SECOND EDITION

ESSENTIAL Algorithms

A PRACTICAL APPROACH TO COMPUTER ALGORITHMS USING PYTHON® AND C#

Rod Stephens

WILEY

Essential Algorithms

Essential Algorithms

A Practical Approach to Computer Algorithms Using Python® and C#

Rod Stephens

WILEY

Essential Algorithms: A Practical Approach to Computer Algorithms Using Python® and C#

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-1-119-57599-3
ISBN: 978-1-119-57596-2 (ebk)
ISBN: 978-1-119-57598-6 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2019933736

For Maki

About the Author

Rod Stephens started out as a mathematician, but while studying at MIT, he discovered how much fun algorithms are. He took every algorithms course MIT offered, and he has been writing complex algorithms ever since.

During his career, Rod has worked on an eclectic assortment of applications in fields such as telephone switching, billing, repair dispatching, tax processing, wastewater treatment, concert ticket sales, cartography, and training for professional football players.

Rod was a Microsoft Visual Basic Most Valuable Professional (MVP) for 15 years and has taught introductory programming courses. He has written more than 30 books that have been translated into languages from all over the world. He has also written more than 250 magazine articles covering C#, Visual Basic, Visual Basic for Applications, Delphi, and Java.

Rod's popular C# Helper website (`http://www.csharphelper.com`) receives millions of hits per year and contains tips, tricks, and example programs for C# programmers. His VB Helper website (`http://www.vb-helper.com`) contains similar material for Visual Basic programmers.

You can contact Rod at: `RodStephens@csharphelper.com`.

About the Technical Editor

John Mueller is a freelance author and technical editor. He has writing in his blood, having produced 112 books and more than 600 articles to date. The topics range from networking to artificial intelligence and from database management to heads-down programming. Some of his current books include discussions of data science, machine learning, and algorithms. His technical editing skills have helped more than 70 authors refine the content of their manuscripts. John has provided technical editing services to numerous magazines, performed various types of consulting, and he writes certification exams as well.

Be sure to read John's blog at: `http://blog.johnmuellerbooks.com/`. You can reach John on the Internet at `John@JohnMuellerBooks.com`. John also has a website at `http://www.johnmuellerbooks.com/`. Be sure to follow John on Amazon at `https://www.amazon.com/John-Mueller/`.

Credits

Senior Acquisitions Editor
Kenyon Brown

Editorial Manager
Pete Gaughan

Associate Publisher
Jim Minatel

Production Manager
Kathleen Wisor

Project Editor
Gary Schwartz

Production Editor
Athiyappan Lalith Kumar

Technical Editor
John Muller

Copy Editor
Kim Wimpsett

Proofreader
Nancy Bell

Indexer
Potomac Indexing, LLC

Cover Designer
Wiley

Acknowledgments

Thanks to Ken Brown, Devon Lewis, Gary Schwartz, Pete Gaughan, Jim Minatel, Athiyappan Lalitkumar, and everyone else at Wiley that helped make this book possible.

Thanks to longtime friend John Mueller, who provided his technical expertise to help make the information in this book as accurate as possible. (Any remaining mistakes are mine, not his.)

Thanks also to Sunil Kumar for his generous feedback on the first edition.

Contents at a glance

Contents

Introduction

Algorithms are the recipes that make efficient programming possible. They explain how to sort records, search for items, calculate numeric values such as prime factors, find the shortest path between two points in a street network, and determine the maximum flow of information possible through a communications network. The difference between using a good algorithm and a bad one can mean the difference between solving a problem in seconds, hours, or never.

Studying algorithms lets you build a useful toolkit of methods for solving specific problems. It lets you understand which algorithms are most effective under different circumstances so that you can pick the one best suited for a particular program. An algorithm that provides excellent performance with one set of data may perform terribly with other data, so it is important that you know how to pick the algorithm that is the best match for your scenario.

Even more important, by studying algorithms, you can learn general problem-solving techniques that you can apply to other problems—even if none of the algorithms you already know is a perfect fit for your current situation. These techniques let you look at new problems in different ways so that you can create and analyze your own algorithms to solve your problems and meet unanticipated needs.

In addition to helping you solve problems while on the job, these techniques may even help you land the job where you can use them! Many large technology companies, such as Microsoft, Google, Yahoo!, IBM, and others, want their programmers to understand algorithms and the related problem-solving techniques. Some of these companies are notorious for making job applicants work through algorithmic programming and logic puzzles during interviews.

The better interviewers don't necessarily expect you to solve every puzzle. In fact, they will probably learn more about you when you don't solve a puzzle. Rather than wanting to know the answer, the best interviewers want to see how you approach an unfamiliar problem. They want to see whether you throw up your hands and say the problem is unreasonable in a job interview. Or perhaps you analyze the problem and come up with a promising line of reasoning for using algorithmic approaches to attack the problem. "Gosh, I don't know. Maybe I'd search the Internet," would be a bad answer. "It seems like a recursive divide-and-conquer approach might work" would be a much better answer.

This book is an easy-to-read introduction to computer algorithms. It describes a number of important classical algorithms and tells when each is appropriate. It explains how to analyze algorithms to understand their behavior. Most importantly, it teaches techniques that you can use to create new algorithms on your own.

Here are some of the useful algorithms that this book describes:

- Numerical algorithms, such as randomization, factoring, working with prime numbers, and numeric integration
- Methods for manipulating common data structures, such as arrays, linked lists, trees, and networks
- Using more-advanced data structures, such as heaps, trees, balanced trees, and B-trees
- Sorting and searching
- Network algorithms, such as shortest path, spanning tree, topological sorting, and flow calculations

Here are some of the general problem-solving techniques this book explains:

- Brute-force or exhaustive search
- Divide and conquer
- Backtracking
- Recursion
- Branch and bound
- Greedy algorithms and hill climbing
- Least-cost algorithms
- Constricting bounds
- Heuristics

To help you master the algorithms, this book provides exercises that you can use to explore ways that you can modify the algorithms to apply them to new situations. This also helps solidify the main techniques demonstrated by the algorithms.

Finally, this book includes some tips for approaching algorithmic questions that you might encounter in a job interview. Algorithmic techniques let you solve many interview puzzles. Even if you can't use algorithmic techniques to solve every puzzle, you will at least demonstrate that you are familiar with approaches that you can use to solve other problems.

Why You Should Study Algorithms

There are several reasons why you should study algorithms. First, they provide useful tools that you can use to solve particular problems such as sorting or finding shortest paths. Even if your programming language includes tools to perform tasks that are handled by an algorithm, it's useful to learn how those tools work. For example, understanding how array and list sorting algorithms work may help you decide which of those data structures would work best in your programs.

Algorithms also teach you methods that you may be able to apply to other problems that have a similar structure. They give you a collection of techniques that you can apply to other problems. Techniques such as recursion, divide and conquer, Monte Carlo simulation, linked data structures, network traversal, and others apply to a wide variety of problems.

Perhaps most importantly, algorithms are like a workout for your brain. Just as weight training can help a football or baseball player build muscle, studying algorithms can build your problem-solving abilities. A professional athlete probably won't need to bench press weights during a game. Similarly, you probably won't need to implement a simple sorting algorithm in your project. In both cases, however, practice can help improve your game, whether it's baseball or programming.

Finally, algorithms can be interesting, satisfying, and sometimes surprising. It never ceases to amaze me when I dump a pile of data into a program and a realistic three-dimensional rendering pops out. Even after decades of study, I still feel the thrill of victory when a particularly complicated algorithm produces the correct result. When all of the pieces fit together perfectly to solve an especially challenging problem, it feels like something at least is right in the world.

Algorithm Selection

Each of the algorithms in this book was included for one or more of the following reasons:

- The algorithm is useful, and a seasoned programmer should be expected to understand how it works and how to use it correctly in programs.

- The algorithm demonstrates important algorithmic programming techniques that you can apply to other problems.
- The algorithm is commonly studied by computer science students, so the algorithm or the techniques it uses could appear in a technical interview.

After reading this book and working through the exercises, you will have a good foundation in algorithms and techniques that you can use to solve your own programming problems.

Who This Book Is For

This book is intended primarily for three kinds of readers: professional programmers, programmers preparing for job interviews, and programming students.

Professional programmers will find the algorithms and techniques described in this book useful for solving problems they face on the job. Even when you encounter a problem that isn't directly addressed by an algorithm in this book, reading about these algorithms will give you new perspectives from which to view problems so that you can find new solutions.

Programmers preparing for job interviews can use this book to hone their algorithmic skills. Your interviews may not include any of the problems described in this book, but they may contain questions that are similar enough so that you can use the techniques you learned in this book to solve them. Even if you can't solve a problem, if you recognize a structure similar to those used in one of the algorithms, you can suggest similar strategies and perhaps get partial credit.

For all the reasons explained in the earlier section "Why You Should Study Algorithms," all programming students should study algorithms. Many of the approaches described in this book are simple, elegant, and powerful, but they're not all obvious, so you won't necessarily stumble across them on your own. Techniques such as recursion, divide and conquer, branch and bound, and using well-known data structures are essential to anyone who has an interest in programming.

NOTE **Personally, I think algorithms are just plain fun! They're my equivalent of crossword puzzles or Sudoku. I love the feeling of successfully assembling a complicated algorithm and watching it work.**

They also make great conversation starters at parties. "What do you think about label setting versus label-correcting, shortest path algorithms?"

Getting the Most Out of This Book

You can learn some new algorithms and techniques just by reading this book, but to really master the methods demonstrated by the algorithms, you need to work with them. You need to implement them in some programming language. You also need to experiment, modify the algorithms, and try new variations on old problems. The book's exercises and interview questions can give you ideas for new ways to use the techniques demonstrated by the algorithms.

To get the greatest benefit from the book, I highly recommend that you implement as many of the algorithms as possible in your favorite programming language or even in more than one language to see how different languages affect implementation issues. You should study the exercises and at least write down outlines for solving them. Ideally, you should implement them, too. Often there's a reason why an exercise is included, and you may not discover it until you take a hard look at the problem. The exercises may lead you down paths that are very interesting but that are too long to squeeze into the book.

Finally, look over some of the other interview questions available on the Internet and figure out how you would approach them. In many interviews, you won't be required to implement a solution, but you should be able to sketch out solutions. And if you have time to implement solutions, you will learn even more.

Understanding algorithms is a hands-on activity. Don't be afraid to put down the book, break out a compiler, and write some actual code!

This Book's Websites

Actually, this book has two websites: Wiley's version and my version. Both sites contain the book's source code.

The Wiley web page for this book is `www.wiley.com/go/essentialalgorithms`. You also can go to `www.wiley.com` and search for the book by title or ISBN. Once you've found the book, click the Downloads tab to obtain all of the source code for the book. Once you download the code, just decompress it with your favorite compression tool.

NOTE **At the Wiley website, you may find it easiest to search by ISBN. This book's ISBN is 978-1-119-57599-3.**

The C# programs are named with a Pascal case naming convention. For example, the program that displays graphical solutions to the Tower of Hanoi puzzle for Exercise 4 in Chapter 9 is named `GraphicalTowerOfHanoi`. The corresponding Python programs are named with underscore casing as in `graphical_tower_of_hanoi.py`.

To find my web page for this book, go to `http://www.CSharpHelper.com/algorithms2e.html`.

How This Book Is Structured

This section describes the book's contents in detail.

Chapter 1, "Algorithm Basics," explains concepts you must understand to analyze algorithms. It discusses the difference between algorithms and data structures, introduces Big O notation, and describes times when practical considerations are more important than theoretical runtime calculations.

Chapter 2, "Numerical Algorithms," explains several algorithms that work with numbers. These algorithms randomize numbers and arrays, calculate greatest common divisors and least common multiples, perform fast exponentiation, and determine whether a number is prime. Some of the algorithms also introduce the important techniques of adaptive quadrature and Monte Carlo simulation.

Chapter 3, "Linked Lists," explains linked-list data structures. These flexible structures can be used to store lists that may grow, shrink, and change in structure over time. The basic concepts are also important for building other linked data structures, such as trees and networks.

Chapter 4, "Arrays," explains specialized array algorithms and data structures, such as triangular and sparse arrays, which can save a program time and memory.

Chapter 5, "Stacks and Queues," explains algorithms and data structures that let a program store and retrieve items in first-in, first-out (FIFO) or last-in, first-out (LIFO) order. These data structures are useful in other algorithms and can be used to model real-world scenarios such as checkout lines at a store.

Chapter 6, "Sorting," explains sorting algorithms that demonstrate a wide variety of useful algorithmic techniques. Different sorting algorithms work best for different kinds of data and have different theoretical run times, so it's good to understand an assortment of these algorithms. These are also some of the few algorithms for which exact theoretical performance bounds are known, so they are particularly interesting to study.

Chapter 7, "Searching," explains algorithms that a program can use to search sorted lists. These algorithms demonstrate important techniques such as binary subdivision and interpolation.

Chapter 8, "Hash Tables," explains hash tables—data structures that use extra memory to allow a program to locate specific items very quickly. They powerfully demonstrate the space-time trade-off that is so important in many programs.

Chapter 9, "Recursion," explains recursive algorithms—those that call themselves. Some problems are naturally recursive, so these techniques make solving them easier. Unfortunately, recursion can sometimes lead to problems, so this chapter also describes how to remove recursion from an algorithm when necessary.

Chapter 10, "Trees," explains highly recursive tree data structures, which are useful for storing, manipulating, and studying hierarchical data. Trees also have applications in unexpected places, such as evaluating arithmetic expressions.

Chapter 11, "Balanced Trees," explains trees that remain balanced as they grow over time. In general, tree structures can grow very tall and thin, and that can ruin the performance of tree algorithms. Balanced trees solve this problem by ensuring that a tree doesn't grow too tall and skinny.

Chapter 12, "Decision Trees," explains algorithms that attempt to solve problems that can be modeled as a series of decisions. These algorithms are often used on very hard problems, so they often find only approximate solutions rather than the best solution possible. However, they are very flexible and can be applied to a wide range of problems.

Chapter 13, "Basic Network Algorithms," explains fundamental network algorithms such as visiting all the nodes in a network, detecting cycles, creating spanning trees, and finding paths through a network.

Chapter 14, "More Network Algorithms," explains more network algorithms, such as topological sorting to arrange dependent tasks, graph coloring, network cloning, and assigning work to employees.

Chapter 15, "String Algorithms," explains algorithms that manipulate strings. Some of these algorithms, such as searching for substrings, are built into tools that most programming languages can use without customized programming. Others, such as parenthesis matching and finding string differences, require some extra work and demonstrate useful techniques.

Chapter 16, "Cryptography," explains how to encrypt and decrypt information. It covers the basics of encryption and describes several interesting encryption techniques, such as Vigenère ciphers, block ciphers, and public key