国外计算机科学教材系列

数据结构与算法分析

（C++版）（第二版）

A Practical Introduction to Data Structures and Algorithm Analysis, Second Edition

英文原版

[美] Clifford A. Shaffer 著

Prentice Hall

PHEI

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

[illegible]

数据结构与算法分析（C++版）（第三版）

A Practical Introduction to Data Structures and Algorithm Analysis, Third Edition

英文原版

[illegible]

电子工业出版社 [illegible]

国外计算机科学教材系列

数据结构与算法分析

（C++ 版）

（第二版）

（英文原版）

A Practical Introduction to Data Structures
and Algorithm Analysis
Second 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

本书采用程序员最爱用的面向对象C++语言来描述数据结构和算法，并把数据结构原理和算法分析技术有机地结合在一起，系统介绍了各种类型的数据结构和排序、检索的各种方法。作者非常注意对每一种数据结构不同存储方法及有关算法进行分析比较。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。

本版的重要改进在于引入了参数化的模板，从而提高了算法中数据类型的通用性，支持高效的代码重用。

本书概念清楚、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学生的教材和参考书，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：01-2002-2623

图书在版编目（CIP）数据

数据结构与算法分析（C++版）（第二版）（英文原版）/（美）谢弗（Shaffer, C. A.）著.
-北京：电子工业出版社，2002.7
（国外计算机科学教材系列）
书名原文：A Practical Introduction to Data Structures and Algorithm Analysis, Second Edition
ISBN 7-5053-7767-1

Ⅰ.数… Ⅱ.谢… Ⅲ.①数据结构－教材－英文 ②算法分析－教材－英文 ③C语言－程序设计－教材－英文
Ⅳ. TP311.12

中国版本图书馆CIP数据核字（2002）第045570号

责任编辑：马　岚　杜闽燕
印　刷　者：北京市增富印刷有限责任公司
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：33.25　字数：830千字
版　　次：2002年7月第1版　　2002年7月第1次印刷
定　　价：45.00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

教材出版委员会

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 for 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, programmers need to learn the commonly used data structures and related algorithms.
4. Data structures follow needs. Programmers 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.

Using the Book in Class: 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 included 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, and including at least some exposure to C++. Readers who are already familiar with recursion will have an advantage. Students of data structures will also benefit from having first completed a good course in Discrete Mathematics. Nonetheless, Chapter 2 attempts to give a reasonably complete survey of the prerequisite mathematical topics at the level necessary to understand their use in this book. Readers may wish to refer back to the appropriate sections as needed when encountering unfamiliar mathematical material.

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 Sections 4.1.3 and 4.2), 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 rearrange the topics as they see fit. The book has been written so that, 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 9.2 on self-organizing lists mentions the buffer replacement schemes covered in Section 8.3. Chapter 14 draws on examples from throughout the book. Section 15.2 relies on knowledge of graphs. Otherwise, most topics depend only on material presented earlier within the same chapter.

Use of C++: The programming examples are written in C++, but I do not wish to discourage those unfamiliar with C++ from reading this book. I have attempted to make the examples as clear as possible while maintaining the advantages of C++. C++ is viewed here strictly as a tool to illustrate data structures concepts, and indeed only a minimal subset of C++ is included. In particular, I make use of C++'s support for hiding implementation details, including features such as classes, private class members, constructors, and destructors. 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.

To keep the presentation as clear as possible, some of the most important features of C++ are completely avoided here. I deliberately minimize use of certain features commonly used by experienced C++ programmers such as class hierarchy, inheritance, and virtual functions. Operator and function overloading is used sparingly. C-like initialization syntax is preferred to some of the alternatives offered by C++.

While the C++ features mentioned above have valid design rationale in real programs, they tend to obscure rather than enlighten the principles espoused in this book. For example, inheritance is important to avoiding duplication and minimizing bugs. From a pedagogical standpoint, however, inheritance makes the code examples harder to understand since it tends to spread data element descriptions among several classes. Thus, my class definitions only use inheritance where inheritance is explicitly relevant to the point illustrated (e.g., Section 5.3.1). 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.

The most painful decision I had to make was whether to use templates in the code examples. In the first edition of this book, the decision was to leave templates out as it was felt that their syntax obscures the meaning of the code for those not familiar with C++. In the years following, the use of C++ in Computer Science curricula greatly expanded, and the editors and I now feel that readers of the text are more likely than before to be familiar with template syntax. Thus, templates are now used extensively throughout the code examples.

My C++ implementations provide concrete illustrations of data structure principles. If you are looking for a complete implementation of a standard data structure for use in commercial software, you should look elsewhere. The code examples are designed explicitly to illustrate how a data structure works, as an aid to the textual exposition. Code examples should not be read or used in isolation from the associated text since the bulk of each example's documentation is contained in the text, not the code. The code complements the text, not the other way around.

The code examples provide less parameter checking than is sound programming practice for professional programmers. Some parameter checking is included in the form of a call to `Assert`, which is a modified version of the C library function `assert`. The inputs to `assert` are a Boolean expression and a character string. If this expression evaluates to `false`, then the string is printed and the program terminates immediately. This behavior is generally considered undesirable in real programs but is adequate for clarifying how a data structure is meant to operate. See the Appendix for the implementation of `Assert`.

I make a distinction in the text between "C++ implementations" and "pseudocode." Code labeled as a C++ implementation has actually been compiled and tested on one or more C++ compilers. Pseudocode examples often conform closely to C++ syntax, but typically contain one or more lines of higher-level description. Pseudocode is used where I perceived a greater pedagogical advantage to a simplified, 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 become familiar to any well-rounded computer scientist.

Exercises and Projects: Proper implementation and anaysis of data structures cannot be learned simply by reading a book. You must practice by implementing real programs, constantly comparing different techniques to see what really works best in a given situation. At the same time, students should also work problems to develop their analytical abilities. I provide approximately 350 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 for improvement. I welcome bug reports and constructive criticism. I can be reached by electronic mail via the Internet at `shaffer@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 the WWW at URL

`http://www.cs.vt.edu/~shaffer/book.html`

The C++ code examples are also available from this site. Online Web pages for Virginia Tech's sophomore-level data structures class can be found at URL

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

This book was typeset by the author with LaTeX. The bibliography was prepared using BIBTeX. The index was prepared using `makeindex`. The figures were mostly drawn with `Xfig`. Figures 3.1 and 9.6 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.

Virginia Tech helped make this whole thing possible through sabbatical research leave during Fall 1994, enabling me to get the project off the ground. My department heads during the time I have written the various editions of this book, Dennis Kafura and Jack Carroll, provided unwavering moral support for this project. Mike Keenan, Lenny Heath, and Jeff Shaffer provided valuable input on early versions of the chapters. I also wish to thank Lenny Heath for many years of stimulating discussions about algorithms and analysis (and how to teach both to students). Steve Edwards deserves special thanks for spending so much time helping me on the redesign of the C++ code for the second edition, and many hours of discussion on the principles of program design. 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 Bill McQuain, Mark Abrams and Dennis Kafura for answering lots of silly questions about C++ and Java.

I am truly indebted to the many reviewers of the various editions of this manuscript. For the first edition these reviewers included 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 Maxim (University of Michigan – Dearborn), Jeff Parker (Agile Networks/Harvard), Dana Richards (George Mason University), Jack Tan (University of Houston), and Lixin Tao (Concordia University). Without their help, this book would contain many more technical errors and many fewer insights.

For the second edition, I wish to thank these reviewers: Gurdip Singh (Kansas State University), Peter Allen (Columbia University), Robin Hill (University of Wyoming), Norman Jacobson (University of California – Irvine), Ben Keller (Virginia Tech), and Ken Bosworth (Idaho State University). In addition, I wish to thank Neil Stewart and Frank J. Thesen for their comments and ideas for improvement.

Without the hard work of many people at Prentice Hall, none of this would be possible. Authors simply do not create printer-ready books on their own. Foremost thanks go to Petra Recter, Laura Steele, and Alan Apt, my editors. My production editors, Irwin Zucker for the second edition, Kathleen Caren for the original C++ version, and Ed DeFelippis for the Java version, kept everything moving smoothly during that horrible rush at the end. Thanks to Bill Zobrist and Bruce Gregory (I think) for getting me into this in the first place. Others at Prentice Hall who helped me along the way include Truly Donovan, Linda Behrens, and Phyllis Bregman. I am sure I owe thanks to many others at Prentice Hall for their help in ways that I am not even aware of.

I wish to express my appreciation to Hanan Samet for teaching me about data structures. I learned much of the philosophy presented here from him as well, though he is not responsible for any problems with the result. Thanks to my wife Terry, for her love and support, and to my daughters Irena and Kate for pleasant diversions from working too hard. Finally, and most importantly, to all of the data structures students over the years who have taught me what is important and what should be skipped in a data structures course, and the many new insights they have provided. This book is dedicated to them.

Clifford A. Shaffer
Blacksburg, Virginia

目录概览

第一部分　预备知识 …… 1
Preliminaries

第 1 章　数据结构和算法 …… 3
Data Structures and Algorithms

第 2 章　数学预备知识 …… 21
Mathematical Perliminaries

第 3 章　算法分析 …… 49
Algorithm Analysis

第二部分　基本数据结构 …… 85
Fundamental Data Structures

第 4 章　线性表、栈和队列 …… 87
Lists, Stacks, and Queues

第 5 章　二叉树 …… 141
Binary Trees

第 6 章　非二叉树 …… 191
Non–Binary Trees

第三部分　排序和检索 …… 217
Sorting and Searching

第 7 章　内排序 …… 219
Internal Sorting

第 8 章　文件管理和外排序 …… 259
File Processing and External Sorting

第 9 章　检索 …… 293
Searching

第10章 索引技术 ………… 327
Indexing

第四部分 应用与高级话题 ………… 357
Applications and Advanced Topics

第11章 图 ………… 359
Graphs

第12章 线性表和数组高级技术 ………… 393
Lists and Arrays Revisited

第13章 高级树形结构 ………… 423
Advanced Tree Structures

第14章 分析技术 ………… 451
Analysis Techniques

第15章 计算的限制 ………… 469
Limits to Computation

第五部分 附录 ………… 493
Appendix

附录A 实用函数 ………… 495
A Utility Functions

参考文献 ………… 497
Bibliography

索引 ………… 502
Index

Contents

I PRELIMINARIES 1

1 Data Structures and Algorithms 3

1.1 A Philosophy of Data Structures 4

1.1.1 The Need for Data Structures 4

1.1.2 Costs and Benefits 6

1.2 Abstract Data Types and Data Structures 8

1.3 Problems, Algorithms, and Programs 12

1.4 Further Reading 15

1.5 Exercises 17

2 Mathematical Preliminaries 21

2.1 Sets and Relations 21

2.2 Miscellaneous Notation 25

2.3 Logarithms 26

2.4 Recursion 28

2.5 Summations and Recurrences 30

2.6 Mathematical Proof Techniques 34

2.6.1 Proof by Contradiction 34

2.6.2 Proof by Mathematical Induction 35

2.7 Estimating 41

2.8 Further Reading 43
2.9 Exercises 43

3 Algorithm Analysis 49
3.1 Introduction 49
3.2 Best, Worst, and Average Cases 55
3.3 A Faster Computer, or a Faster Algorithm? 57
3.4 Asymptotic Analysis 59
3.4.1 Upper Bounds 60
3.4.2 Lower Bounds 62
3.4.3 Θ Notation 63
3.4.4 Simplifying Rules 64
3.5 Calculating the Running Time of a Program 65
3.6 Analyzing Problems 70
3.7 Common Misunderstandings 71
3.8 Multiple Parameters 72
3.9 Space Bounds 73
3.10 Some Practical Considerations 76
3.11 Further Reading 78
3.12 Exercises 79
3.13 Projects 83

II FUNDAMENTAL DATA STRUCTURES 85

4 Lists, Stacks, and Queues 87
4.1 Lists 88
4.1.1 Array-Based List Implementation 91
4.1.2 Linked Lists 95
4.1.3 Comparison of List Implementations 105
4.1.4 Element Implementations 107
4.1.5 Doubly Linked Lists 108
4.2 The Dictionary ADT 113
4.3 Stacks 119

4.3.1 Array-Based Stacks 121
4.3.2 Linked Stacks 122
4.3.3 Comparison of Array-Based and Linked Stacks 124
4.3.4 Implementing Recursion 125
4.4 Queues 128
4.4.1 Array-Based Queues 129
4.4.2 Linked Queues 133
4.4.3 Comparison of Array-Based and Linked Queues 133
4.5 Further Reading 133
4.6 Exercises 133
4.7 Projects 138

5 Binary Trees 141
5.1 Definitions and Properties 141
5.1.1 The Full Binary Tree Theorem 143
5.1.2 A Binary Tree Node ADT 145
5.2 Binary Tree Traversals 146
5.3 Binary Tree Node Implementations 149
5.3.1 Pointer-Based Node Implementations 149
5.3.2 Space Requirements 154
5.3.3 Array Implementation for Complete Binary Trees 157
5.4 Binary Search Trees 159
5.5 Heaps and Priority Queues 167
5.6 Huffman Coding Trees 174
5.6.1 Building Huffman Coding Trees 176
5.6.2 Assigning and Using Huffman Codes 182
5.7 Further Reading 185
5.8 Exercises 186
5.9 Projects 189

6 Non-Binary Trees 191
6.1 General Tree Definitions and Terminology 191
6.1.1 An ADT for General Tree Nodes 192