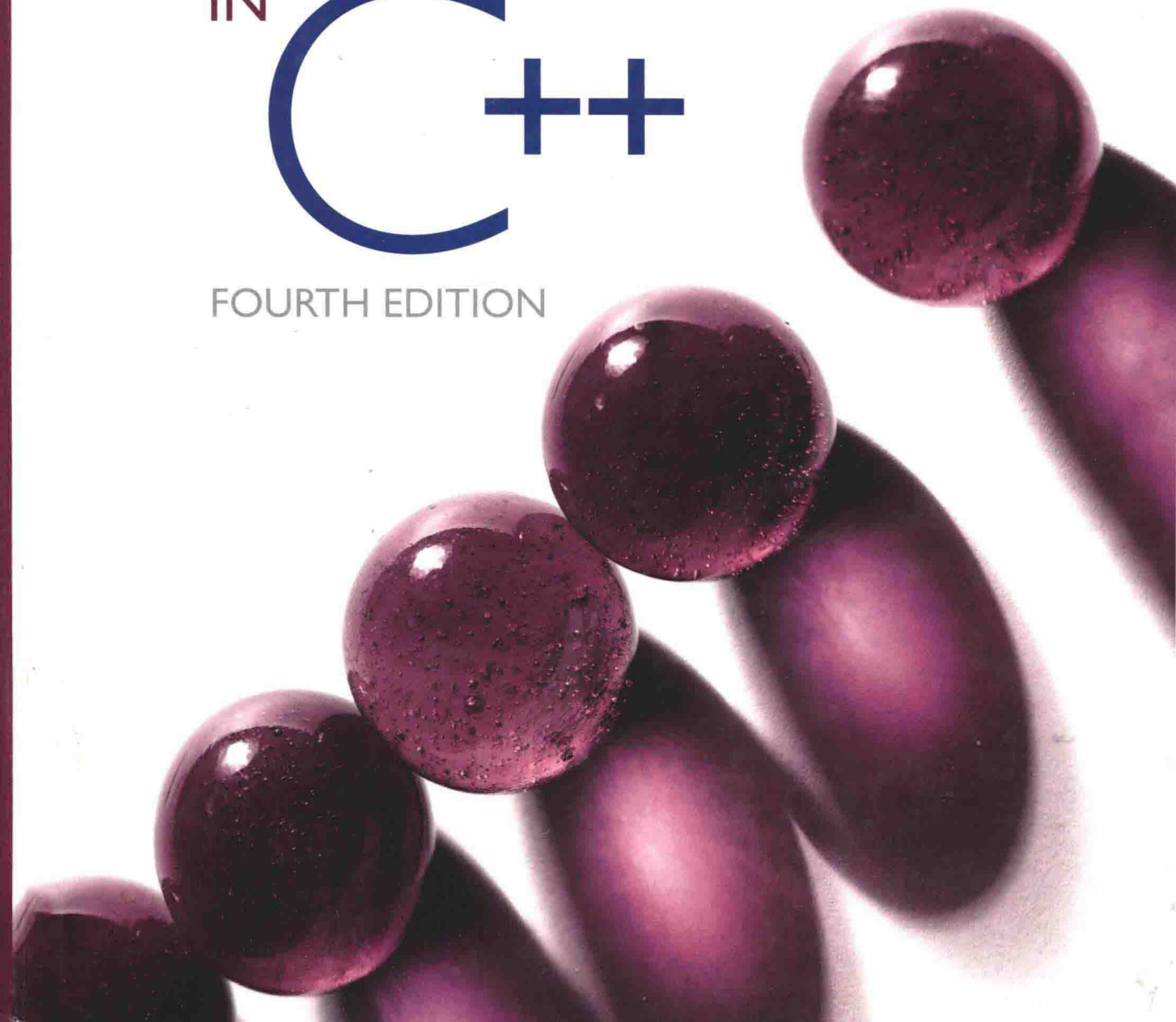


MARK ALLEN WEISS

DATA STRUCTURES AND ALGORITHM ANALYSIS IN C++

FOURTH EDITION



Fourth Edition

Data Structures and Algorithm Analysis in **C++**

Mark Allen Weiss
Florida International University



Boston Columbus Indianapolis New York San Francisco
Upper Saddle River ~~London~~ London Dubai London
Madrid Milan Munich Paris Montreal Toronto Delhi
Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore
Taipei Tokyo

Editorial Director, ECS: Marcia Horton
Executive Editor: Tracy Johnson
Editorial Assistant: Jenah Blitz-Stoehr
Director of Marketing: Christy Lesko
Marketing Manager: Yez Alayan
Senior Marketing Coordinator: Kathryn Ferranti
Marketing Assistant: Jon Bryant
Director of Production: Erin Gregg
Senior Managing Editor: Scott Disanno
Senior Production Project Manager: Marilyn Lloyd
Manufacturing Buyer: Linda Sager
Art Director: Jayne Conte

Cover Designer: Bruce Kenselaar
Permissions Supervisor: Michael Joyce
Permissions Administrator: Jenell Forschler
Cover Image: © De-kay | Dreamstime.com
Media Project Manager: Renata Butera
Full-Service Project Management: Integra Software Services Pvt. Ltd.
Composition: Integra Software Services Pvt. Ltd.
Text and Cover Printer/Binder: Courier Westford

Copyright © 2014, 2006, 1999 Pearson Education, Inc., publishing as Addison-Wesley. All rights reserved. Printed in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to 201-236-3290.

Many of the designations by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Library of Congress Cataloging-in-Publication Data

Weiss, Mark Allen.

Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition.

pages cm

ISBN-13: 978-0-13-284737-7 (alk. paper)

ISBN-10: 0-13-284737-X (alk. paper)

1. C++ (Computer program language) 2. Data structures (Computer science) 3. Computer algorithms. I. Title.

QA76.73.C153W46 2014

005.7'3—dc23

2013011064

10 9 8 7 6 5 4 3 2 1

PEARSON

www.pearsonhighered.com

ISBN-10: 0-13-284737-X

ISBN-13: 978-0-13-284737-7

Fourth Edition

Data Structures and Algorithm Analysis in C++

To my kind, brilliant, and inspiring Sara.

Purpose/Goals

The fourth edition of *Data Structures and Algorithm Analysis in C++* describes *data structures*, methods of organizing large amounts of data, and *algorithm analysis*, the estimation of the running time of algorithms. As computers become faster and faster, the need for programs that can handle large amounts of input becomes more acute. Paradoxically, this requires more careful attention to efficiency, since inefficiencies in programs become most obvious when input sizes are large. By analyzing an algorithm before it is actually coded, students can decide if a particular solution will be feasible. For example, in this text students look at specific problems and see how careful implementations can reduce the time constraint for large amounts of data from centuries to less than a second. Therefore, no algorithm or data structure is presented without an explanation of its running time. In some cases, minute details that affect the running time of the implementation are explored.

Once a solution method is determined, a program must still be written. As computers have become more powerful, the problems they must solve have become larger and more complex, requiring development of more intricate programs. The goal of this text is to teach students good programming and algorithm analysis skills simultaneously so that they can develop such programs with the maximum amount of efficiency.

This book is suitable for either an advanced data structures course or a first-year graduate course in algorithm analysis. Students should have some knowledge of intermediate programming, including such topics as pointers, recursion, and object-based programming, as well as some background in discrete math.

Approach

Although the material in this text is largely language-independent, programming requires the use of a specific language. As the title implies, we have chosen C++ for this book.

C++ has become a leading systems programming language. In addition to fixing many of the syntactic flaws of C, C++ provides direct constructs (the *class* and *template*) to implement generic data structures as abstract data types.

The most difficult part of writing this book was deciding on the amount of C++ to include. Use too many features of C++ and one gets an incomprehensible text; use too few and you have little more than a C text that supports classes.

The approach we take is to present the material in an *object-based approach*. As such, there is almost no use of inheritance in the text. We use class templates to describe generic data structures. We generally avoid esoteric C++ features and use the `vector` and `string` classes that are now part of the C++ standard. Previous editions have implemented class templates by separating the class template interface from its implementation. Although this is arguably the preferred approach, it exposes compiler problems that have made it

difficult for readers to actually use the code. As a result, in this edition the online code represents class templates as a single unit, with no separation of interface and implementation. Chapter 1 provides a review of the C++ features that are used throughout the text and describes our approach to class templates. Appendix A describes how the class templates could be rewritten to use separate compilation.

Complete versions of the data structures, in both C++ and Java, are available on the Internet. We use similar coding conventions to make the parallels between the two languages more evident.

Summary of the Most Significant Changes in the Fourth Edition

The fourth edition incorporates numerous bug fixes, and many parts of the book have undergone revision to increase the clarity of presentation. In addition,

- Chapter 4 includes implementation of the AVL tree deletion algorithm—a topic often requested by readers.
- Chapter 5 has been extensively revised and enlarged and now contains material on two newer algorithms: cuckoo hashing and hopscotch hashing. Additionally, a new section on universal hashing has been added. Also new is a brief discussion of the `unordered_set` and `unordered_map` class templates introduced in C++11.
- Chapter 6 is mostly unchanged; however, the implementation of the binary heap makes use of move operations that were introduced in C++11.
- Chapter 7 now contains material on radix sort, and a new section on lower-bound proofs has been added. Sorting code makes use of move operations that were introduced in C++11.
- Chapter 8 uses the new union/find analysis by Seidel and Sharir and shows the $O(M \alpha(M, N))$ bound instead of the weaker $O(M \log^* N)$ bound in prior editions.
- Chapter 12 adds material on suffix trees and suffix arrays, including the linear-time suffix array construction algorithm by Karkkainen and Sanders (with implementation). The sections covering deterministic skip lists and AA-trees have been removed.
- Throughout the text, the code has been updated to use C++11. Notably, this means use of the new C++11 features, including the `auto` keyword, the range `for` loop, move construction and assignment, and uniform initialization.

Overview

Chapter 1 contains review material on discrete math and recursion. I believe the only way to be comfortable with recursion is to see good uses over and over. Therefore, recursion is prevalent in this text, with examples in every chapter except Chapter 5. Chapter 1 also includes material that serves as a review of basic C++. Included is a discussion of templates and important constructs in C++ class design.

Chapter 2 deals with algorithm analysis. This chapter explains asymptotic analysis and its major weaknesses. Many examples are provided, including an in-depth explanation of logarithmic running time. Simple recursive programs are analyzed by intuitively converting them into iterative programs. More complicated divide-and-conquer programs are introduced, but some of the analysis (solving recurrence relations) is implicitly delayed until Chapter 7, where it is performed in detail.

Chapter 3 covers lists, stacks, and queues. This chapter includes a discussion of the STL `vector` and `list` classes, including material on iterators, and it provides implementations of a significant subset of the STL `vector` and `list` classes.

Chapter 4 covers trees, with an emphasis on search trees, including external search trees (B-trees). The UNIX file system and expression trees are used as examples. AVL trees and splay trees are introduced. More careful treatment of search tree implementation details is found in Chapter 12. Additional coverage of trees, such as file compression and game trees, is deferred until Chapter 10. Data structures for an external medium are considered as the final topic in several chapters. Included is a discussion of the STL `set` and `map` classes, including a significant example that illustrates the use of three separate maps to efficiently solve a problem.

Chapter 5 discusses hash tables, including the classic algorithms such as separate chaining and linear and quadratic probing, as well as several newer algorithms, namely cuckoo hashing and hopscotch hashing. Universal hashing is also discussed, and extendible hashing is covered at the end of the chapter.

Chapter 6 is about priority queues. Binary heaps are covered, and there is additional material on some of the theoretically interesting implementations of priority queues. The Fibonacci heap is discussed in Chapter 11, and the pairing heap is discussed in Chapter 12.

Chapter 7 covers sorting. It is very specific with respect to coding details and analysis. All the important general-purpose sorting algorithms are covered and compared. Four algorithms are analyzed in detail: insertion sort, Shellsort, heapsort, and quicksort. New to this edition is radix sort and lower bound proofs for selection-related problems. External sorting is covered at the end of the chapter.

Chapter 8 discusses the disjoint set algorithm with proof of the running time. This is a short and specific chapter that can be skipped if Kruskal's algorithm is not discussed.

Chapter 9 covers graph algorithms. Algorithms on graphs are interesting, not only because they frequently occur in practice but also because their running time is so heavily dependent on the proper use of data structures. Virtually all of the standard algorithms are presented along with appropriate data structures, pseudocode, and analysis of running time. To place these problems in a proper context, a short discussion on complexity theory (including *NP*-completeness and undecidability) is provided.

Chapter 10 covers algorithm design by examining common problem-solving techniques. This chapter is heavily fortified with examples. Pseudocode is used in these later chapters so that the student's appreciation of an example algorithm is not obscured by implementation details.

Chapter 11 deals with amortized analysis. Three data structures from Chapters 4 and 6 and the Fibonacci heap, introduced in this chapter, are analyzed.

Chapter 12 covers search tree algorithms, the suffix tree and array, the k -d tree, and the pairing heap. This chapter departs from the rest of the text by providing complete and careful implementations for the search trees and pairing heap. The material is structured so that the instructor can integrate sections into discussions from other chapters. For example, the top-down red-black tree in Chapter 12 can be discussed along with AVL trees (in Chapter 4).

Chapters 1 to 9 provide enough material for most one-semester data structures courses. If time permits, then Chapter 10 can be covered. A graduate course on algorithm analysis could cover chapters 7 to 11. The advanced data structures analyzed in Chapter 11 can easily be referred to in the earlier chapters. The discussion of *NP*-completeness in Chapter 9

is far too brief to be used in such a course. You might find it useful to use an additional work on *NP*-completeness to augment this text.

Exercises

Exercises, provided at the end of each chapter, match the order in which material is presented. The last exercises may address the chapter as a whole rather than a specific section. Difficult exercises are marked with an asterisk, and more challenging exercises have two asterisks.

References

References are placed at the end of each chapter. Generally the references either are historical, representing the original source of the material, or they represent extensions and improvements to the results given in the text. Some references represent solutions to exercises.

Supplements

The following supplements are available to all readers at <http://cssupport.pearsoncmg.com/>

- Source code for example programs
- Errata

In addition, the following material is available only to qualified instructors at Pearson Instructor Resource Center (www.pearsonhighered.com/irc). Visit the IRC or contact your Pearson Education sales representative for access.

- Solutions to selected exercises
- Figures from the book
- Errata

Acknowledgments

Many, many people have helped me in the preparation of books in this series. Some are listed in other versions of the book; thanks to all.

As usual, the writing process was made easier by the professionals at Pearson. I'd like to thank my editor, Tracy Johnson, and production editor, Marilyn Lloyd. My wonderful wife Jill deserves extra special thanks for everything she does.

Finally, I'd like to thank the numerous readers who have sent e-mail messages and pointed out errors or inconsistencies in earlier versions. My website www.cis.fiu.edu/~weiss will also contain updated source code (in C++ and Java), an errata list, and a link to submit bug reports.

M.A.W.

Miami, Florida

Preface xv

Chapter 1 Programming: A General Overview 1

- 1.1 What's This Book About? 1
- 1.2 Mathematics Review 2
 - 1.2.1 Exponents 3
 - 1.2.2 Logarithms 3
 - 1.2.3 Series 4
 - 1.2.4 Modular Arithmetic 5
 - 1.2.5 The P Word 6
- 1.3 A Brief Introduction to Recursion 8
- 1.4 C++ Classes 12
 - 1.4.1 Basic `class` Syntax 12
 - 1.4.2 Extra Constructor Syntax and Accessors 13
 - 1.4.3 Separation of Interface and Implementation 16
 - 1.4.4 `vector` and `string` 19
- 1.5 C++ Details 21
 - 1.5.1 Pointers 21
 - 1.5.2 Lvalues, Rvalues, and References 23
 - 1.5.3 Parameter Passing 25
 - 1.5.4 Return Passing 27
 - 1.5.5 `std::swap` and `std::move` 29
 - 1.5.6 The Big-Five: Destructor, Copy Constructor, Move Constructor, Copy Assignment `operator=`, Move Assignment `operator=` 30
 - 1.5.7 C-style Arrays and Strings 35
- 1.6 Templates 36
 - 1.6.1 Function Templates 37
 - 1.6.2 Class Templates 38
 - 1.6.3 `Object`, `Comparable`, and an Example 39
 - 1.6.4 Function Objects 41
 - 1.6.5 Separate Compilation of Class Templates 44
- 1.7 Using Matrices 44
 - 1.7.1 The Data Members, Constructor, and Basic Accessors 44
 - 1.7.2 `operator[]` 45

1.7.3 Big-Five	46
Summary	46
Exercises	46
References	48

Chapter 2 Algorithm Analysis

51

2.1 Mathematical Background	51
2.2 Model	54
2.3 What to Analyze	54
2.4 Running-Time Calculations	57
2.4.1 A Simple Example	58
2.4.2 General Rules	58
2.4.3 Solutions for the Maximum Subsequence Sum Problem	60
2.4.4 Logarithms in the Running Time	66
2.4.5 Limitations of Worst-Case Analysis	70
Summary	70
Exercises	71
References	76

Chapter 3 Lists, Stacks, and Queues

77

3.1 Abstract Data Types (ADTs)	77
3.2 The List ADT	78
3.2.1 Simple Array Implementation of Lists	78
3.2.2 Simple Linked Lists	79
3.3 <code>vector</code> and <code>list</code> in the STL	80
3.3.1 Iterators	82
3.3.2 Example: Using <code>erase</code> on a List	83
3.3.3 <code>const_iterators</code>	84
3.4 Implementation of <code>vector</code>	86
3.5 Implementation of <code>list</code>	91
3.6 The Stack ADT	103
3.6.1 Stack Model	103
3.6.2 Implementation of Stacks	104
3.6.3 Applications	104
3.7 The Queue ADT	112
3.7.1 Queue Model	113
3.7.2 Array Implementation of Queues	113
3.7.3 Applications of Queues	115
Summary	116
Exercises	116

Chapter 4 Trees

121

- 4.1 Preliminaries 121
 - 4.1.1 Implementation of Trees 122
 - 4.1.2 Tree Traversals with an Application 123
- 4.2 Binary Trees 126
 - 4.2.1 Implementation 128
 - 4.2.2 An Example: Expression Trees 128
- 4.3 The Search Tree ADT—Binary Search Trees 132
 - 4.3.1 `contains` 134
 - 4.3.2 `findMin` and `findMax` 135
 - 4.3.3 `insert` 136
 - 4.3.4 `remove` 139
 - 4.3.5 Destructor and Copy Constructor 141
 - 4.3.6 Average-Case Analysis 141
- 4.4 AVL Trees 144
 - 4.4.1 Single Rotation 147
 - 4.4.2 Double Rotation 149
- 4.5 Splay Trees 158
 - 4.5.1 A Simple Idea (That Does Not Work) 158
 - 4.5.2 Splaying 160
- 4.6 Tree Traversals (Revisited) 166
- 4.7 B-Trees 168
- 4.8 Sets and Maps in the Standard Library 173
 - 4.8.1 Sets 173
 - 4.8.2 Maps 174
 - 4.8.3 Implementation of `set` and `map` 175
 - 4.8.4 An Example That Uses Several Maps 176
- Summary 181
- Exercises 182
- References 189

Chapter 5 Hashing

193

- 5.1 General Idea 193
- 5.2 Hash Function 194
- 5.3 Separate Chaining 196
- 5.4 Hash Tables without Linked Lists 201
 - 5.4.1 Linear Probing 201
 - 5.4.2 Quadratic Probing 202
 - 5.4.3 Double Hashing 207
- 5.5 Rehashing 208
- 5.6 Hash Tables in the Standard Library 210

5.7	Hash Tables with Worst-Case $O(1)$ Access	212
5.7.1	Perfect Hashing	213
5.7.2	Cuckoo Hashing	215
5.7.3	Hopscotch Hashing	227
5.8	Universal Hashing	230
5.9	Extendible Hashing	233
	Summary	236
	Exercises	237
	References	241

Chapter 6 Priority Queues (Heaps)

245

6.1	Model	245
6.2	Simple Implementations	246
6.3	Binary Heap	247
6.3.1	Structure Property	247
6.3.2	Heap-Order Property	248
6.3.3	Basic Heap Operations	249
6.3.4	Other Heap Operations	252
6.4	Applications of Priority Queues	257
6.4.1	The Selection Problem	258
6.4.2	Event Simulation	259
6.5	d -Heaps	260
6.6	Leftist Heaps	261
6.6.1	Leftist Heap Property	261
6.6.2	Leftist Heap Operations	262
6.7	Skew Heaps	269
6.8	Binomial Queues	271
6.8.1	Binomial Queue Structure	271
6.8.2	Binomial Queue Operations	271
6.8.3	Implementation of Binomial Queues	276
6.9	Priority Queues in the Standard Library	282
	Summary	283
	Exercises	283
	References	288

Chapter 7 Sorting

291

7.1	Preliminaries	291
7.2	Insertion Sort	292
7.2.1	The Algorithm	292
7.2.2	STL Implementation of Insertion Sort	293
7.2.3	Analysis of Insertion Sort	294
7.3	A Lower Bound for Simple Sorting Algorithms	295

7.4	Shellsort	296
7.4.1	Worst-Case Analysis of Shellsort	297
7.5	Heapsort	300
7.5.1	Analysis of Heapsort	301
7.6	Mergesort	304
7.6.1	Analysis of Mergesort	306
7.7	Quicksort	309
7.7.1	Picking the Pivot	311
7.7.2	Partitioning Strategy	313
7.7.3	Small Arrays	315
7.7.4	Actual Quicksort Routines	315
7.7.5	Analysis of Quicksort	318
7.7.6	A Linear-Expected-Time Algorithm for Selection	321
7.8	A General Lower Bound for Sorting	323
7.8.1	Decision Trees	323
7.9	Decision-Tree Lower Bounds for Selection Problems	325
7.10	Adversary Lower Bounds	328
7.11	Linear-Time Sorts: Bucket Sort and Radix Sort	331
7.12	External Sorting	336
7.12.1	Why We Need New Algorithms	336
7.12.2	Model for External Sorting	336
7.12.3	The Simple Algorithm	337
7.12.4	Multiway Merge	338
7.12.5	Polyphase Merge	339
7.12.6	Replacement Selection	340
	Summary	341
	Exercises	341
	References	347

Chapter 8 The Disjoint Sets Class

351

8.1	Equivalence Relations	351
8.2	The Dynamic Equivalence Problem	352
8.3	Basic Data Structure	353
8.4	Smart Union Algorithms	357
8.5	Path Compression	360
8.6	Worst Case for Union-by-Rank and Path Compression	361
8.6.1	Slowly Growing Functions	362
8.6.2	An Analysis by Recursive Decomposition	362
8.6.3	An $O(M \log^* N)$ Bound	369
8.6.4	An $O(M \alpha(M, N))$ Bound	370
8.7	An Application	372

Summary	374
Exercises	375
References	376

Chapter 9 Graph Algorithms

379

9.1	Definitions	379
9.1.1	Representation of Graphs	380
9.2	Topological Sort	382
9.3	Shortest-Path Algorithms	386
9.3.1	Unweighted Shortest Paths	387
9.3.2	Dijkstra's Algorithm	391
9.3.3	Graphs with Negative Edge Costs	400
9.3.4	Acyclic Graphs	400
9.3.5	All-Pairs Shortest Path	404
9.3.6	Shortest Path Example	404
9.4	Network Flow Problems	406
9.4.1	A Simple Maximum-Flow Algorithm	408
9.5	Minimum Spanning Tree	413
9.5.1	Prim's Algorithm	414
9.5.2	Kruskal's Algorithm	417
9.6	Applications of Depth-First Search	419
9.6.1	Undirected Graphs	420
9.6.2	Biconnectivity	421
9.6.3	Euler Circuits	425
9.6.4	Directed Graphs	429
9.6.5	Finding Strong Components	431
9.7	Introduction to NP-Completeness	432
9.7.1	Easy vs. Hard	433
9.7.2	The Class NP	434
9.7.3	NP-Complete Problems	434
	Summary	437
	Exercises	437
	References	445

Chapter 10 Algorithm Design Techniques

449

10.1	Greedy Algorithms	449
10.1.1	A Simple Scheduling Problem	450
10.1.2	Huffman Codes	453
10.1.3	Approximate Bin Packing	459
10.2	Divide and Conquer	467
10.2.1	Running Time of Divide-and-Conquer Algorithms	468
10.2.2	Closest-Points Problem	470