

Data Structures & Algorithms



in **Python**

MICHAEL T. GOODRICH • ROBERTO TAMASSIA • MICHAEL H. GOLDWASSER

Data Structures and Algorithms in Python

Michael T. Goodrich

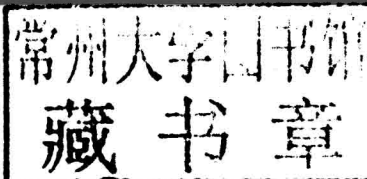
Department of Computer Science
University of California, Irvine

Roberto Tamassia

Department of Computer Science
Brown University

Michael H. Goldwasser

Department of Mathematics and Computer Science
Saint Louis University



WILEY

VP & PUBLISHER
EXECUTIVE EDITOR
EDITORIAL PROGRAM ASSISTANT
MARKETING MANAGER
DESIGNER
SENIOR PRODUCTION MANAGER
ASSOCIATE PRODUCTION MANAGER

Don Fowley
Beth Lang Golub
Katherine Willis
Christopher Ruel
Kenji Ngieng
Janis Soo
Joyce Poh

This book was set in L^AT_EX by the authors. Printed and bound by Courier Kendallville.
The cover was printed by Courier Kendallville.

This book is printed on acid free paper.

Founded in 1807, John Wiley & Sons, Inc. has been a valued source of knowledge and understanding for more than 200 years, helping people around the world meet their needs and fulfill their aspirations. Our company is built on a foundation of principles that include responsibility to the communities we serve and where we live and work. In 2008, we launched a Corporate Citizenship Initiative, a global effort to address the environmental, social, economic, and ethical challenges we face in our business. Among the issues we are addressing are carbon impact, paper specifications and procurement, ethical conduct within our business and among our vendors, and community and charitable support. For more information, please visit our website: www.wiley.com/go/citizenship.

Copyright © 2013 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, website www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201)748-6011, fax (201)748-6008, website <http://www.wiley.com/go/permissions>.

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instructions and a free of charge return mailing label are available at www.wiley.com/go/returnlabel. If you have chosen to adopt this textbook for use in your course, please accept this book as your complimentary desk copy. Outside of the United States, please contact your local sales representative.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To Karen, Paul, Anna, and Jack
– *Michael T. Goodrich*

To Isabel
– *Roberto Tamassia*

To Susan, Calista, and Maya
– *Michael H. Goldwasser*

Preface

The design and analysis of efficient data structures has long been recognized as a vital subject in computing and is part of the core curriculum of computer science and computer engineering undergraduate degrees. *Data Structures and Algorithms in Python* provides an introduction to data structures and algorithms, including their design, analysis, and implementation. This book is designed for use in a beginning-level data structures course, or in an intermediate-level introduction to algorithms course. We discuss its use for such courses in more detail later in this preface.

To promote the development of robust and reusable software, we have tried to take a consistent object-oriented viewpoint throughout this text. One of the main ideas of the object-oriented approach is that data should be presented as being encapsulated with the methods that access and modify them. That is, rather than simply viewing data as a collection of bytes and addresses, we think of data objects as instances of an **abstract data type (ADT)**, which includes a repertoire of methods for performing operations on data objects of this type. We then emphasize that there may be several different implementation strategies for a particular ADT, and explore the relative pros and cons of these choices. We provide complete Python implementations for almost all data structures and algorithms discussed, and we introduce important object-oriented **design patterns** as means to organize those implementations into reusable components.

Desired outcomes for readers of our book include that:

- They have knowledge of the most common abstractions for data collections (e.g., stacks, queues, lists, trees, maps).
- They understand algorithmic strategies for producing efficient realizations of common data structures.
- They can analyze algorithmic performance, both theoretically and experimentally, and recognize common trade-offs between competing strategies.
- They can wisely use existing data structures and algorithms found in modern programming language libraries.
- They have experience working with concrete implementations for most foundational data structures and algorithms.
- They can apply data structures and algorithms to solve complex problems.

In support of the last goal, we present many example applications of data structures throughout the book, including the processing of file systems, matching of tags in structured formats such as HTML, simple cryptography, text frequency analysis, automated geometric layout, Huffman coding, DNA sequence alignment, and search engine indexing.

Book Features

This book is based upon the book *Data Structures and Algorithms in Java* by Goodrich and Tamassia, and the related *Data Structures and Algorithms in C++* by Goodrich, Tamassia, and Mount. However, this book is not simply a translation of those other books to Python. In adapting the material for this book, we have significantly redesigned the organization and content of the book as follows:

- The code base has been entirely redesigned to take advantage of the features of Python, such as use of generators for iterating elements of a collection.
- Many algorithms that were presented as pseudo-code in the Java and C++ versions are directly presented as complete Python code.
- In general, ADTs are defined to have consistent interface with Python's built-in data types and those in Python's collections module.
- Chapter 5 provides an in-depth exploration of the dynamic array-based underpinnings of Python's built-in list, tuple, and str classes. New Appendix A serves as an additional reference regarding the functionality of the str class.
- Over 450 illustrations have been created or revised.
- New and revised exercises bring the overall total number to 750.

Online Resources

This book is accompanied by an extensive set of online resources, which can be found at the following Web site:

www.wiley.com/college/goodrich

Students are encouraged to use this site along with the book, to help with exercises and increase understanding of the subject. Instructors are likewise welcome to use the site to help plan, organize, and present their course materials. Included on this Web site is a collection of educational aids that augment the topics of this book, for both students and instructors. Because of their added value, some of these online resources are password protected.

For all readers, and especially for students, we include the following resources:

- All the Python source code presented in this book.
- PDF handouts of Powerpoint slides (four-per-page) provided to instructors.
- A database of hints to *all* exercises, indexed by problem number.

For instructors using this book, we include the following additional teaching aids:

- Solutions to hundreds of the book's exercises.
- Color versions of all figures and illustrations from the book.
- Slides in Powerpoint and PDF (one-per-page) format.

The slides are fully editable, so as to allow an instructor using this book full freedom in customizing his or her presentations. All the online resources are provided at no extra charge to any instructor adopting this book for his or her course.

Contents and Organization

The chapters for this book are organized to provide a pedagogical path that starts with the basics of Python programming and object-oriented design. We then add foundational techniques like algorithm analysis and recursion. In the main portion of the book, we present fundamental data structures and algorithms, concluding with a discussion of memory management (that is, the architectural underpinnings of data structures). Specifically, the chapters for this book are organized as follows:

1. **Python Primer**
2. **Object-Oriented Programming**
3. **Algorithm Analysis**
4. **Recursion**
5. **Array-Based Sequences**
6. **Stacks, Queues, and Deques**
7. **Linked Lists**
8. **Trees**
9. **Priority Queues**
10. **Maps, Hash Tables, and Skip Lists**
11. **Search Trees**
12. **Sorting and Selection**
13. **Text Processing**
14. **Graph Algorithms**
15. **Memory Management and B-Trees**
 - A. **Character Strings in Python**
 - B. **Useful Mathematical Facts**

A more detailed table of contents follows this preface, beginning on page xi.

Prerequisites

We assume that the reader is at least vaguely familiar with a high-level programming language, such as C, C++, Python, or Java, and that he or she understands the main constructs from such a high-level language, including:

- Variables and expressions.
- Decision structures (such as if-statements and switch-statements).
- Iteration structures (for loops and while loops).
- Functions (whether stand-alone or object-oriented methods).

For readers who are familiar with these concepts, but not with how they are expressed in Python, we provide a primer on the Python language in Chapter 1. Still, this book is primarily a data structures book, not a Python book; hence, it does not give a comprehensive treatment of Python.

We delay treatment of object-oriented programming in Python until Chapter 2. This chapter is useful for those new to Python, and for those who may be familiar with Python, yet not with object-oriented programming.

In terms of mathematical background, we assume the reader is somewhat familiar with topics from high-school mathematics. Even so, in Chapter 3, we discuss the seven most-important functions for algorithm analysis. In fact, sections that use something other than one of these seven functions are considered optional, and are indicated with a star (★). We give a summary of other useful mathematical facts, including elementary probability, in Appendix B.

Relation to Computer Science Curriculum

To assist instructors in designing a course in the context of the IEEE/ACM 2013 Computing Curriculum, the following table describes curricular knowledge units that are covered within this book.

Knowledge Unit	Relevant Material
AL/Basic Analysis	Chapter 3 and Sections 4.2 & 12.2.4
AL/Algorithmic Strategies	Sections 12.2.1, 13.2.1, 13.3, & 13.4.2
AL/Fundamental Data Structures and Algorithms	Sections 4.1.3, 5.5.2, 9.4.1, 9.3, 10.2, 11.1, 13.2, Chapter 12 & much of Chapter 14
AL/Advanced Data Structures	Sections 5.3, 10.4, 11.2 through 11.6, 12.3.1, 13.5, 14.5.1, & 15.3
AR/Memory System Organization and Architecture	Chapter 15
DS/Sets, Relations and Functions	Sections 10.5.1, 10.5.2, & 9.4
DS/Proof Techniques	Sections 3.4, 4.2, 5.3.2, 9.3.6, & 12.4.1
DS/Basics of Counting	Sections 2.4.2, 6.2.2, 12.2.4, 8.2.2 & Appendix B
DS/Graphs and Trees	Much of Chapters 8 and 14
DS/Discrete Probability	Sections 1.11.1, 10.2, 10.4.2, & 12.3.1
PL/Object-Oriented Programming	Much of the book, yet especially Chapter 2 and Sections 7.4, 9.5.1, 10.1.3, & 11.2.1
PL/Functional Programming	Section 1.10
SDF/Algorithms and Design	Sections 2.1, 3.3, & 12.2.1
SDF/Fundamental Programming Concepts	Chapters 1 & 4
SDF/Fundamental Data Structures	Chapters 6 & 7, Appendix A, and Sections 1.2.1, 5.2, 5.4, 9.1, & 10.1
SDF/Developmental Methods	Sections 1.7 & 2.2
SE/Software Design	Sections 2.1 & 2.1.3

Mapping *IEEE/ACM 2013 Computing Curriculum* knowledge units to coverage in this book.

About the Authors

Michael Goodrich received his Ph.D. in Computer Science from Purdue University in 1987. He is currently a Chancellor's Professor in the Department of Computer Science at University of California, Irvine. Previously, he was a professor at Johns Hopkins University. He is a Fulbright Scholar and a Fellow of the American Association for the Advancement of Science (AAAS), Association for Computing Machinery (ACM), and Institute of Electrical and Electronics Engineers (IEEE). He is a recipient of the IEEE Computer Society Technical Achievement Award, the ACM Recognition of Service Award, and the Pond Award for Excellence in Undergraduate Teaching.

Roberto Tamassia received his Ph.D. in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 1988. He is the Plastech Professor of Computer Science and the Chair of the Department of Computer Science at Brown University. He is also the Director of Brown's Center for Geometric Computing. His research interests include information security, cryptography, analysis, design, and implementation of algorithms, graph drawing and computational geometry. He is a Fellow of the American Association for the Advancement of Science (AAAS), Association for Computing Machinery (ACM) and Institute for Electrical and Electronic Engineers (IEEE). He is also a recipient of the Technical Achievement Award from the IEEE Computer Society.

Michael Goldwasser received his Ph.D. in Computer Science from Stanford University in 1997. He is currently a Professor in the Department of Mathematics and Computer Science at Saint Louis University and the Director of their Computer Science program. Previously, he was a faculty member in the Department of Computer Science at Loyola University Chicago. His research interests focus on the design and implementation of algorithms, having published work involving approximation algorithms, online computation, computational biology, and computational geometry. He is also active in the computer science education community.

Additional Books by These Authors

- M.T. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java*, Wiley.
- M.T. Goodrich, R. Tamassia, and D.M. Mount, *Data Structures and Algorithms in C++*, Wiley.
- M.T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*, Wiley.
- M.T. Goodrich and R. Tamassia, *Introduction to Computer Security*, Addison-Wesley.
- M.H. Goldwasser and D. Letscher, *Object-Oriented Programming in Python*, Prentice Hall.

Acknowledgments

We have depended greatly upon the contributions of many individuals as part of the development of this book. We begin by acknowledging the wonderful team at Wiley. We are grateful to our editor, Beth Golub, for her enthusiastic support of this project, from beginning to end. The efforts of Elizabeth Mills and Katherine Willis were critical in keeping the project moving, from its early stages as an initial proposal, through the extensive peer review process. We greatly appreciate the attention to detail demonstrated by Julie Kennedy, the copyeditor for this book. Finally, many thanks are due to Joyce Poh for managing the final months of the production process.

We are truly indebted to the outside reviewers and readers for their copious comments, emails, and constructive criticism, which were extremely useful in writing this edition. We therefore thank the following reviewers for their comments and suggestions: Claude Anderson (Rose Hulman Institute of Technology), Alistair Campbell (Hamilton College), Barry Cohen (New Jersey Institute of Technology), Robert Franks (Central College), Andrew Harrington (Loyola University Chicago), Dave Musicant (Carleton College), and Victor Norman (Calvin College). We wish to particularly acknowledge Claude for going above and beyond the call of duty, providing us with an enumeration of 400 detailed corrections or suggestions.

We thank David Mount, of University of Maryland, for graciously sharing the wisdom gained from his experience with the C++ version of this text. We are grateful to Erin Chambers and David Letscher, of Saint Louis University, for their intangible contributions during many hallway conversations about the teaching of data structures, and to David for comments on early versions of the Python code base for this book. We thank David Zampino, a student at Loyola University Chicago, for his feedback while using a draft of this book during an independent study course, and to Andrew Harrington for supervising David's studies.

We also wish to reiterate our thanks to the many research collaborators and teaching assistants whose feedback shaped the previous Java and C++ versions of this material. The benefits of those contributions carry forward to this book.

Finally, we would like to warmly thank Susan Goldwasser, Isabel Cruz, Karen Goodrich, Giuseppe Di Battista, Franco Preparata, Ioannis Tollis, and our parents for providing advice, encouragement, and support at various stages of the preparation of this book, and Calista and Maya Goldwasser for offering their advice regarding the artistic merits of many illustrations. More importantly, we thank all of these people for reminding us that there are things in life beyond writing books.

Michael T. Goodrich
Roberto Tamassia
Michael H. Goldwasser

Contents

Preface	v
1 Python Primer	1
1.1 Python Overview	2
1.1.1 The Python Interpreter	2
1.1.2 Preview of a Python Program	3
1.2 Objects in Python	4
1.2.1 Identifiers, Objects, and the Assignment Statement	4
1.2.2 Creating and Using Objects	6
1.2.3 Python's Built-In Classes	7
1.3 Expressions, Operators, and Precedence	12
1.3.1 Compound Expressions and Operator Precedence	17
1.4 Control Flow	18
1.4.1 Conditionals	18
1.4.2 Loops	20
1.5 Functions	23
1.5.1 Information Passing	24
1.5.2 Python's Built-In Functions	28
1.6 Simple Input and Output	30
1.6.1 Console Input and Output	30
1.6.2 Files	31
1.7 Exception Handling	33
1.7.1 Raising an Exception	34
1.7.2 Catching an Exception	36
1.8 Iterators and Generators	39
1.9 Additional Python Conveniences	42
1.9.1 Conditional Expressions	42
1.9.2 Comprehension Syntax	43
1.9.3 Packing and Unpacking of Sequences	44
1.10 Scopes and Namespaces	46
1.11 Modules and the Import Statement	48
1.11.1 Existing Modules	49
1.12 Exercises	51

2	Object-Oriented Programming	56
2.1	Goals, Principles, and Patterns	57
2.1.1	Object-Oriented Design Goals	57
2.1.2	Object-Oriented Design Principles	58
2.1.3	Design Patterns	61
2.2	Software Development	62
2.2.1	Design	62
2.2.2	Pseudo-Code	64
2.2.3	Coding Style and Documentation	64
2.2.4	Testing and Debugging	67
2.3	Class Definitions	69
2.3.1	Example: CreditCard Class	69
2.3.2	Operator Overloading and Python's Special Methods	74
2.3.3	Example: Multidimensional Vector Class	77
2.3.4	Iterators	79
2.3.5	Example: Range Class	80
2.4	Inheritance	82
2.4.1	Extending the CreditCard Class	83
2.4.2	Hierarchy of Numeric Progressions	87
2.4.3	Abstract Base Classes	93
2.5	Namespaces and Object-Orientation	96
2.5.1	Instance and Class Namespaces	96
2.5.2	Name Resolution and Dynamic Dispatch	100
2.6	Shallow and Deep Copying	101
2.7	Exercises	103
3	Algorithm Analysis	109
3.1	Experimental Studies	111
3.1.1	Moving Beyond Experimental Analysis	113
3.2	The Seven Functions Used in This Book	115
3.2.1	Comparing Growth Rates	122
3.3	Asymptotic Analysis	123
3.3.1	The "Big-Oh" Notation	123
3.3.2	Comparative Analysis	128
3.3.3	Examples of Algorithm Analysis	130
3.4	Simple Justification Techniques	137
3.4.1	By Example	137
3.4.2	The "Contra" Attack	137
3.4.3	Induction and Loop Invariants	138
3.5	Exercises	141

4	Recursion	148
4.1	Illustrative Examples	150
4.1.1	The Factorial Function	150
4.1.2	Drawing an English Ruler	152
4.1.3	Binary Search	155
4.1.4	File Systems	157
4.2	Analyzing Recursive Algorithms	161
4.3	Recursion Run Amok	165
4.3.1	Maximum Recursive Depth in Python	168
4.4	Further Examples of Recursion	169
4.4.1	Linear Recursion	169
4.4.2	Binary Recursion	174
4.4.3	Multiple Recursion	175
4.5	Designing Recursive Algorithms	177
4.6	Eliminating Tail Recursion	178
4.7	Exercises	180
5	Array-Based Sequences	183
5.1	Python's Sequence Types	184
5.2	Low-Level Arrays	185
5.2.1	Referential Arrays	187
5.2.2	Compact Arrays in Python	190
5.3	Dynamic Arrays and Amortization	192
5.3.1	Implementing a Dynamic Array	195
5.3.2	Amortized Analysis of Dynamic Arrays	197
5.3.3	Python's List Class	201
5.4	Efficiency of Python's Sequence Types	202
5.4.1	Python's List and Tuple Classes	202
5.4.2	Python's String Class	208
5.5	Using Array-Based Sequences	210
5.5.1	Storing High Scores for a Game	210
5.5.2	Sorting a Sequence	214
5.5.3	Simple Cryptography	216
5.6	Multidimensional Data Sets	219
5.7	Exercises	224
6	Stacks, Queues, and Deques	228
6.1	Stacks	229
6.1.1	The Stack Abstract Data Type	230
6.1.2	Simple Array-Based Stack Implementation	231
6.1.3	Reversing Data Using a Stack	235
6.1.4	Matching Parentheses and HTML Tags	236

6.2	Queues	239
6.2.1	The Queue Abstract Data Type	240
6.2.2	Array-Based Queue Implementation	241
6.3	Double-Ended Queues	247
6.3.1	The Deque Abstract Data Type	247
6.3.2	Implementing a Deque with a Circular Array	248
6.3.3	Dequeues in the Python Collections Module	249
6.4	Exercises	250
7	Linked Lists	255
7.1	Singly Linked Lists	256
7.1.1	Implementing a Stack with a Singly Linked List	261
7.1.2	Implementing a Queue with a Singly Linked List	264
7.2	Circularly Linked Lists	266
7.2.1	Round-Robin Schedulers	267
7.2.2	Implementing a Queue with a Circularly Linked List	268
7.3	Doubly Linked Lists	270
7.3.1	Basic Implementation of a Doubly Linked List	273
7.3.2	Implementing a Deque with a Doubly Linked List	275
7.4	The Positional List ADT	277
7.4.1	The Positional List Abstract Data Type	279
7.4.2	Doubly Linked List Implementation	281
7.5	Sorting a Positional List	285
7.6	Case Study: Maintaining Access Frequencies	286
7.6.1	Using a Sorted List	286
7.6.2	Using a List with the Move-to-Front Heuristic	289
7.7	Link-Based vs. Array-Based Sequences	292
7.8	Exercises	294
8	Trees	299
8.1	General Trees	300
8.1.1	Tree Definitions and Properties	301
8.1.2	The Tree Abstract Data Type	305
8.1.3	Computing Depth and Height	308
8.2	Binary Trees	311
8.2.1	The Binary Tree Abstract Data Type	313
8.2.2	Properties of Binary Trees	315
8.3	Implementing Trees	317
8.3.1	Linked Structure for Binary Trees	317
8.3.2	Array-Based Representation of a Binary Tree	325
8.3.3	Linked Structure for General Trees	327
8.4	Tree Traversal Algorithms	328

8.4.1	Preorder and Postorder Traversals of General Trees . . .	328
8.4.2	Breadth-First Tree Traversal	330
8.4.3	Inorder Traversal of a Binary Tree	331
8.4.4	Implementing Tree Traversals in Python	333
8.4.5	Applications of Tree Traversals	337
8.4.6	Euler Tours and the Template Method Pattern ★	341
8.5	Case Study: An Expression Tree	348
8.6	Exercises	352
9	Priority Queues	362
9.1	The Priority Queue Abstract Data Type	363
9.1.1	Priorities	363
9.1.2	The Priority Queue ADT	364
9.2	Implementing a Priority Queue	365
9.2.1	The Composition Design Pattern	365
9.2.2	Implementation with an Unsorted List	366
9.2.3	Implementation with a Sorted List	368
9.3	Heaps	370
9.3.1	The Heap Data Structure	370
9.3.2	Implementing a Priority Queue with a Heap	372
9.3.3	Array-Based Representation of a Complete Binary Tree .	376
9.3.4	Python Heap Implementation	376
9.3.5	Analysis of a Heap-Based Priority Queue	379
9.3.6	Bottom-Up Heap Construction ★	380
9.3.7	Python's heapq Module	384
9.4	Sorting with a Priority Queue	385
9.4.1	Selection-Sort and Insertion-Sort	386
9.4.2	Heap-Sort	388
9.5	Adaptable Priority Queues	390
9.5.1	Locators	390
9.5.2	Implementing an Adaptable Priority Queue	391
9.6	Exercises	395
10	Maps, Hash Tables, and Skip Lists	401
10.1	Maps and Dictionaries	402
10.1.1	The Map ADT	403
10.1.2	Application: Counting Word Frequencies	405
10.1.3	Python's MutableMapping Abstract Base Class	406
10.1.4	Our MapBase Class	407
10.1.5	Simple Unsorted Map Implementation	408
10.2	Hash Tables	410
10.2.1	Hash Functions	411

10.2.2	Collision-Handling Schemes	417
10.2.3	Load Factors, Rehashing, and Efficiency	420
10.2.4	Python Hash Table Implementation	422
10.3	Sorted Maps	427
10.3.1	Sorted Search Tables	428
10.3.2	Two Applications of Sorted Maps	434
10.4	Skip Lists	437
10.4.1	Search and Update Operations in a Skip List	439
10.4.2	Probabilistic Analysis of Skip Lists ★	443
10.5	Sets, Multisets, and Multimaps	446
10.5.1	The Set ADT	446
10.5.2	Python's MutableSet Abstract Base Class	448
10.5.3	Implementing Sets, Multisets, and Multimaps	450
10.6	Exercises	452
11	Search Trees	459
11.1	Binary Search Trees	460
11.1.1	Navigating a Binary Search Tree	461
11.1.2	Searches	463
11.1.3	Insertions and Deletions	465
11.1.4	Python Implementation	468
11.1.5	Performance of a Binary Search Tree	473
11.2	Balanced Search Trees	475
11.2.1	Python Framework for Balancing Search Trees	478
11.3	AVL Trees	481
11.3.1	Update Operations	483
11.3.2	Python Implementation	488
11.4	Splay Trees	490
11.4.1	Splaying	490
11.4.2	When to Splay	494
11.4.3	Python Implementation	496
11.4.4	Amortized Analysis of Splaying ★	497
11.5	(2,4) Trees	502
11.5.1	Multiway Search Trees	502
11.5.2	(2,4)-Tree Operations	505
11.6	Red-Black Trees	512
11.6.1	Red-Black Tree Operations	514
11.6.2	Python Implementation	525
11.7	Exercises	528