Data Structures

RICK DECKER

TP33
D295

9161849

Data Structures

RICK DECKER

Hamilton College
Clinton, New York

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Decker, Rick
Data structures / Rick Decker.
p. cm.
Includes index.
ISBN 0-13-198813-1
1. Data structures (Computer science) I. Title.
QA76.9.D35D43. 1989
005.7'3—dc19 88-37633
CIP#

Editorial/production supervision and
interior design: David Ershun
Cover design: Diane Saxe
Cover art: Thomas Cole: *The Voyage of Life: Youth*, 1840. Oil on canvas, $52\frac{1}{2} \times 78\frac{1}{2}$ inches. *Credit:* The Munson Williams Proctor Institute Museum of Art, Utica, New York.
Manufacturing buyer: Mary Noonan

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-198813-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editoria Prentice-Hall do Brasil Ltda., *Rio de Janeiro*

Preface

The overwhelming majority of authors have very little to say. If we suppose, rather charitably, that in a typical book of fifteen chapters there are only eight passages worthy of quotation, then simple mathematics will convince us that in short order there will be no original quotations left for chapter headings. The implication is obvious. ...

ARMAND BLAGUE, *How to Write*

Over the years, I've had a number of students who have said, in one form or another,"I want to be a computer scientist because I really like programming and am very good at it." Of course, computer scientists, both novices and seasoned veterans, are often called upon to write programs, but to equate computer science with programming is to confuse the product with the process. Being an excellent draftsman who can faithfully represent a scene on paper is no guarantee that your works will eventually hang in the Metropolitan Museum. It's a step in the right direction, but an artist must also have an intimate familiarity with the more general principles of composition, persepctive, color, and so on.

In essence, programming is little more than the efficient management of a particular kind of large intellectual process, and the guidelines for good programming are nothing but the application of commonsense principles that apply to any complex creative task. It goes without saying, however, that before you can think efficiently you must have something to think *about*, which for our purposes means that in order to write good programs, you must have an idea about how information may be represented in a program.

Computer science is a young discipline, but it has developed enough over the past few decades that there is a growing consensus about what should constitute the core data structures. In this book, I tried to capture this core by providing what might be called the "classic" data structures, the most commonly applied methods for representing information in a computer program, along with the algorithms for manipulating this information. In terms of something to think *about* when thinking about programming, you have here a collection of tools that should be part of the working knowledge of any programmer.

This book is not about programming, however. Computer science is a science and, in common with other sciences, consists in the main of seeking a theoretical framework that can be used to describe the behavior of the objects it studies—in our case, computers and their programs. One of the themes that determined the form of the book is to provide a broad view of what a data structure really is. I have taken the approach that data structures are not just a collection of ad hoc declarations and procedure definitions, but, rather, that any data structure is a particular instance of an *abstract data type*, which consists of (1) a set of objects, with (2) a common logical structure defined for each of the objects, along with (3) a collection of structure-preserving operations on the objects.

I chose to define the structure of an abstract data type by specifying a *structural relation* on each set of positions. Doing so provides a natural progression of the chapters, where each new abstract data type is introduced by removing some of the structural restrictions from a prior type. Thus, we begin with lists, whose structure is defined by a linear order, and progress to trees by removing the requirement that each position have a unique successor, then to directed graphs by removing the requirement of unique predecessor, then to sets by removing all restrictions on the structural relation. Throughout this process, we see that each new abstract data type still can be described by the threefold view of a collection of sets with a structural relation and a collection of operations on those sets.

THE AUDIENCE

Though I did not set out to tailor this book to any preexisting curriculum, it turned out that it covers essentially all of CS2 and part of CS7, as described in the ACM *Curriculum '78*, and a subset of the union of CS2 and CO2, set forth in Norman Gibbs's and Alan Tucker's 1985 *Model Curriculum for a Liberal Arts Degree in Computer Science*. The material contained here should be covered early in any computer science curriculum, and I wrote this book for an audience of first- and second-year students in computer science who were familiar with a high-level language such as Pascal. A course in discrete mathematics is desirable as a pre- or corequisite for this material, but the relevant mathematical background in relations, probability, and other topics is summarized in the appendixes for those who need it.

THE CONTENTS

My intent was to write a book that could be used as the basis for a semester-length course in data structures or advanced programming. Realizing that the subject matter of this book comes at an early stage in the education of a computer scientist, I included a number of mentions, necessarily brief, of some of the interesting topics that await the student down the road. Most of the canonical sorting algorithms are covered, along with mentions of computational complexity, compiler design, unsolvable problems, NP-completeness and the fundamental paradigms for algorithms.

Chapters 1 and 2 form the introduction of the text. Chapter 1 covers the necessary preliminaries, such as program design, the definition of an abstract data type, a review of Pascal-based pseudocode and pointers, big-O estimates, and the notion of time and space estimates. Appendix A, on relations, provides a more detailed study of the material of Chapter 1. Chapter 2 is typical of subsequent chapters, in that we introduce an abstract data type, STRING, describe how it may be implemented, compare the implementations, and provide extensive application, in this case the Boyer-Moore string-matching algorithm.

In Chapters 3 and 4 we continue the investigation of linear data structures. Chapter 3 covers lists, along with related list-like structures such as doubly linked lists, sorted lists, and braids, and concludes with a discussion of memory management. Chapter 4 covers the rest of standard linear structures, stacks and queues, motivating these by applications to manipulate postfix expressions. Since a considerable amount of queue applications involve simulation, Appendix C would be appropriate at this point.

Chapter 5 provides a segue into nonlinear structures by providing an introduction to recursion and recursively defined data structures. Timing estimates for recursive algorithms are covered in depth, along with an introduction to LISP. Appendix B covers logs and exponentials, induction, and elementary combinatorics, and would be appropriate supplementary material at this stage.

Chapters 6 and 7 cover trees. Chapter 6 provides the necessary background on binary trees and their implementations, traversal algorithms, treesort and heapsort, and discusses minimal-length codes and tries. Chapter 7, which could be omitted if necessary, covers two extensions of binary search trees, namely AVL trees and B-trees.

Chapter 8 could also be optional. It covers graphs and digraphs, along with a representative sample of graph algorithms for traversal, spanning trees, minimal-cost paths, minimal spanning trees, and an introduction to complexity theory through the Traveling Salesperson Problem.

Chapter 9, on sets, describes bit vector and list implementations of sets, as well as dictionaries, and provides a comprehensive introduction to hashing. The last section of Chapter 9 discusses the UNION-FIND data structure, and could be omitted, if worse came to worse.

In Chapter 10 we consider a problem of regenerating text from a large sample and trace the development of programs to solve this problem, using a real computer/compiler system to show how time and space constraints arise in practice from choices of data structure.

ACKNOWLEDGMENTS

Stephen King will probably make more money from his next book than I'll see in the rest of my life. It's worth every penny, folks: writing is just plain hard work. A lot of people deserve praise for seeing this book of mine through to completion. Thanks go to my colleagues for reading advance copies, especially to Peter Allen of Columbia University, Brian Kernaghan of AT&T/Bell Labs, and Seymour Pollack of Washington University for their review of the manuscript; to my students for catching the countless errors in the earlier versions; to Eloise Starkweather for producing a cover that knocked my socks off; to the nice folks at Prentice Hall for being so tolerant of my first effort with them; and especially to my family and friends, none of whom will ever read *Data Structures*, but all of whom were supportive and understanding when I came dragging in after a long day's work.

Contents

PART 1

Introduction

1

Preliminaries

When making haggis, the easiest part is obtaining the ingredients.
Anyone can get a sheep's stomach and a quantity of oatmeal,
but the real trick is knowing how to combine them.

Roberta Campbell, *The Cuisine of Scotland*

The discipline of computer science is concerned with the study of problem solving with computers. Notice that we did not say that computer science *consists* of problem solving with computers, any more than mathematics consists of solving equations or music consists of producing notes. It is not enough to be able to answer the question, "How do we solve a particular problem with the help of a computer?" If it were, the study of computer science could stop after one or two introductory programming courses. Instead, the proper subject matter includes questions like

1. What are the possible different ways to solve a problem?
2. How are the solutions for a particular problem related?
3. What technique is best for a particular problem?
4. What do we mean by a "best" solution for a problem?
5. In what ways are solutions for different problems related?
6. How do we verify that we have a solution for a problem?
7. What problems can, and cannot, be solved with a computer?

Although all of these questions contain the word *problem*, they all seek answers in a context that is broader than simply solving a particular problem. In fact, all of these questions, and all of the questions of computer science, are different aspects of the same fundamental question:

What *general principles* underlie the notion of problem solving with computers?

In this text, we will concern ourselves primarily with those aspects of this fundamental question that deal with the structure of the data in a program and, to a lesser extent, with the techniques of manipulating that data.

1.1 PROGRAM DESIGN: ALGORITHMS AND DATA STRUCTURES

We can view the subject matter of computer science as the result of the process of generalizing from specific problem-solving instances, which is to say that computer science seeks to find properties common to many instances of problem solving. Program design—the writing of programs to solve a problem—proceeds in the opposite direction: from a vague notion of what needs to be done, to the writing of a program in a specific language for a specific computer. The aim of program design is captured in the most appropriate title yet invented for a book on computer science, Nicholas Wirth's *Algorithms + Data Structures = Programs*. Wirth, the developer of the Pascal language, chose his title to point out the twofold nature of a computer program: that a program consists of an algorithm describing how to manipulate information with a computer, along with a data structure that provides a logical basis for organization of that information in the computer. These two aspects of a program are intimately intertwined: Making a decision about one of the aspects often profoundly affects the other.

Algorithms

An **algorithm** is a finite list of unambiguous instructions that can be performed on a computer in such a way that the process is guaranteed to halt in a finite amount of time. "Add up the integers from 1 to 100" almost qualifies as an algorithm, except that the single instruction it uses is ambiguous—it does not provide sufficient detail for us to decide how to perform the required operation. The instruction does provide us with a useful starting point, however: Reading it, we have a clear idea of what problem we have to solve. Indeed, just getting to the point where we know what the problem is can often represent the major part of a programming task. Knowing the problem, we can now try to refine the problem into a suitable algorithm.

This simple addition problem occupies a hallowed place in mathematical folklore, and will serve as a good example of a situation in which there is more than one algorithm to solve a given problem. Karl Frederich Gauss was born in Germany in 1777, and grew to be, if not the best, then certainly one of the best mathematicians who ever lived. The story goes* that when Gauss was a boy in what would be the eighteenth-

*This story has about the same amount of truth to it as the tale of George Washington and the cherry tree, and has survived for about the same reasons.