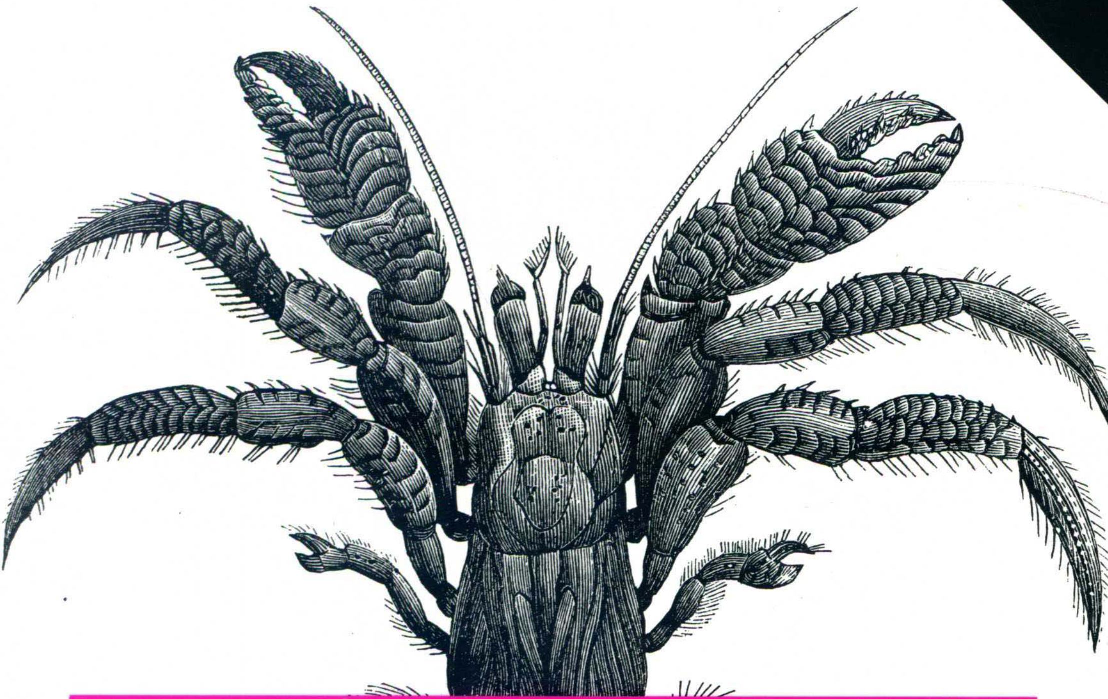
O'REILLY

第2版



Algorithms in a Nutshell

算法技术手册（影印版）

George T. Heineman,
Gary Pollice, Stanley Selkow 著

東南大學出版社

第2版

算法技术手册（影印版）

Algorithms in a Nutshell

George T. Heineman, Gary Pollice,
Stanley Selkow 著

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc.授权东南大学出版社出版

京 东南大学出版社

图书在版编目(CIP)数据

算法技术手册:第2版:英文/(美)乔治·T·海涅曼(George T. Heineman),(美)加里·波利斯(Gary Pollice),(美)斯坦利·塞克欧(Stanley Selkow)著.—影印本.—南京:东南大学出版社,2017.10

书名原文:Algorithms in a Nutshell, 2E

ISBN 978-7-5641-7373-9

Ⅰ.①算… Ⅱ.①乔… ②加… ③斯… Ⅲ.①电子计算机—算法理论—技术手册—英文 Ⅳ.①TP301.6-62

中国版本图书馆CIP数据核字(2017)195474号

图字:10-2017-341号

算法技术手册 第2版(影印版)

出版发行:东南大学出版社
地 址:南京四牌楼2号 邮编:210096
出 版 人:江建中
网 址:http://www.seupress.com
电子邮件:press@seupress.com
印 刷:常州市武进第三印刷有限公司
开 本:787毫米×980毫米 16开本
印 张:24.5
字 数:423千字
版 次:2017年10月第1版
印 次:2017年10月第1次印刷
书 号:ISBN 978-7-5641-7373-9
定 价:96.00元

本社图书若有印装质量问题,请直接与营销部联系。电话(传真):025-83791830

Preface to the Second Edition

Revising a book for a new edition is always an arduous task. We wanted to make sure that we retained all the good qualities of the first edition, published in 2009, while fixing some of its shortcomings and adding additional material. We continue to follow the principles outlined in the first edition:

- Use real code, not just pseudocode to describe algorithms
- Separate the algorithm from the problem being solved
- Introduce just enough mathematics
- Support mathematical analysis empirically

As we updated this second edition, we reduced the length of our text descriptions and simplified the layout to make room for new algorithms and additional material. We believe we continue to offer a *Nutshell* perspective on an important area of computer science that has significant impact on practical software systems.

Changes to the Second Edition

In updating this book for the second edition, we followed these principles:

Select New Algorithms

After the publication of the first edition, we often received comments such as "Why was **Merge Sort** left out?" or "Why didn't you cover **Fast Fourier Transform** (FFT)?" It was impossible to satisfy all of these requests, but we were able to add the following algorithms:

- **Fortune's algorithm**, to compute the Voronoi Diagram for a set of points ("Voronoi Diagram" on page 268)

- **Merge Sort**, for both internal memory data as well as external files ("Merge Sort" on page 81)
- Multithreaded **Quicksort** ("Parallel Algorithms" on page 332)
- **AVL Balanced Binary Tree** implementation ("Solution" on page 121)
- A new *Spatial Algorithms* chapter (Chapter 10) contains **R-Trees** and **Quadtrees**

In total, the book covers nearly 40 essential algorithms.

Streamline Presentation
: To make room for the new material, we revised nearly every aspect of the first edition. We simplified the template used to describe each algorithm and reduced the accompanying descriptions.

Add Python Implementations
: Rather than reimplement existing algorithms in Python, we intentionally used Python to implement most of the new algorithms added.

Manage Code Resources
: The code for the first edition was made available as a ZIP file. We have since transitioned to a GitHub repository (*https://github.com/heineman/algorithms-nutshell-2ed*). Over the years we improved the quality of the code and its documentation. We have incorporated a number of blog entries that were written after the publication of the first edition. There are over 500 unit test cases and we use code coverage tools to ensure coverage of 99% of our Java code. In total, the code repository consists of over 110 KLOC.

Audience

We intend this book to be your primary reference when seeking practical information on how to implement or use an algorithm. We cover a range of existing algorithms for solving a large number of problems and adhere to the following principles:

- When describing each algorithm, we use a stylized template to properly frame each discussion and explain the essential points of each algorithm
- We use a variety of languages to implement each algorithm (including C, C++, Java, and Python). In doing so, we make concrete the discussion of algorithms and speak using languages you are already familiar with
- We describe the expected performance of each algorithm and empirically provide evidence to support these claims

We intend this book to be most useful to software practitioners, programmers, and designers. To meet your objectives, you need access to a quality resource that explains real solutions to practical algorithms you need to solve real problems. You already know how to program in a variety of programming languages. You know about the essential computer science data structures, such as arrays, linked lists, stacks, queues, hash tables, binary trees, and undirected and directed graphs. You don't need to implement these data structures, since they are typically provided by code libraries.

We expect you will use this book to learn about tried and tested solutions to solve problems efficiently. You will learn some advanced data structures and novel ways to apply standard data structures to improve the efficiency of algorithms. Your problem-solving abilities will improve when you see the key decision for each algorithm that make for efficient solutions.

Conventions Used in This Book

The following typographical conventions are used in this book:

`Code`
: `All code examples appear in this typeface.`

  `This code is replicated directly from the code repository and reflects real code. All code listings are "pretty-printed" to highlight the appropriate syntax of the programming language.`

Italic
: Indicates key terms used to describe algorithms and data structures. Also used when referring to variables within a pseudocode description of an example.

`Constant width`
: Indicates the name of actual software elements within an implementation, such as a Java class, the name of an array within a C implementation, and constants such as `true` or `false`.

We cite numerous books, articles, and websites throughout the book. These citations appear in text using parentheses, such as (Cormen et al., 2009), and each chapter closes with a listing of references used within that chapter. When the reference citation immediately follows the name of the author in the text, we do not duplicate the name in the reference. Thus, we refer to the *Art of Computer Programming* books by Donald Knuth (1998) by just including the year in parentheses.

All URLs used in the book were verified as of January 2016, and we tried to use only URLs that should be around for some time. We include small URLs, such as *http://www.oreilly.com*, directly within the text; otherwise, they appear in footnotes and within the references at the end of a chapter.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/heineman/algorithms-nutshell-2ed*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Algorithms in a Nutshell, Second Edition* by George T. Heineman, Gary Pollice, and Stanley Selkow. Copyright 2016 George Heineman, Gary Pollice and Stanley Selkow, 978-1-4919-4892-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/algorithms_nutshell_2e*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

We would like to thank the book reviewers for their attention to detail and suggestions, which improved the presentation and removed defects from earlier drafts: From the first edition: Alan Davidson, Scot Drysdale, Krzysztof Duleba, Gene Hughes, Murali Mani, Jeffrey Yasskin, and Daniel Yoo. For the second edition: Alan Solis, Robert P. J. Day, and Scot Drysdale.

George Heineman would like to thank those who helped instill in him a passion for algorithms, including Professors Scot Drysdale (Dartmouth College) and Zvi Galil (Columbia University, now Dean of Computing at Georgia Tech). As always, George thanks his wife, Jennifer, and his children Nicholas (who has now started learning how to program) and Alexander (who loves making origami creations from the printed rough drafts of this edition).

Gary Pollice would like to thank his wife Vikki for 46 great years. He also wants to thank the WPI computer science department for a great environment and a great job.

Stanley Selkow would like to thank his wife, Deb. This book was another step on their long path together.

Table of Contents

1 Thinking in Algorithms

Algorithms matter! Knowing which algorithm to apply under which set of circumstances can make a big difference in the software you produce. Let this book be your guide to learning about a number of important algorithm domains, such as sorting and searching. We will introduce a number of general approaches used by algorithms to solve problems, such as the Divide and Conquer or Greedy strategy. You will be able to apply this knowledge to improve the efficiency of your own software.

Data structures have been tightly tied to algorithms since the dawn of computing. In this book, you will learn the fundamental data structures used to properly represent information for efficient processing.

What do you need to do when choosing an algorithm? We'll explore that in the following sections.

Understand the Problem

The first step in designing an algorithm is to understand the problem you want to solve. Let's start with a sample problem from the field of computational geometry. Given a set of points, *P*, in a two-dimensional plane, such as shown in Figure 1-1, picture a rubberband that has been stretched around the points and released. The resulting shape is known as the *convex hull* (i.e., the smallest convex shape that fully encloses all points in *P*). Your task is to write an algorithm to compute the convex hull from a set of two-dimensional points.

Given a convex hull for *P*, any line segment drawn between any two points in *P* lies totally within the hull. Let's assume we order the points in the hull clockwise. Thus, the hull is formed by a clockwise ordering of *h* points $L_0, L_1, \ldots, L_{h-1}$ as shown in Figure 1-2. Each sequence of three hull points L_i, L_{i+1}, L_{i+2} creates a right turn.

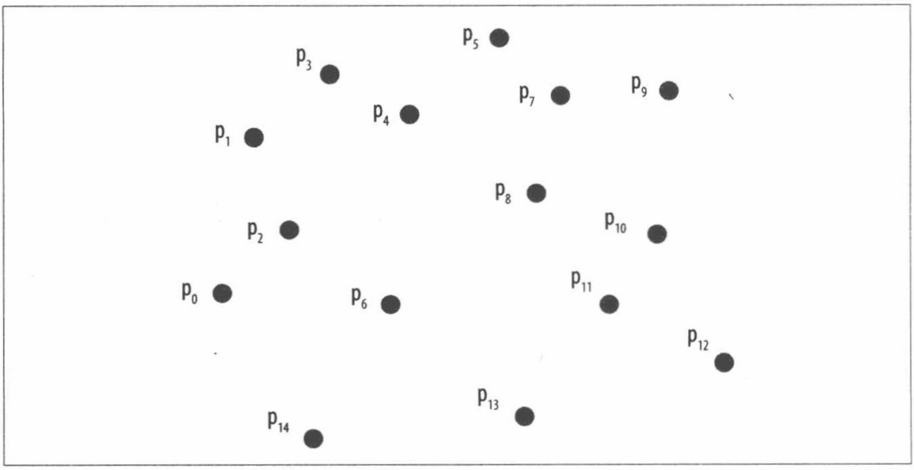


Figure 1-1. Sample set of 15 points in plane

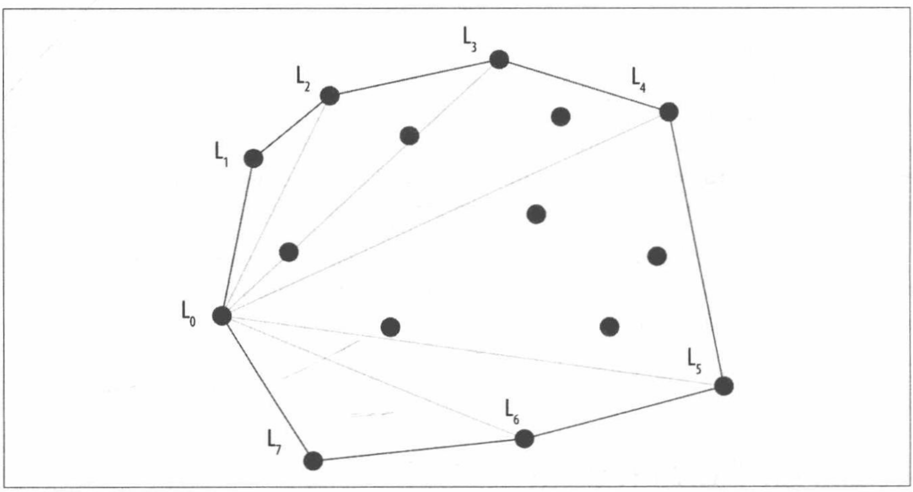


Figure 1-2. Computed convex hull for points

With just this information, you can probably draw the convex hull for any set of points, but could you come up with an *algorithm* (i.e., a step-by-step sequence of instructions that will efficiently compute the convex hull for any set of points)?

What we find interesting about the convex hull problem is that it doesn't seem to be easily classified into existing algorithmic domains. There doesn't seem to be any linear sorting of the points from left to right, although the points are ordered in clockwise fashion around the hull. Similarly, there is no obvious search being performed, although you can identify a line segment on the hull because the remaining $n - 2$ points are "to the right" of that line segment in the plane.

Naïve Solution

Clearly a convex hull exists for any collection of three or more points. But how do you construct one? Consider the following idea. Select any three points from the original collection and form a triangle. If any of the remaining $n - 3$ points are contained within this triangle, then they cannot be part of the convex hull. We'll describe the general process using pseudocode, and you will find similar descriptions for each of the algorithms in the book.

Slow Hull Summary

Best, Average, Worst: O(n^4)

```
slowHull (P)
  foreach p0 in P do
    foreach p1 in {P-p0} do
      foreach p2 in {P-p0-p1} do ❶
        foreach p3 in {P-p0-p1-p2} do
          if p3 is contained within Triangle(p0,p1,p2) then
            mark p3 as internal ❷

  create array A with all non-internal points in P
  determine leftmost point, left, in A
  sort A by angle formed with vertical line through left ❸
  return A
```

❶ Points *p0*, *p1*, *p2* form a triangle.

❷ Points *not marked* as internal are on convex hull.

❸ These angles (in degrees) range from –90 to 90.

In the next chapter, we will explain the mathematical analysis that shows why this approach is considered to be inefficient. This pseudocode summary explains the steps that produce a convex hull for each input set; in particular, it created the convex hull in Figure 1-2. Is this the best we can do?

Intelligent Approaches

The numerous algorithms in this book are the results of striving for more efficient solutions to existing code. We identify common themes in this book to help you solve your problems. There are many different ways to compute a convex hull. In sketching these approaches, we give you a sample of the material in the chapters that follow.

Greedy

Here's a way to construct the convex hull one point at a time:

1. Remove from P its lowest point, *low*, which must be part of the hull.
2. Sort the remaining $n - 1$ points in *descending* order by the angle formed in relation to a vertical line through *low*. These angles range from 90 degrees for points to the left of the line down to –90 degrees for points to the right. p_{n-2} is the rightmost point and p_0 is the leftmost point. Figure 1-3 shows the vertical line and the angles to it from each point as light lines.
3. Start with a partial convex hull formed from three points in this order $\{p_{n-2}, low, p_0\}$. Try to extend the hull by considering, in order, each of the points p_1 to p_{n-2}. If the last three points of the partial hull ever turn left, the hull contains an incorrect point that must be removed.
4. Once all points are considered, the partial hull completes. See Figure 1-3.

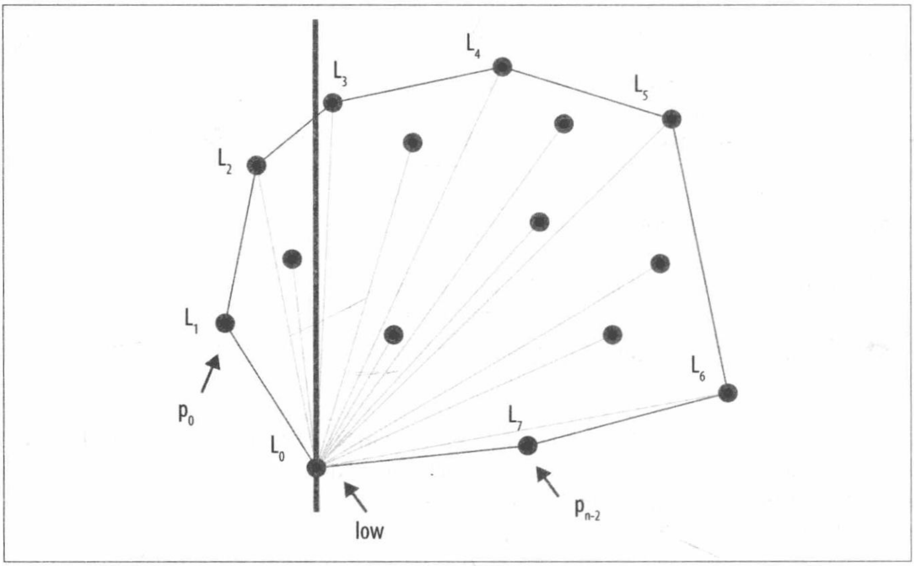


Figure 1-3. Hull formed using a Greedy approach

Divide and Conquer

We can divide the problem in half if we first sort all points, P, left to right by x coordinate (breaking ties by considering their y coordinate). From this sorted collection, we first compute the upper *partial* convex hull by considering points in order left to right from p_0 to p_{n-1} in the clockwise direction. Then the lower partial convex hull is constructed by processing the same points in order right to left from p_{n-1} to p_0 again in the clockwise direction. **Convex Hull Scan** (described in Chapter 9) computes these partial hulls (shown in Figure 1-4) and merges them together to produce the final convex hull.

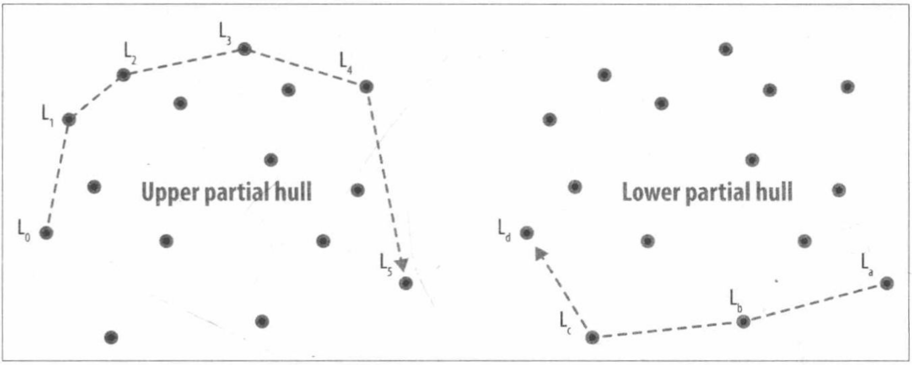


Figure 1-4. Hull formed by merging upper and lower partial hulls

Parallel

If you have a number of processors, partition the initial points by *x* coordinate and have each processor compute the convex hull for its subset of points. Once these are completed, the final hull is *stitched* together by the repeated merging of neighboring partial solutions. A parallel approach divides subproblems among a number of processors to speed up the overall solution.

Figure 1-5 shows this approach on three processors. Two neighboring hulls are stitched together by adding two tangent lines—one on the top and one on the bottom—and then eliminating the line segments contained within the quadrilateral formed by these two lines.

Approximation

Even with these improvements, there is still fixed *lower bound* performance for computing the convex hull that cannot be beaten. However, instead of computing the exact answer, perhaps you would be satisfied with an approximate answer that can be computed quickly and whose error *can be accurately determined.*

The **Bentley–Faust–Preparata** algorithm constructs an approximate convex hull by partitioning the points into vertical strips (Bentley et al., 1982). Within each strip, the maximum and minimum points (based on *y* coordinate) are identified (they are drawn in Figure 1-6 with squares around the points). Together with the leftmost point and the rightmost point in *P*, these extreme points are stitched together to form the approximate convex hull. In doing so, it may happen that a point falls outside the actual convex hull, as shown for point p_1 in Figure 1-6.