国外计算机科学教材系列

数据结构与算法分析 （Java版）

A Practical Introduction to Data Structures and Algorithm Analysis

Java Edition

英文原版

[美] Clifford A. Shaffer 著

Prentice Hall

電子工業出版社
Publishing House of Electronics Industry
www.phei.com.cn

国外计算机科学教材系列

数据结构与算法分析

(Java 版)

(英文原版)

A Practical Introduction to Data Structures
and Algorithm Analysis
Java Edition

［美］ Clifford A. Shaffer　著

電子工業出版社·

Publishing House of Electronics Industry

北京·BEIJING

内容简介

本书采用了当前十分流行且适合于Internet环境的面向对象的程序设计语言Java作为算法描述语言。本书利用Java的接口来定义抽象数据类型，并把数据结构原理和算法分析技术有机地结合在一起，系统地介绍了各种类型的数据结构和排序、检索的各种方法。书中还引入了一些比较高级的数据结构与先进的算法分析技术，并介绍了可计算性理论的一般知识。

本书概念清晰，逻辑性强，内容新颖，可作为大专院校计算机软件专业与计算机应用专业的教材和参考书，也可供计算机工程技术人员参考。

版权贸易合同登记号：图字：01-2002-1400

图书在版编目（CIP）数据

数据结构与算法分析（Java版）（英文原版）/（美）谢弗（Shaffer, C. A.）著；
-北京：电子工业出版社，2002.5
（国外计算机科学教材系列）
书名原文：A Practical Introduction to Data Structures and Algorithm Analysis Java Edition
ISBN 7-5053-7581-4

Ⅰ. 数… Ⅱ. 谢… Ⅲ. ①Java语言－数据结构－教材－英文 ②Java语言－算法分析－教材－英文
Ⅳ. ①TP312 ②TP311.12

中国版本图书馆CIP数据核字（2002）第027766号

责任编辑：赵宏英
印 刷 者：北京兴华印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092 1/16 印张：24 字数：599千字
版　　次：2002年5月第1版 2002年5月第1次印刷
定　　价：39.00元

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默(Douglas E. Comer)、威廉·斯托林斯(William Stallings)、哈维·戴特尔(Harvey M. Deitel)、尤利斯·布莱克(Uyless Black)等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

目录概览

前言 …… 1
Preface

第一部分　预备知识 …… 7
Preliminaries

第 1 章　数据结构和算法 …… 8
Data Structures and Algorithms

第 2 章　数学预备知识 …… 19
Mathematical Preliminaries

第 3 章　算法分析 …… 35
Algorithm Analysis

第二部分　基本数据结构 …… 61
Fundamental Data Structures

第 4 章　线性表、栈和队列 …… 62
Lists, Stacks, and queues

第 5 章　二叉树 …… 97
Binary Trees

第 6 章　树 …… 132
General Trees

第 7 章　图 …… 150
Graphs

第三部分　排序和检索 …… 179
Sorting and Searching

第 8 章　内排序 …… 180
Internal Sorting

第 9 章　文件管理和外排序 …… 213
File Processing and External Sorting

第 10 章　检索 …… 238
Searching

第 11 章　索引技术 …… 262
Indexing

第四部分　应用与高级话题 …… 287
Applications and Advanced Topics

第 12 章　线性表和数组高级技术 …… 288
Lists and Arrays Revisited

第 13 章　高级树形结构 …… 312
Advanced Tree Structures

第 14 章　分析技术 …… 330
Analysis Techniques

第 15 章　计算的限制 …… 344
Limits to Computation

第五部分　附录 …… 361
Appendix

附录 A　C 和 Pascal 程序员的 Java 导引 …… 362
Java Tutorial for C and Pascal Programmers

参考文献 …… 368
Bibliography

Contents

Preface I

Part Ⅰ Preliminaries 7

Chapter 1 Data Structures and Algorithms 8

1.1 A Philosophy of Data Structures 8

1.1.1 The Need for Data Structures 8

1.1.2 Costs and Benefits 9

1.1.3 Goals of This Book 10

1.2 Abstract Data Types and Data Structures 10

1.3 Problems, Algorithms, and Programs 13

1.4 Algorithm Efficiency 15

1.5 Further Reading 16

1.6 Exercises 16

Chapter 2 Mathematical Preliminaries 19

2.1 Sets 19

2.2 Miscellaneous Notation 20

2.3 Logarithms 22

2.4 Recursion 23

2.5 Summations and Recurrences 25

2.6 Mathematical Proof Techniques 26

2.6.1 Proof by Contradiction 27

2.6.2 Proof by Mathematical Induction 27

2.7 Estimating 30

2.8 Further Reading 31

2.9 Exercises 32

chapter 3 Algorithm Analysis 35

3.1 Introduction 35

3.2 Best, Worst, and Average Cases 39

3.3 A Faster Computer, or a Faster Algorithm? 40

3.4 Asymptotic Analysis 42

3.4.1 Upper Bounds 42
3.4.2 Lower Bounds 44
3.4.3 Q Notation 45
3.4.4 Simplifying Rules 45
3.5 Calculating the Running Time of a Program 46
3.6 Analyzing Problems 50
3.7 Multiple Parameters 51
3.8 Space Bounds 51
3.9 Some Practical Considerations 54
3.10 Further Reading 55
3.11 Exercises 56
3.12 Projects 59

Part II Fundamental Data Structures 61

Chapter 4 Lists, Stacks, and Queues 62
4.1 Lists 62
4.1.1 Array-Based List Implementation 65
4.1.2 Linked List 68
4.1.3 Comparison of List Implementations 76
4.1.4 Element Implementations 77
4.1.5 Doubly Linked List 78
4.1.6 Circular Linked Lists 82
4.2 Stacks 82
4.2.1 Array-Based Stacks 83
4.2.2 Linked Stacks 85
4.2.3 Comparison of Array-Based and Linked Stacks 85
4.2.4 Implementing Recursion 86
4.3 Queues 89
4.3.1 Array-Based Queues 89
4.3.2 Linked Queues 92
4.3.3 Comparison of Array-Based and Linked Queues 93
4.4 Exercises 93
4.5 Projects 95

Chapter 5 Binary Trees 97
5.1 Definitions and Properties 97
5.1.1 The Full Binary Tree Theorem 99
5.1.2 A Binary Tree Node ADT 100
5.2 Binary Tree Traversals 101
5.3 Binary Tree Implementations 102

5.3.1 Pointer-Based Node Implementations 102
5.3.2 Space Requirements 106
5.3.3 Array Implementation for Complete Binary Trees 107
5.4 Huffman Coding Trees 107
5.4.1 Building Huffman Coding Trees 109
5.4.2 Assigning and Using Huffman Codes 113
5.5 Binary Search Trees 116
5.6 Heaps and Priority Queues 122
5.7 Further Reading 128
5.8 Exercises 128
5.9 Projects 130

Chapter 6 General Trees 132
6.1 General Tree Definitions and Terminology 132
6.1.1 An ADT for General Tree Nodes 132
6.1.2 General Tree Traversals 133
6.2 The Parent Pointer Implementation 134
6.3 General Tree Implementations 139
6.3.1 List of Children 139
6.3.2 The Left Child/Right Sibling Implementation 140
6.3.3 Dynamic Node Implementations 141
6.3.4 Dynamic "Left Child/Right Sibling" Implementation 142
6.4 K-ary Trees 143
6.5 Sequential Tree Implementations 144
6.6 Further Reading 146
6.7 Exercises 146
6.8 Projects 148

Chapter 7 Graphs 150
7.1 Terminology and Representations 150
7.2 Graph Implementations 153
7.3 Graph Traversals 160
7.3.1 Depth-First Search 161
7.3.2 Breadth-First Search 162
7.3.3 Topological Sort 163
7.4 Shortest-Paths Problems 166
7.4.1 Single-Source Shortest Paths 166
7.4.2 All-Pairs Shortest Paths 170
7.5 Minimum-Cost Spanning Trees 171
7.5.1 Prim's Algorithm 171
7.5.2 Kruskal's Algorithm 174

7.6 Further Reading 174
7.7 Exercises 176
7.8 Projects 177

Part III Sorting and Searching 179

Chapter 8 Internal Sorting 180
8.1 Sorting Terminology and Notation 180
8.2 Three $\Theta(n^2)$ Sorting Algorithms 181
8.2.1 Insertion Sort 181
8.2.2 Bubble Sort 183
8.2.3 Selection Sort 184
8.2.4 The Cost of Exchange Sorting 185
8.3 Shellsort 186
8.4 Quicksort 188
8.5 Mergesort 194
8.6 Heapsort 197
8.7 Binsort and Radix Sort 199
8.8 An Empirical Comparison of Sorting Algorithms 203
8.9 Lower Bounds for Sorting 205
8.10 Further Reading 208
8.11 Exercises 209
8.12 Projects 211

Chapter 9 File Processing and External Sorting 213
9.1 Primary Versus Secondary Storage 213
9.2 Disk and Tape Drives 215
9.2.1 Disk Access Costs 219
9.2.2 Magnetic Tape 221
9.3 Buffers and Buffer Pools 221
9.4 The Programmer's View of Files 223
9.5 External Sorting 225
9.6 Simple Approaches to External Sorting 227
9.7 Replacement Selection 228
9.8 Multiway Merging 231
9.9 Further Reading 234
9.10 Exercises 234
9.11 Projects 236

Chapter 10 Searching 238
10.1 Searching Sorted Arrays 239

10.2 Self-Organizing Lists 240
10.3 Searching in Sets 244
10.4 Hashing 244
10.4.1 Hash Functions 245
10.4.2 Open Hashing 248
10.4.3 Closed Hashing 249
10.5 Further Reading 258
10.6 Exercises 259
10.7 Projects 260

Chapter 11 Indexing 262
11.1 Linear Indexing 263
11.2 ISAM 265
11.3 Tree Indexing 267
11.4 2-3 Trees 268
11.5 B-Trees 274
11.5.1 B^+-Trees 276
11.5.2 B-Tree Analysis 282
11.6 Further Reading 282
11.7 Exercises 282
11.8 Projects 284

Part IV Applications and Advanced Topics 287

Chapter 12 Lists and Arrays Revisited 288
12.1 Skip Lists 288
12.2 Multilists 292
12.3 Matrix Representations 295
12.4 Memory Management 298
12.4.1 Dynamic Storage Allocation 298
12.4.2 Failure Policies and Garbage Collection 306
12.5 Further Reading 309
12.6 Exercises 310
12.7 Projects 311

Chapter 13 Advanced Tree Structures 312
13.1 Tries 312
13.2 Splay Trees 316
13.3 Spatial Data Structures 318
13.3.1 The K-D Tree 320
13.3.2 The PR quadtree 324

13.3.3 Other Spatial Data Structures 325
13.4 Further Reading 327
13.5 Exercises 327
13.6 Projects 328

Chapter 14 Analysis Techniques 330
14.1 Summation Techniques 330
14.2 Recurrence Relations 333
14.2.1 Estimating Upper and Lower Bounds 333
14.2.2 Expanding Recurrences 334
14.2.3 Divide and Conquer Recurrences 335
14.2.4 Average-Case Analysis of Quicksort 336
14.3 Amortized Analysis 337
14.4 Further Reading 340
14.5 Exercises 340
14.6 Projects 343

Chapter 15 Limits to Computation 344
15.1 Introduction 344
15.2 Reductions 344
15.3 Hard Problems 348
15.3.1 $\mathcal{NP}$-Completeness 349
15.3.2 Getting Around $\mathcal{NP}$-Complete Problems 352
15.4 Impossible Problems 353
15.4.1 Uncountability 353
15.4.2 The Halting Problem is Unsolvable 355
15.4.3 Determining Program Behavior is Unsolvable 357
15.5 Further Reading 358
15.6 Exercises 358
15.7 Projects 359

Part V Appendix 361

Appendix A Java Tutorial for C and Pascal Programmers 362
A.1 Example 1: Interface for Lists 362
A.2 Example 2: Array-Based List Implementation 363
A.3 Example 3: Linked List Implementation 366

Bibliography 368

Preface

We study data structures so that we can learn to write more efficient programs. But why must programs be efficient when new computers are faster every year? The reason is that our ambitions grow with our capabilities. Instead of rendering efficiency needs obsolete, the modern revolution in computing power and storage capability merely raises the efficiency stakes as we computerize more complex tasks.

The quest for program efficiency need not and should not conflict with sound design and clear coding. Creating efficient programs has little to do with "programming tricks," but rather is based on good organization of information and good algorithms. A programmer who has not mastered the basic principles of clear design will not likely write efficient programs. Conversely, clear programs require clear data organization and clear algorithms. Most computer science curricula recognize that good programming skills begin with a strong emphasis on fundamental software engineering principles. Then, once a programmer has learned the principles of clear program design and implementation, the next step is to study the effects of data organization and algorithms on program efficiency.

Approach: Many techniques for representing data are described in this book. These techniques are presented within the context of the following principles:

1. Each data structure and each algorithm has costs and benefits. Practitioners need a thorough understanding of how to assess costs and benefits to be able to adapt to new design challenges. This requires an understanding of the principles of algorithm analysis, and also an appreciation of the significant effects of the physical medium employed (e.g., data stored on disk versus main memory).
2. Related to costs and benefits is the notion of tradeoffs. For example, it is quite common to reduce time requirements at the expense of an increase in space requirements, or vice versa. Programmers face tradeoff issues regularly in all phases of software design and implementation, so the concept must become deeply ingrained.
3. Programmers should know enough about common practice to avoid reinventing the wheel. Thus, students need to learn the commonly used data structures and related algorithms.
4. Data structures follow needs. Students must learn to assess application needs first, then find a data structure with matching capabilities. To do this requires competence in principles 1, 2, and 3.

Organization: Data structures and algorithms textbooks tend to fall into one of two categories: teaching texts or encyclopedias. Books that attempt to do both usually fail at both. This book is intended as a teaching text. I believe it is more important for a practitioner to understand the principles required to select or design the data structure that will best solve some problem than it is to memorize a lot of textbook implementations. Hence, I have designed this as a teaching text that covers most standard data structures, but not all. A few data structures that are not widely adopted are used to illustrate important principles.

Some relatively new data structures that should become widely used in the future are included.

This book is intended for a single semester course at the undergraduate level, or for self-study by technical professionals. Readers should have programming experience, typically two semesters or the equivalent of a structured programming language such as Pascal or C. Prerequisite mathematical techniques are reviewed in Chapter 2; readers who are already familiar with induction proofs and recursion will have an advantage.

While this book is designed for a one semester course, there is more material here than can properly be covered in one semester. This is deliberate, and provides some flexibility to the instructor. A sophomore level class where students have little background in basic data structures or analysis might cover Chapters 1-12 in detail, as well as selected topics from Chapter 13. That is how I use the book for my own sophomore level class. Students with greater background might cover Chapter 1, skip most of Chapter 2 except for reference, briefly cover Chapters 3 and 4 (but pay attention to Section 4.1.3), and then cover the remaining chapters in detail. Again, only certain topics from Chapter 13 might be covered, depending on the programming assignments selected by the instructor.

Chapter 13 is intended in part as a source for larger programming exercises. I recommend that all students taking a data structures course be required to implement some advanced tree structure, or another dynamic structure of comparable difficulty such as the Skip List or sparse matrix representations of Chapter 12. None of these data structures are significantly more difficult to implement than the Binary Search Tree, and any of them should be within a student's ability after completing Chapter 5.

While I have attempted to arrange the presentation in an order that makes sense, instructors should feel free to re-arrange the topics as they see fit. Once the reader has mastered Chapters 1-6, the remaining material has relatively few dependencies. Clearly, external sorting depends on understanding internal sorting and disk files. Section 6.2 on the UNION/FIND algorithm is used in Kruskal's Minimum-Cost Spanning Tree algorithm. Section 10.2 on self-organizing lists mentions the buffer replacement schemes covered in Section 9.3. Chapter 14 draws on examples from throughout the book. Section 15.3 relies on knowledge of graphs. Otherwise, most topics depend only on material presented earlier within the same chapter.

Use of Java: The programming examples are written in Java™.As with any programming language, Java has both advantages and disadvantages.Java is a small language.There usually is only one way to do something, and this has the happy tendency of encouraging a programmer toward clarity when usedcorrectly. In this respect, it is superior to C or C++. Java serves nicely for defining and using most traditional data structures such as lists and trees.On the other hand, Java is quite poor when used to do fileprocessing, being both cumbersome and inefficient.It is also a poor language when fine control of memory is required. As an example, applications requiring memorymanagement, such as those discussed in Section12.4, are difficult to write in Java.Since I wish to stick to a single language throughout the text, like any programmer I must take the bad along with the good.The most important issue is to get the ideas across, whether or not those ideas are natural to a particular language of discourse.Most programmers will use a variety of programming languages throughout their career, and the concepts described in this book should prove useful in a variety of circumstances.

I do not wish to discourage those unfamiliar with Java from readingthis book.I have attempted to make the examples as clear as possible while maintaining the advantages of Java. Java is used here strictly as a tool to illustrate data structures concepts.Fortunately, Java is an easy language for C or Pascal programmers to read with a minimal amount of study of the syntax related to object-oriented programming.In particular, I make use of Java's support for hiding implementation details, including features such as classes,private class members, and interfaces.These features of the language support the crucial concept of separating logical design, as embodied in the abstract data type, from physical implementation as embodied in the data structure.

I make no attempt to teach Java within the text.An Appendix is provided that describes the Java syntax and concepts necessary to understand the program examples.I also provide the actual Java code used in the text throughanonymous FTP.

Inheritance, a key feature of object-oriented programming, is used only sparingly in the code examples. Inheritance is an important tool that helps programmers avoid duplication, and thus minimize bugs. From a pedagogical standpoint, however, inheritance often makes code examples harder to understand since it tends to spread the description for one logical unit among several classes.Thus, some of my class definitions for objects such as tree or list nodes to not take full advantage of possible inheritance from earliercode examples.This does not mean that a programmer should do likewise.Avoiding code duplication and minimizing errors are important goals.Treat the programming examples as illustrations of data structure principles, but do not copy them directly into your own programs.

My Java implementations serve to provide concrete illustrations of data structure principles.The are not meant to be a series of commercial-quality Java class implementations.The code examples provide less parameter checking than is sound programming practice for commercial programmers.Some parameter checking is included in the form of calls to functions in class `Assert`.These functions are modeled after the C standard library function `assert`.Method `Assert.notFalse` takes a Boolean expression. If this expression evaluates to `false`, then the program terminates immediately.Method `Assert.notNull` takes a reference to class `Object`,and terminates the program if the value of the reference is `null`.(To be precise, these functions throw an `IllegalArgumentException`, which typically results in terminating the program unless the programmer takes action to handlethe exception.)Terminating a program when a function receives a bad parameter isgenerally considered undesirable in real programs, but is quite adequate for understanding how a data structure ismeant to operate.In real programming applications, Java's exception handling featuresshould be used to deal with input data errors.

I make a distinction in the text between "Java implementations" and "pseudocode".Code labeled as a Java implementation has actually beencompiled and tested on one or more Java compilers.Pseudocode examples often conform closely to Java syntax, but typically contain one or more lines of higher level description.Pseudocode is used where I perceived a greater pedagogical advantage to a simpler, but less precise, description.

Most chapters end with a section entitled "Further Reading."These sections are not comprehensive lists of references on the topics presented.Rather, I include books and articles that, in my opinion,may prove exceptionally informative or entertaining to the reader.In some cases I include references to works that should becomefamiliar to any well-rounded computer scientist.

Exercises and Projects: Proper use of data structures cannot be learned simply by readinga book.

You must practice by implementing real programs,constantly comparing different techniques to see what really worksbest in a given situation.At the same time, students should also work problems to develop their analytical abilities.I provide over 300 exercises and suggestions for programming projects.I urge readers to take advantage of them.

Contacting the Author and Supplementary Materials: A book such as this is sure to contain errors and have room forimprovement.I welcome bug reports and constructive criticism.I can be reached by electronic mail via the Internet at `shaffer@cs.vt.edu`.Alternatively, comments can be mailed to:

Cliff Shaffer
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

A set of LATEX-based transparency masters for use in conjunction with this book can be obtained via anonymous FTP at `ftp.prenhall.com` in directory

`pub/esm/computer_science.s-041/shaffer/ds/supplements/transparencies`

The Java code examples are also available from this site at

`pub/esm/computer_science.s-041/shaffer/ds/code`

Online WWW pages for Virginia Tech's sophomore level data structures class can be found at URL

`http://ei.cs.vt.edu/~cs2604`

as can information about SWAN, a data structure visualization and graphical debugging tool.

This book was typeset by the author with LATEX, a macro package for TEX.The bibliography was prepared using BIBTEX.The index was prepared using `makeindex`.The figures were mostly drawn with **Xfig**.Figures 3.1 and 10.5 were partially created using Mathematica.

Acknowledgments: It takes a lot of help from a lot of people to make a book. I wish to acknowledge a few of those who helped to make this book possible. I apologize for the inevitable omissions. My department head, Jack Carroll, provided unwavering moral support of this project. Virginia Tech helped make this whole thing possible through sabbatical research leave during Fall 1994, enabling me to get this project off the ground. Mike Keenan, Lenny Heath, and Jeff Shaffer provided valuable input on early versions of the chapters. I especially wish to thank Lenny Heath for many years of stimulating discussions about algorithms and analysis (and how to teach both to students). Thanks to Layne Watson for his help with Mathematica, and to Bo Begole, Philip Isenhour, Jeff Nielsen, and Craig Struble for much technical assistance. Thanks to Steve Edwards, Mark Abrams and Dennis Kafura for answering lots of silly questions about C++ and Java.

I am truly indebted to the many reviewers of this manuscript. Reviewers include: J. David Bezek (University of Evansville), Douglas Campbell (Brigham Young University), Karen Davis (University of Cincinnati), Vijay Kumar Garg (University of Texas — Austin), Jim Miller (University of Kansas), Bruce