including B-trees and index structures.

Chapter 12 contains material on memory management, divided into four subareas, depending on whether allocations involve fixed or varying sized blocks, and whether the freeing of blocks takes place by explicit program action or implicitly when garbage collection occurs.

Material from this book has been used by the authors in data structures and algorithms courses at Columbia, Cornell, and Stanford, at both undergraduate and graduate levels. For example, a preliminary version of this book was used at Stanford in a 10-week course on data structures, taught to a population consisting primarily of Juniors through first-year graduate students. The coverage was limited to Chapters 1-4, 9, 10, and 12, with parts of 5-7.

Exercises

A number of exercises of varying degrees of difficulty are found at the end of each chapter. Many of these are fairly straightforward tests of the mastery of the material of the chapter. Some exercises require more thought, and these have been singly starred. Doubly starred exercises are harder still, and are suitable for more advanced courses. The bibliographic notes at the end of each chapter provide references for additional reading.

Acknowledgments

We wish to acknowledge Bell Laboratories for the use of its excellent UNIXTM-based text preparation and data communication facilities that significantly eased the preparation of a manuscript by geographically separated authors. Many of our colleagues have read various portions of the manuscript and have given us valuable comments and advice. In particular, we would like to thank Ed Beckham, Jon Bentley, Kenneth Chu, Janet Coursey, Hank Cox, Neil Immerman, Brian Kernighan, Steve Mahaney, Craig McMurray, Alberto Mendelzon, Alistair Moffat, Jeff Naughton, Kerry Nemovicher, Paul Niamkey, Yoshio Ohno, Rob Pike, Chris Rouen, Maurice Schlumberger, Stanley Selkow, Chengya Shih, Bob Tarjan, W. Van Snyder, Peter Weinberger, and Anthony Yeracaris for helpful suggestions. Finally, we would like to give our warmest thanks to Mrs. Claire Metzger for her expert assistance in helping prepare the manuscript for typesetting.

A.V.A.

J.E.H.

J.D.U.

Contents

Chapter 1 Design and Analysis of Algorithms

1.1	From Problems to Programs	1
1.2	Abstract Data Types	10
1.3	Data Types, Data Structures, and Abstract Data Types	13
1.4	The Running Time of a Program	16
1.5	Calculating the Running Time of a Program	21
1.6	Good Programming Practice	27
1.7	Super Pascal	29

Chapter 2 Basic Data Types

2.1	The Data Type “List”	37
2.2	Implementation of Lists	40
2.3	Stacks	53
2.4	Queues	56
2.5	Mappings	61
2.6	Stacks and Recursive Procedures	64

Chapter 3 Trees

3.1	Basic Terminology	75
3.2	The ADT TREE	82
3.3	Implementations of Trees	84
3.4	Binary Trees	93

Chapter 4 Basic Operations on Sets

4.1	Introduction to Sets	107
4.2	An ADT with Union, Intersection, and Difference	109
4.3	A Bit-Vector Implementation of Sets	112
4.4	A Linked-List Implementation of Sets	115
4.5	The Dictionary	117
4.6	Simple Dictionary Implementations	119
4.7	The Hash Table Data Structure	122
4.8	Estimating the Efficiency of Hash Functions	129
4.9	Implementation of the Mapping ADT	135
4.10	Priority Queues	135
4.11	Implementations of Priority Queues	138
4.12	Some Complex Set Structures	145

Chapter 5	Advanced Set Representation Methods	
5.1	Binary Search Trees	155
5.2	Time Analysis of Binary Search Tree Operations	160
5.3	Tries	163
5.4	Balanced Tree Implementations of Sets	169
5.5	Sets with the MERGE and FIND Operations	180
5.6	An ADT with MERGE and SPLIT	189
Chapter 6	Directed Graphs	
6.1	Basic Definitions	198
6.2	Representations for Directed Graphs	199
6.3	The Single-Source Shortest Paths Problem	203
6.4	The All-Pairs Shortest Path Problem	208
6.5	Traversals of Directed Graphs	215
6.6	Directed Acyclic Graphs	219
6.7	Strong Components	222
Chapter 7	Undirected Graphs	
7.1	Definitions	230
7.2	Minimum-Cost Spanning Trees	233
7.3	Traversals	239
7.4	Articulation Points and Biconnected Components	244
7.5	Graph Matching	246
Chapter 8	Sorting	
8.1	The Internal Sorting Model	253
8.2	Some Simple Sorting Schemes	254
8.3	Quicksort	260
8.4	Heapsort	271
8.5	Bin Sorting	274
8.6	A Lower Bound for Sorting by Comparisons	282
8.7	Order Statistics	286
Chapter 9	Algorithm Analysis Techniques	
9.1	Efficiency of Algorithms	293
9.2	Analysis of Recursive Programs	294
9.3	Solving Recurrence Equations	296
9.4	A General Solution for a Large Class of Recurrences	298
Chapter 10	Algorithm Design Techniques	
10.1	Divide-and-Conquer Algorithms	306
10.2	Dynamic Programming	311
10.3	Greedy Algorithms	321
10.4	Backtracking	324
10.5	Local Search Algorithms	336

Chapter 11 Data Structures and Algorithms for External Storage

11.1	A Model of External Computation.....	347
11.2	External Sorting	349
11.3	Storing Information in Files	361
11.4	External Search Trees.....	368

Chapter 12 Memory Management

12.1	The Issues in Memory Management.....	378
12.2	Managing Equal-Sized Blocks.....	382
12.3	Garbage Collection Algorithms for Equal-Sized Blocks	384
12.4	Storage Allocation for Objects with Mixed Sizes	392
12.5	Buddy Systems.....	400
12.6	Storage Compaction	404

Bibliography	411
---------------------------	------------

Index	419
--------------------	------------

CHAPTER 1

Design and Analysis of Algorithms

There are many steps involved in writing a computer program to solve a given problem. The steps go from problem formulation and specification, to design of the solution, to implementation, testing and documentation, and finally to evaluation of the solution. This chapter outlines our approach to these steps. Subsequent chapters discuss the algorithms and data structures that are the building blocks of most computer programs.

1.1 From Problems to Programs

Half the battle is knowing what problem to solve. When initially approached, most problems have no simple, precise specification. In fact, certain problems, such as creating a “gourmet” recipe or preserving world peace, may be impossible to formulate in terms that admit of a computer solution. Even if we suspect our problem can be solved on a computer, there is usually considerable latitude in several problem parameters. Often it is only by experimentation that reasonable values for these parameters can be found.

If certain aspects of a problem can be expressed in terms of a formal model, it is usually beneficial to do so, for once a problem is formalized, we can look for solutions in terms of a precise model and determine whether a program already exists to solve that problem. Even if there is no existing program, at least we can discover what is known about this model and use the properties of the model to help construct a good solution.

Almost any branch of mathematics or science can be called into service to help model some problem domain. Problems essentially numerical in nature can be modeled by such common mathematical concepts as simultaneous linear equations (e.g., finding currents in electrical circuits, or finding stresses in frames made of connected beams) or differential equations (e.g., predicting population growth or the rate at which chemicals will react). Symbol and text processing problems can be modeled by character strings and formal grammars. Problems of this nature include compilation (the translation of programs written in a programming language into machine language) and information retrieval tasks such as recognizing particular words in lists of titles owned by a library.

Algorithms

Once we have a suitable mathematical model for our problem, we can attempt to find a solution in terms of that model. Our initial goal is to find a solution in the form of an *algorithm*, which is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. An integer assignment statement such as $x := y + z$ is an example of an instruction that can be executed in a finite amount of effort. In an algorithm instructions can be executed any number of times, provided the instructions themselves indicate the repetition. However, we require that, no matter what the input values may be, an algorithm terminate after executing a finite number of instructions. Thus, a program is an algorithm as long as it never enters an infinite loop on any input.

There is one aspect of this definition of an algorithm that needs some clarification. We said each instruction of an algorithm must have a “clear meaning” and must be executable with a “finite amount of effort.” Now what is clear to one person may not be clear to another, and it is often difficult to prove rigorously that an instruction can be carried out in a finite amount of time. It is often difficult as well to prove that on any input, a sequence of instructions terminates, even if we understand clearly what each instruction means. By argument and counterargument, however, agreement can usually be reached as to whether a sequence of instructions constitutes an algorithm. The burden of proof lies with the person claiming to have an algorithm. In Section 1.5 we discuss how to estimate the running time of common programming language constructs that can be shown to require a finite amount of time for their execution.

In addition to using Pascal programs as algorithms, we shall often present algorithms using a *pseudo-language* that is a combination of the constructs of a programming language together with informal English statements. We shall use Pascal as the programming language, but almost any common programming language could be used in place of Pascal for the algorithms we shall discuss. The following example illustrates many of the steps in our approach to writing a computer program.

Example 1.1. A mathematical model can be used to help design a traffic light for a complicated intersection of roads. To construct the pattern of lights, we shall create a program that takes as input a set of permitted turns at an intersection (continuing straight on a road is a “turn”) and partitions this set into as few groups as possible such that all turns in a group are simultaneously permissible without collisions. We shall then associate a phase of the traffic light with each group in the partition. By finding a partition with the smallest number of groups, we can construct a traffic light with the smallest number of phases.

For example, the intersection shown in Fig. 1.1 occurs by a watering hole called JoJo’s near Princeton University, and it has been known to cause some navigational difficulty, especially on the return trip. Roads *C* and *E* are one-way, the others two way. There are 13 turns one might make at this