PEARSON Prentice Hall

大学计算机教育国外著名教材系列

影印版

Data Structures and Problem Solving Using Java

Fourth Edition

数据结构与问题求解

(Java语言版)(第4版)

Mark Allen Weiss 著

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Data Structures and Problem Solving Using Java

Fourth Edition

数据结构与问题求解

（Java 语言版）

（第 4 版）

清华大学出版社
北 京

图书在版编目(CIP)数据

数据结构与问题求解：Java 语言版：第 4 版：英文 /（美）韦斯（Weiss. M. A.）著. --影印本. --北京：清华大学出版社，2010.10
（大学计算机教育国外著名教材系列（影印版））
ISBN 978-7-302-23761-7

Ⅰ. ①数…　Ⅱ. ①韦…　Ⅲ. ①数据结构－高等学校－教材－英文　②Java 语言－程序设计－高等学校－教材－英文　Ⅳ. ①TP311.12　②TP312

中国版本图书馆 CIP 数据核字（2010）第 168141 号

责任编辑： 龙启铭
责任印制： 孟凡玉

出版发行： 清华大学出版社　　**地　　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：** 100084
社　总　机： 010-62770175　　**邮　购：** 010-62786544
投稿与读者服务： 010-62795954, jsjjc@tup.tsinghua.edu.cn
质　量　反　馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者： 北京密云胶印厂
装 订 者： 北京市密云县京文制本装订厂
发 行 者： 全国新华书店
开　　本： 185×230　　**印张：** 62.5
版　　次： 2010 年 10 月第 1 版　　**印　　次：** 2010 年 10 月第 1 次印刷
印　　数： 1~3000
定　　价： 99.00 元

产品编号：035087-01

出 版 说 明

进入21世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从1996年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到国内读者的欢迎和支持。跨入21世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社

preface

This book is designed for a two-semester sequence in computer science, beginning with what is typically known as Data Structures and continuing with advanced data structures and algorithm analysis. It is appropriate for the courses from both the two-course and three-course sequences in "B.1 Introductory Tracks," as outlined in the final report of the Computing Curricula 2001 project (CC2001)—a joint undertaking of the ACM and the IEEE.

The content of the Data Structures course has been evolving for some time. Although there is some general consensus concerning topic coverage, considerable disagreement still exists over the details. One uniformly accepted topic is principles of software development, most notably the concepts of encapsulation and information hiding. Algorithmically, all Data Structures courses tend to include an introduction to running-time analysis, recursion, basic sorting algorithms, and elementary data structures. Many universities offer an advanced course that covers topics in data structures, algorithms, and running-time analysis at a higher level. The material in this text has been designed for use in both levels of courses, thus eliminating the need to purchase a second textbook.

Although the most passionate debates in Data Structures revolve around the choice of a programming language, other fundamental choices need to be made:

- Whether to introduce object-oriented design or object-based design early
- The level of mathematical rigor

- The appropriate balance between the implementation of data structures and their use
- Programming details related to the language chosen (for instance, should GUIs be used early)

My goal in writing this text was to provide a practical introduction to data structures and algorithms from the viewpoint of abstract thinking and problem solving. I tried to cover all the important details concerning the data structures, their analyses, and their Java implementations, while staying away from data structures that are theoretically interesting but not widely used. It is impossible to cover all the different data structures, including their uses and the analysis, described in this text in a single course. So I designed the textbook to allow instructors flexibility in topic coverage. The instructor will need to decide on an appropriate balance between practice and theory and then choose the topics that best fit the course. As I discuss later in this Preface, I organized the text to minimize dependencies among the various chapters.

summary of changes in the fourth edition

1. This edition provides additional discussion on using classes (Chapter 2), writing classes (Chapter 3), and interfaces (Chapter 4).
2. Chapter 6 contains additional material discussing the running time of lists, the use of maps, and the use of views in the Java Collections API.
3. The `Scanner` class is described, and code throughout the text makes use of the `Scanner` class.
4. Chapter 9 describes and implements the 48-bit linear congruential generator that is part of both the Java and many C++ libraries.
5. Chapter 20 has new material on separate chaining hash tables and the `String` `hashCode` method.
6. There are numerous revisions to the text that improve on the prose in the previous edition.
7. Many new exercises are provided in Parts I, II, and IV.

a unique approach

My basic premise is that software development tools in all languages come with large libraries, and many data structures are part of these libraries. I envision an eventual shift in emphasis of data structures courses from implementation to

use. In this book I take a unique approach by separating the data structures into their specification and subsequent implementation and taking advantage of an already existing data structures library, the Java Collections API.

A subset of the Collections API suitable for most applications is discussed in a single chapter (Chapter 6) in Part Two. Part Two also covers basic analysis techniques, recursion, and sorting. Part Three contains a host of applications that use the Collections API's data structures. Implementation of the Collections API is not shown until Part Four, once the data structures have already been used. Because the Collections API is part of Java, students can design large projects early on, using existing software components.

Despite the central use of the Collections API in this text, it is neither a book on the Collections API nor a primer on implementing the Collections API specifically; it remains a book that emphasizes data structures and basic problem-solving techniques. Of course, the general techniques used in the design of data structures are applicable to the implementation of the Collections API, so several chapters in Part Four include Collections API implementations. However, instructors can choose the simpler implementations in Part Four that do not discuss the Collections API protocol. Chapter 6, which presents the Collections API, is essential to understanding the code in Part Three. I attempted to use only the basic parts of the Collections API.

Many instructors will prefer a more traditional approach in which each data structure is defined, implemented, and then used. Because there is no dependency between material in Parts Three and Four, a traditional course can easily be taught from this book.

prerequisites

Students using this book should have knowledge of either an object-oriented or procedural programming language. Knowledge of basic features, including primitive data types, operators, control structures, functions (methods), and input and output (but not necessarily arrays and classes) is assumed.

Students who have taken a first course using C++ or Java may find the first four chapters "light" reading in some places. However, other parts are definitely "heavy" with Java details that may not have been covered in introductory courses.

Students who have had a first course in another language should begin at Chapter 1 and proceed slowly. If a student would like to use a Java reference book as well, some recommendations are given in Chapter 1.

Knowledge of discrete math is helpful but is not an absolute prerequisite. Several mathematical proofs are presented, but the more complex proofs are preceded by a brief math review. Chapters 7 and 19–24 require

some degree of mathematical sophistication. The instructor may easily elect to skip mathematical aspects of the proofs by presenting only the results. All proofs in the text are clearly marked and are separate from the body of the text.

java

This textbook presents material using the Java programming language. Java is a language that is often examined in comparison with C++. Java offers many benefits, and programmers often view Java as a safer, more portable, and easier-to-use language than C++.

The use of Java requires that some decisions be made when writing a textbook. Some of the decisions made are as follows:

1. *The minimum required compiler is Java 5.* Please make sure you are using a compiler that is Java 5-compatible.
2. *GUIs are not emphasized.* Although GUIs are a nice feature in Java, they seem to be an implementation detail rather than a core Data Structures topic. We do not use Swing in the text, but because many instructors may prefer to do so, a brief introduction to Swing is provided in Appendix B.
3. *Applets are not emphasized.* Applets use GUIs. Further, the focus of the course is on data structures, rather than language features. Instructors who would like to discuss applets will need to supplement this text with a Java reference.
4. *Inner classes are used.* Inner classes are used primarily in the implementation of the Collections API, and can be avoided by instructors who prefer to do so.
5. *The concept of a pointer is discussed when reference variables are introduced.* Java does not have a pointer type. Instead, it has a reference type. However, pointers have traditionally been an important Data Structures topic that needs to be introduced. I illustrate the concept of pointers in other languages when discussing reference variables.
6. *Threads are not discussed.* Some members of the CS community argue that multithreaded computing should become a core topic in the introductory programming sequence. Although it is possible that this will happen in the future, few introductory programming courses discuss this difficult topic.

7. Some Java 5 features are not used. Including:
   - Static imports, not used because in my opinion it actually makes the code harder to read.
   - Enumerated types, not used because there were few places to declare public enumerated types that would be usable by clients. In the few possible places, it did not seem to help the code's readability.

text organization

In this text I introduce Java and object-oriented programming (particularly abstraction) in Part One. I discuss primitive types, reference types, and some of the predefined classes and exceptions before proceeding to the design of classes and inheritance.

In Part Two, I discuss Big-Oh and algorithmic paradigms, including recursion and randomization. An entire chapter is devoted to sorting, and a separate chapter contains a description of basic data structures. I use the Collections API to present the interfaces and running times of the data structures. At this point in the text, the instructor may take several approaches to present the remaining material, including the following two.

1. Discuss the corresponding implementations (either the Collections API versions or the simpler versions) in Part Four as each data structure is described. The instructor can ask students to extend the classes in various ways, as suggested in the exercises.
2. Show how each Collections API class is used and cover implementation at a later point in the course. The case studies in Part Three can be used to support this approach. As complete implementations are available on every modern Java compiler, the instructor can use the Collections API in programming projects. Details on using this approach are given shortly.

Part Five describes advanced data structures such as splay trees, pairing heaps, and the disjoint set data structure, which can be covered if time permits or, more likely, in a follow-up course.

chapter-by-chapter text organization

Part One consists of four chapters that describe the basics of Java used throughout the text. Chapter 1 describes primitive types and illustrates how to write basic programs in Java. Chapter 2 discusses reference types and illustrates

the general concept of a pointer—even though Java does not have pointers—so that students learn this important Data Structures topic. Several of the basic reference types (strings, arrays, files, and Scanners) are illustrated, and the use of exceptions is discussed. Chapter 3 continues this discussion by describing how a class is implemented. Chapter 4 illustrates the use of inheritance in designing hierarchies (including exception classes and I/O) and generic components. Material on design patterns, including the wrapper, adapter, and decorator patterns can be found in Part One.

Part Two focuses on the basic algorithms and building blocks. In Chapter 5 a complete discussion of time complexity and Big-Oh notation is provided. Binary search is also discussed and analyzed. Chapter 6 is crucial because it covers the Collections API and argues intuitively what the running time of the supported operations should be for each data structure. (The implementation of these data structures, in both Collections API-style and a simplified version, is not provided until Part Four). This chapter also introduces the iterator pattern as well as nested, local, and anonymous classes. Inner classes are deferred until Part Four, where they are discussed as an implementation technique. Chapter 7 describes recursion by first introducing the notion of proof by induction. It also discusses divide-and-conquer, dynamic programming, and backtracking. A section describes several recursive numerical algorithms that are used to implement the RSA cryptosystem. For many students, the material in the second half of Chapter 7 is more suitable for a follow-up course. Chapter 8 describes, codes, and analyzes several basic sorting algorithms, including the insertion sort, Shellsort, mergesort, and quicksort, as well as indirect sorting. It also proves the classic lower bound for sorting and discusses the related problems of selection. Finally, Chapter 9 is a short chapter that discusses random numbers, including their generation and use in randomized algorithms.

Part Three provides several case studies, and each chapter is organized around a general theme. Chapter 10 illustrates several important techniques by examining games. Chapter 11 discusses the use of stacks in computer languages by examining an algorithm to check for balanced symbols and the classic operator precedence parsing algorithm. Complete implementations with code are provided for both algorithms. Chapter 12 discusses the basic utilities of file compression and cross-reference generation, and provides a complete implementation of both. Chapter 13 broadly examines simulation by looking at one problem that can be viewed as a simulation and then at the more classic event-driven simulation. Finally, Chapter 14 illustrates how data

structures are used to implement several shortest path algorithms efficiently for graphs.

Part Four presents the data structure implementations. Chapter 15 discusses inner classes as an implementation technique and illustrates their use in the `ArrayList` implementation. In the remaining chapters of Part Four, implementations that use simple protocols (`insert`, `find`, `remove` variations) are provided. In some cases, Collections API implementations that tend to use more complicated Java syntax (in addition to being complex because of their large set of required operations) are presented. Some mathematics is used in this part, especially in Chapters 19–21, and can be skipped at the discretion of the instructor. Chapter 16 provides implementations for both stacks and queues. First these data structures are implemented using an expanding array, then they are implemented using linked lists. The Collections API versions are discussed at the end of the chapter. General linked lists are described in Chapter 17. Singly linked lists are illustrated with a simple protocol, and the more complex Collections API version that uses doubly linked lists is provided at the end of the chapter. Chapter 18 describes trees and illustrates the basic traversal schemes. Chapter 19 is a detailed chapter that provides several implementations of binary search trees. Initially, the basic binary search tree is shown, and then a binary search tree that supports order statistics is derived. AVL trees are discussed but not implemented, but the more practical red–black trees and AA-trees are implemented. Then the Collections API `TreeSet` and `TreeMap` are implemented. Finally, the B-tree is examined. Chapter 20 discusses hash tables and implements the quadratic probing scheme as part of `HashSet` and `HashMap`, after examination of a simpler alternative. Chapter 21 describes the binary heap and examines heapsort and external sorting.

Part Five contains material suitable for use in a more advanced course or for general reference. The algorithms are accessible even at the first-year level. However, for completeness, sophisticated mathematical analyses that are almost certainly beyond the reach of a first-year student were included. Chapter 22 describes the splay tree, which is a binary search tree that seems to perform extremely well in practice and is competitive with the binary heap in some applications that require priority queues. Chapter 23 describes priority queues that support merging operations and provides an implementation of the pairing heap. Finally, Chapter 24 examines the classic disjoint set data structure.

The appendices contain additional Java reference material. Appendix A lists the operators and their precedence. Appendix B has material on Swing, and Appendix C describes the bitwise operators used in Chapter 12.

chapter dependencies

Generally speaking, most chapters are independent of each other. However, the following are some of the notable dependencies.

- *Part One (Tour of Java):* The first four chapters should be covered in their entirety in sequence first, prior to continuing on to the rest of the text.
- *Chapter 5 (Algorithm Analysis)*: This chapter should be covered prior to Chapters 6 and 8. Recursion (Chapter 7) can be covered prior to this chapter, but the instructor will have to gloss over some details about avoiding inefficient recursion.
- *Chapter 6 (The Collections API)*: This chapter can be covered prior to or in conjunction with material in Part Three or Four.
- *Chapter 7 (Recursion)*: The material in Sections 7.1–7.3 should be covered prior to discussing recursive sorting algorithms, trees, the Tic-Tac-Toe case study,. and shortest-path algorithms. Material such as the RSA cryptosystem, dynamic programming, and backtracking (unless Tic-Tac-Toe is discussed) is otherwise optional.
- *Chapter 8 (Sorting Algorithms)*: This chapter should follow Chapters 5 and 7. However, it is possible to cover Shellsort without Chapters 5 and 7. Shellsort is not recursive (hence there is no need for Chapter 7), and a rigorous analysis of its running time is too complex and is not covered in the book (hence there is little need for Chapter 5).
- *Chapter 15 (Inner Classes and Implementations of ArrayLists)*: This material should precede the discussion of the Collections API implementations.
- *Chapters 16 and 17 (Stacks and Queues/Linked Lists)*: These chapters may be covered in either order. However, I prefer to cover Chapter 16 first because I believe that it presents a simpler example of linked lists.
- *Chapters 18 and 19 (Trees/ Binary Search Trees)*: These chapters can be covered in either order or simultaneously.

separate entities

The other chapters have little or no dependencies:

- *Chapter 9 (Randomization)*: The material on random numbers can be covered at any point as needed.

- *Part Three* (*Applications*): Chapters 10–14 can be covered in conjunction with or after the Collections API (in Chapter 6) and in roughly any order. There are a few references to earlier chapters. These include Section 10.2 (Tic-Tac-Toe), which refers to a discussion in Section 7.7, and Section 12.2 (cross-reference generation), which refers to similar lexical analysis code in Section 11.1 (balanced symbol checking).
- *Chapters 20 and 21* (*Hash Tables/A Priority Queue*): These chapters can be covered at any point.
- *Part Five* (*Advanced Data Structures*): The material in Chapters 22–24 is self-contained and is typically covered in a follow-up course.

mathematics

I have attempted to provide mathematical rigor for use in Data Structures courses that emphasize theory and for follow-up courses that require more analysis. However, this material stands out from the main text in the form of separate theorems and, in some cases, separate sections or subsections. Thus it can be skipped by instructors in courses that deemphasize theory.

In all cases, the proof of a theorem is not necessary to the understanding of the theorem's meaning. This is another illustration of the separation of an interface (the theorem statement) from its implementation (the proof). Some inherently mathematical material, such as Section 7.4 (*Numerical Applications of Recursion*), can be skipped without affecting comprehension of the rest of the chapter.

course organization

A crucial issue in teaching the course is deciding how the materials in Parts Two–Four are to be used. The material in Part One should be covered in depth, and the student should write one or two programs that illustrate the design, implementation, testing of classes and generic classes, and perhaps object-oriented design, using inheritance. Chapter 5 discusses Big-Oh notation. An exercise in which the student writes a short program and compares the running time with an analysis can be given to test comprehension.

In the separation approach, the key concept of Chapter 6 is that different data structures support different access schemes with different efficiency. Any case study (except the Tic-Tac-Toe example that uses recursion) can be used

to illustrate the applications of the data structures. In this way, the student can see the data structure and how it is used but not how it is efficiently implemented. This is truly a separation. Viewing things this way will greatly enhance the ability of students to think abstractly. Students can also provide simple implementations of some of the Collections API components (some suggestions are given in the exercises in Chapter 6) and see the difference between efficient data structure implementations in the existing Collections API and inefficient data structure implementations that they will write. Students can also be asked to extend the case study, but again, they are not required to know any of the details of the data structures.

Efficient implementation of the data structures can be discussed afterward, and recursion can be introduced whenever the instructor feels it is appropriate, provided it is prior to binary search trees. The details of sorting can be discussed at any time after recursion. At this point, the course can continue by using the same case studies and experimenting with modifications to the implementations of the data structures. For instance, the student can experiment with various forms of balanced binary search trees.

Instructors who opt for a more traditional approach can simply discuss a case study in Part Three after discussing a data structure implementation in Part Four. Again, the book's chapters are designed to be as independent of each other as possible.

exercises

Exercises come in various flavors; I have provided four varieties. The basic *In Short* exercise asks a simple question or requires hand-drawn simulations of an algorithm described in the text. The *In Theory* section asks questions that either require mathematical analysis or asks for theoretically interesting solutions to problems. The *In Practice* section contains simple programming questions, including questions about syntax or particularly tricky lines of code. Finally, the *Programming Projects* section contains ideas for extended assignments.

pedagogical features

- Margin notes are used to highlight important topics.
- The *Key Concepts* section lists important terms along with definitions and page references.

- The *Common Errors* section at the end of each chapter provides a list of commonly made errors.
- References for further reading are provided at the end of most chapters.

supplements

A variety of supplemental materials are available for this text. The following resources are available at `http://www.aw.com/cssupport` for all readers of this textbook:

- *Source code files* from the book. (The *On the Internet* section at the end of each chapter lists the filenames for the chapter's code.)

In addition, the following supplements are available to qualified instructors. To access them, visit `http://www.pearsonhighered.com/cs` and search our catalog by title for *Data Structures and Problem Solving Using Java*. Once on the catalog page for this book, select the link to Instructor Resources.

- *PowerPoint slides* of all figures in the book.
- *Instructor's Guide* that illustrates several approaches to the material. It includes samples of test questions, assignments, and syllabi. Answers to select exercises are also provided.

acknowledgments

Many, many people have helped me in the preparation of this book. Many have already been acknowledged in the prior edition and the related C++ version. Others, too numerous to list, have sent e-mail messages and pointed out errors or inconsistencies in explanations that I have tried to fix in this edition.

For this edition I would like to thank my editor Michael Hirsch, editorial assistant Stephanie Sellinger, senior production supervisor Marilyn Lloyd, and project manager Rebecca Lazure and her team at Laserwords. Thanks also go to Allison Michael and Erin Davis in marketing and Elena Sidorova and Suzanne Heiser of Night & Day Design for a terrific cover.

Some of the material in this text is adapted from my textbook *Efficient C Programming: A Practical Approach* (Prentice Hall, 1995) and is used with

permission of the publisher. I have included end-of-chapter references where appropriate.

My World Wide Web page, `http://www.cs.fiu.edu/~weiss`, will contain updated source code, an errata list, and a link for receiving bug reports.

M. A. W.
Miami, Florida

contents

part one **Tour of Java**