数据结构与问题求解

（Java版）（第二版）

Data Structures and Problem Solving Using Java

Second Edition

英文版

[美] Mark Allen Weiss 著

数据结构与问题求解

（Java版）（第二版）

（英文版）

Data Structures and Problem Solving Using Java
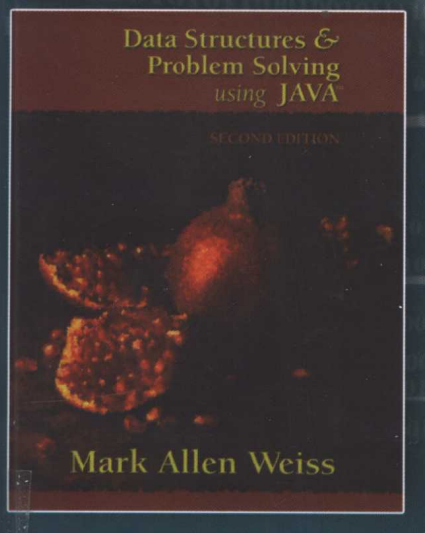
Second Edition

［美］ Mark Allen Weiss 著

电子工业出版社·

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书使用流行的 Java 语言作为描述语言, 详细介绍了数据结构和算法。全书共分为五大部分。第一部分的 Java 教程是全书的基础, 具体讲述 Java 的运行环境、数据类型和运算符、基本语法等; 同时介绍了面向对象的一些概念。第二部分对 Java 应用程序接口集 (API) 中的各种数据结构接口和其中涉及到的算法及算法分析进行了详细介绍, 并用实例说明了如何使用这些数据结构。第三部分是这些数据结构在实际中的应用, 每一章对不同应用的理论和具体实现做了详尽阐述。第四部分则针对第6章应用程序接口集中介绍过的各种数据结构接口, 分别给予更加细致的实例解说。第五部分介绍了一些高级的数据结构。

通过对本书的学习, 读者能够抽象地思考不同数据结构的功能, 了解它们之间的相关性, 掌握在计算机工程中使用这些数据结构的能力。

本书概念清楚, 逻辑性强, 内容新颖, 可作为高等院校计算机软件专业与计算机应用专业学生的双语教材和参考用书, 也可供计算机工程技术人员参考。

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

THIS book is designed for a two-semester sequence in computer science, beginning with what is typically known as Data Structures (CS-2) and continuing with advanced data structures and algorithm analysis.

The content of the CS-2 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 CS-2 courses tend to include an introduction to running-time analysis, recursion, basic sorting algorithms, and elementary data structures. An advanced course is offered at many universities 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 CS-2 revolve around the choice of a programming language, other fundamental choices need to be made, including:

- 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, and
- 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 of 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 those topics that best fit the course. As I discuss later in this Preface, I organized the text to minimize dependencies among the various chapters.

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 II. Part II also covers basic analysis techniques, recursion, and sorting. Part III contains a host of applications that use the Collections API's data structures. Implementation of the Collections API is not shown until Part IV, once the data structures have already been used. Because the Collections API is part of Java since version 1.2 (older compilers can use the textbook's Collections API code instead—see *Code Availability*, page xv), 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 IV include Collections API implementations. However, instructors can choose the simpler implementations in Part IV that do not discuss the Collections API protocol. Chapter 6, which presents the Collections API, is essential to understanding the code in Part III. 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 III and IV, 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, pages 3–25.

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.

SUMMARY OF CHANGES IN THE SECOND EDITION

1. Much of Part I was rewritten. In Chapter 2, after primitive arrays are presented, a discussion of ArrayList and the add method is introduced. Material in Chapter 3 now includes a more detailed example concerning static fields and methods. In Chapter 4, the discussion on inheritance was rewritten to simplify the initial presentation. The end of the chapter contains the more esoteric Java details that are important for advanced uses.

2. Material on design patterns has been added in various parts of the text. Several patterns, including Composite is described in Chapter 3, Wrapper, Adapter, Decorator, and Functor, are described in Chapter 4, and Iterator is described in Chapter 6.

3. The Data Structures chapter in Part II was rewritten with the Collections API in mind. Both generic interfaces (as in the first edition) and Collections API interfaces are illustrated in the revised Chapter 6.

4. The code in Part III is based on the Collections API. In several places, the code is more object-oriented than before. The Huffman coding example is completely coded.

5. In Part IV, generic data structures were rewritten to be much simpler and cleaner. Additionally, as appropriate, a simplified Collections API implementation is illustrated at the end of the chapters in Part IV. Implemented components include `ArrayList`, `LinkedList`, `Stack`, `TreeSet`, `TreeMap`, `HashSet`, `HashMap`, and various interfaces, function objects and algorithms.

JAVA

This textbook presents material using the Java programming language. Java is a relatively new 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 1.2*: Of course all code will work with Java 1.3 or 1.4. However, using Collections API requires a Java 1.2 compiler. Please make sure you are using a compiler that is Java 1.2-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 CS-2 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*: These 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 CS-2 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 multi-threaded computing should become a core CS-1/2 topic. Although it is possible that this will happen in the future, few CS-1/2 courses discuss this difficult topic.

As with every programming language, Java also has some disadvantages. It does not directly support generic programing, but a workaround is required that is discussed in Chapter 4. I/O support when using Java is minimal. The examples herein make minimal use of the Java I/O facilities.

TEXT ORGANIZATION

In this text I introduce Java and object-oriented programming (particularly abstraction) in Part I. I discuss primitive types, reference types, and some of the predefined classes and exceptions before proceeding to the design of classes and inheritance. The material in these chapters was substantially rewritten. New to this edition is material on design patterns.

In Part II, 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 IV 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 III 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 V 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 I 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 CS-2 topic. Several of the basic reference types (strings, arrays, files, and string tokenizers) 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, decorator patterns can be found in Part I.

Part II 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 IV). This chapter also introduces the iterator pattern as well as nested, local, and anonymous classes. Inner classes are deferred until Part IV, 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 III 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 lan-

guages 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 IV presents the data structure implementations. Chapter 15 is new. It discusses inner classes as an implementation technique and illustrates their use in the ArrayList implementation. In the remaining chapters of Part IV, 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 be 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. There is no priority queue in the Java 1.2 Collections API, so we implement a simple non-standard version.

Part V 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 I* (*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 III or IV.

- *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 III (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 references a discussion in Section 7.7, and Section 12.2 (cross-reference generation), which references 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 V (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 CS-2 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 II–IV are to be used. The material in Part I 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 III after discussing a data structure implementation in Part IV. 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 *Objects of the Game* 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.

CODE AND POWERPOINT SLIDES AVAILABILITY

The code in the text is fully functional and has been tested on Sun's JDK 1.2, 1.3, and 1.4 (beta). It is available from http://www.aw.com/cssupport. The *On the Internet* section at the end of each chapter lists the filenames for the chapter's code. Power Point slides of all the figures in the book are also available from this site.

INSTRUCTOR'S RESOURCE GUIDE

An *Instructor's Guide* that illustrates several approaches to the material is available. It includes samples of test questions, assignments, and syllabi. Answers to select exercises are also provided. Instructors should contact their Addison Wesley local sales representative for information on its availability. This guide is not available for sale and is available to instructors only.

ACKNOWLEDGMENTS

Many, many people have helped me in the preparation of this book. Many have already been acknowledged in the first edition and the related title, *Data Structures and Problem Solving Using C++*. 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 version.

For this book, I would like to thank all of the folks at Addison-Wesley: my editor, Susan Hartman Sullivan, and project editor, Katherine

Harutunian, helped me make some difficult decisions regarding the organization of the Java material and were very helpful in bringing this book to fruition. Gina Hagen did a lovely cover design. As always, Michael Hirsch has done a superb marketing job. I would especially like to thank Pat Mahtani, my production editor, and Caroline Roop at Argosy for their outstanding efforts coordinating the entire project.

I also thank the reviewers, who provided valuable comments, many of which have been incorporated into the text:

Divyakant Agrawal, University of California at Santa Barbara
Claude W. Anderson, Rose-Hulman Institute of Technology
David Avis, McGill University
Michael Clancy, University of California at Berkeley
Chris Eason, Valdosta State University
Tim Herman, Digital Anatomy, Inc.
Gurdip Singh, Kansas State University

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