HZ BOOKS
华章教育

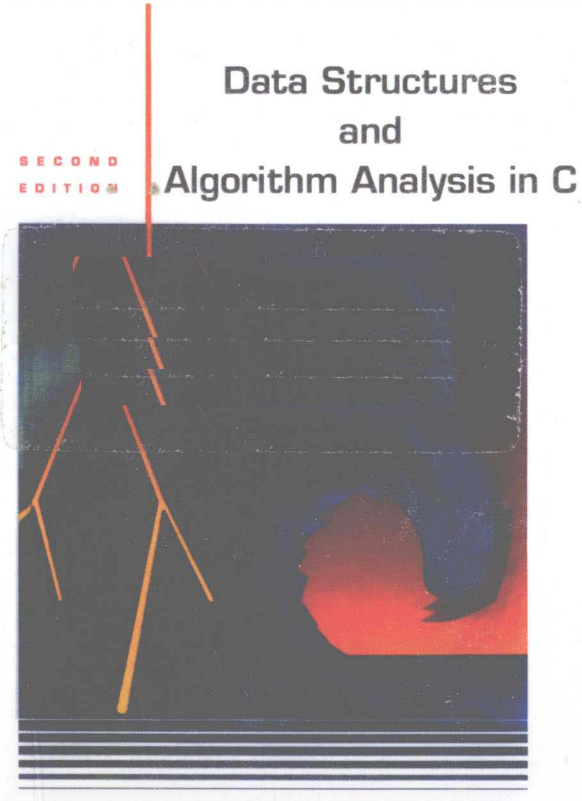
PEARSON

经　典　原　版　书　库

数据结构与算法分析

C语言描述

（英文版 · 第2版）



（美）Mark Allen Weiss 著
佛罗里达国际大学

机械工业出版社
China Machine Press

经 典 原 版 书 库

数据结构与算法分析

C语言描述

（英文版 · 第2版）

Data Structures and Algorithm Analysis in C

机械工业出版社
China Machine Press

本书版权登记号：图字：01-2010-4175

图书在版编目（CIP）数据

数据结构与算法分析：C语言描述（英文版·第2版）/（美）韦斯（Weiss, M. A.）著. —北京：机械工业出版社，2010.7
（经典原版书库）
书名原文：Data Structures and Algorithm Analysis in C，Second Edition

ISBN 978-7-111-31280-2

Ⅰ. 数…　Ⅱ. 韦…　Ⅲ. ①数据结构－英文　②算法分析－英文　③C语言－程序设计－英文　Ⅳ. ①TP311.12　②TP312

中国版本图书馆CIP数据核字（2010）第131749号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）
责任编辑：李俊竹
北京京师印务有限公司印刷
2010年8月第1版第1次印刷
150mm × 214mm · 16.5印张
标准书号：ISBN 978-7-111-31280-2
定价：45.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校

的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：(010) 88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章教育

华章科技图书出版中心

PREFACE

Purpose/Goals

This book 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 16 years 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 (CS7) course or a first-year graduate course in algorithm analysis. Students should have some knowledge of intermediate programming, including such topics as pointers and recursion, and some background in discrete math.

Approach

I believe it is important for students to learn how to program for themselves, not how to copy programs from a book. On the other hand, it is virtually impossible to discuss realistic programming issues without including sample code. For this reason, the book usually provides about one-half to three-quarters of an implementation, and the student is encouraged to supply the rest. Chapter 12, which is new to this edition, discusses additional data structures with an emphasis on implementation details.

The algorithms in this book are presented in ANSI C, which, despite some flaws, is arguably the most popular systems programming language. The use of C instead of Pascal allows the use of dynamically allocated arrays (see, for instance, rehashing in Chapter 5). It also produces simplified code in several places, usually because the *and* (&&) operation is short-circuited.

Most criticisms of C center on the fact that it is easy to write code that is barely readable. Some of the more standard tricks, such as the simultaneous assignment and testing against 0 via

```
    if (x=y)
```

are generally not used in the text, since the loss of clarity is compensated by only a few keystrokes and no increased speed. I believe that this book demonstrates that unreadable code can be avoided by exercising reasonable care.

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 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. The emphasis here is on coding these data structures using ADTs, fast implementation of these data structures, and an exposition of some of their uses. There are almost no programs (just routines), but the exercises contain plenty of ideas for programming assignments.

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 but not analyzed. Seventy-five percent of the code is written, leaving similar cases to be completed by the student. 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.

Chapter 5 is a relatively short chapter concerning hash tables. Some analysis is performed, 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. The analysis of the average-case running time of heapsort is new to this edition. 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 is new to this edition. It covers search tree algorithms, 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 under AVL trees (in Chapter 4).

Chapters 1–9 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 7–11. 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. Garey and Johnson's book on *NP*-completeness can be used 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.

A solutions manual containing solutions to almost all the exercises is available to instructors from the Addison-Wesley Publishing Company.

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.

Code Availability

The example program code in this book is available via anonymous ftp at `aw.com`. It is also accessible through the World Wide Web; the URL is `http://www.aw.com/cseng/` (follow the links from there). The exact location of this material may change.

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.

For this edition, I would like to thank my editors at Addison-Wesley, Carter Shanklin and Susan Hartman. Teri Hyde did another wonderful job with the production, and Matthew Harris and his staff at Publication Services did their usual fine work putting the final pieces together.

M.A.W.

Miami, Florida
July, 1996

CONTENTS

CHAPTER 1

Introduction

In this chapter, we discuss the aims and goals of this text and briefly review programming concepts and discrete mathematics. We will

- See that how a program performs for reasonably large input is just as important as its performance on moderate amounts of input.
- Summarize the basic mathematical background needed for the rest of the book.
- Briefly review recursion.

1.1. What's the Book About?

Suppose you have a group of N numbers and would like to determine the kth largest. This is known as the *selection problem*. Most students who have had a programming course or two would have no difficulty writing a program to solve this problem. There are quite a few "obvious" solutions.

One way to solve this problem would be to read the N numbers into an array, sort the array in decreasing order by some simple algorithm such as bubblesort, and then return the element in position k.

A somewhat better algorithm might be to read the first k elements into an array and sort them (in decreasing order). Next, each remaining element is read one by one. As a new element arrives, it is ignored if it is smaller than the kth element in the array. Otherwise, it is placed in its correct spot in the array, bumping one element out of the array. When the algorithm ends, the element in the kth position is returned as the answer.

Both algorithms are simple to code, and you are encouraged to do so. The natural questions, then, are which algorithm is better and, more important, is either algorithm good enough? A simulation using a random file of 1 million elements and k = 500,000 will show that neither algorithm finishes in a reasonable amount of time; each requires several days of computer processing to terminate (albeit