HZ BOOKS
华章IT

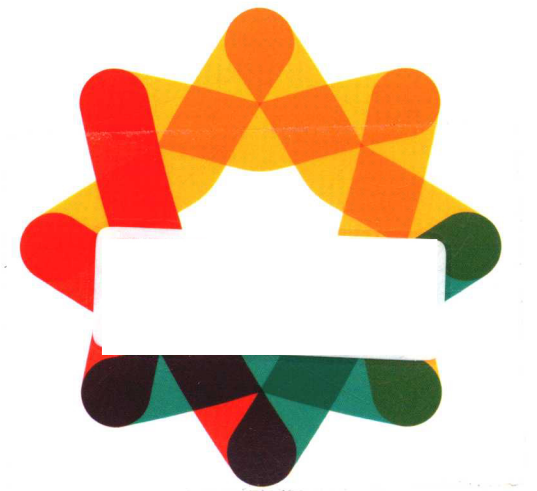
涵盖世界知名IT公司技术面试的程序设计问题及其解题思路
解析IT顶尖企业（微软、谷歌、亚马逊、雅虎、脸谱、苹果、Adobe）的面试过程
针对不同问题，提供多个具有不同复杂度的解决方法。兼顾自学教材和面试辅导的不同需求

数据结构与算法经典问题解析

Java语言描述

（原书第2版）

[印] 纳拉辛哈·卡鲁曼希 (Narasimha Karumanchi) 著　骆嘉伟 李晓鸿 肖正 吴帆 等译



DATA STRUCTURES AND ALGORITHMS MADE EASY IN JAVA

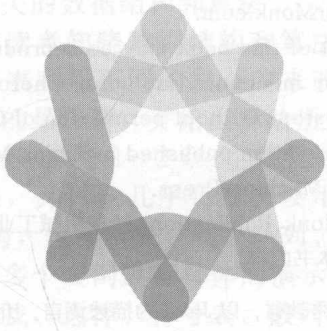
SECOND EDITION

机械工业出版社
China Machine Press

数据结构与算法经典问题解析

Java语言描述

（原书第2版）



DATA STRUCTURES AND
ALGORITHMS MADE EASY IN JAVA

SECOND EDITION

[印] 纳拉辛哈·卡鲁曼希 (Narasimha Karumanchi) 著　骆嘉伟 李晓鸿 肖正 吴帆 等译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

数据结构与算法经典问题解析：Java 语言描述（原书第 2 版）/（印）纳拉辛哈·卡鲁曼希（Narasimha Karumanchi）著；骆嘉伟等译．—北京：机械工业出版社，2016.5
书名原文：Data Structures and Algorithms Made Easy in Java, Second Edition

ISBN 978-7-111-53845-5

I. 数… II. ①纳… ②骆… III. ①数据结构 ②算法分析 ③ JAVA 语言－程序设计
IV. ① TP311.12 ② TP312

中国版本图书馆 CIP 数据核字（2016）第 114674 号

本书版权登记号：图字：01-2016-0676

本书是一本数据结构方面的优秀教材，以 Java 为描述语言，介绍了计算机编程中使用的数据结构和算法。本书强调问题及其分析，而非理论阐述，共分为 21 章，讲述了基本概念、递归和回溯、链表、栈、队列、树、优先队列和堆、并查集 DAT、图算法、排序、查找、选择算法（中位数）、符号表、散列、字符串算法、算法设计技术、贪婪算法、分治算法、动态规划算法、复杂度类型等内容。每章首先阐述必要的理论基础，然后给出问题集。全书中大约有 700 个算法问题及相应的解法，对于许多问题，本书提供了多个具有不同复杂度的解决方法。

本书可作为高等院校计算机及其相关专业的数据结构课程的教材或教学参考书，同时也可以作为从事计算机研究与开发的技术人员的参考书，特别是对正在准备面试、参加选拔性考试以及校园面试的读者尤为有用。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：盛思源　　责任校对：殷　虹
印　　刷：北京诚信伟业印刷有限公司　　版　　次：2016 年 6 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：28.5
书　　号：ISBN 978-7-111-53845-5　　定　　价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

The Translator's Words 译者序

数据结构是计算机科学与技术专业非常重要的一门核心基础课，计算机科学各个领域及各种应用软件都要使用相关的数据结构和算法。

作者编著本书的目的是使读者知晓数据结构和算法的设计原理和实现，而并非单纯地讲述定理及证明。为此，本书利用不同的复杂度来改善问题的解。对于许多问题，从穷举解法开始，逐步引入问题的最佳解，并给出算法所需的运行时间和空间。

全书包含4个部分，第一部分(第1～2章)主要描述抽象数据类型，给出算法的基本概念和复杂度分析与评价方法，并讨论几乎每章都要用到的递归和回溯技术。第二部分(第3～9章)介绍基本数据结构，包括链表、栈、队列、树、优先队列、堆、并查集和图，对于每一种数据结构分别采用多个实例进行具体的演示。第三部分(第10～15章)介绍数据处理的技术，包括排序、查找、选择、符号表、散列和字符串算法。第四部分(第16～21章)重点介绍一些常用的算法设计技术及应用，包括贪婪算法、分治算法、动态规划算法、复杂度类型，并讨论对于面试和考试的一些有用话题。本书强调问题及分析，而不侧重于理论。本书可作为高等院校计算机及其相关专业的数据结构课程的教材或教学参考书，同时也可以作为从事计算机研究与开发的技术人员的参考书，特别是对正在准备面试、参加选拔性考试以及校园面试的读者尤为有用。

本书的前言、第9～13章由骆嘉伟翻译，第2～5章、第15章由李晓鸿翻译，第16～21章由肖正翻译，第1章、第6～8章由吴帆翻译，第14章由朱宁波翻译。此外，梁成、夏艳、罗洪、张天伍、杨亦、齐逸也参与了部分翻译工作。

尽管我们从事数据结构和算法的教学和科研工作多年，在翻译过程中本着认真负责、力求精准的精神，但错误难免，希望广大读者批评指正。

译者

2015年12月于长沙

前　言 *Preface*

我知道许多读者往往不读前言，但是强烈建议你至少浏览一下本书前言，因为本书前言与众不同。

本书的主要目的不是提供关于数据结构和算法的定理及证明。本书采用的模式是利用不同的复杂度改善问题的解(对于每个问题，你将发现多个具有不同复杂度及降低复杂度的解法)。基本上，这一思路就是列举某个问题的所有可能解。通过这种方式，即使你遇到一个新问题，它也能够向你指明如何思考该问题所有可能的解。本书对于正在准备面试、参加选拔性考试以及校园面试的读者很有帮助。

作为一个求职者，如果你能完整地阅读本书并且很好地领会书中的内容，相信你会从容地面对面试官，这正是本书的目的所在。若作为一个教师来阅读本书，你将能够用简单的方法来提升授课质量，学生也会为选择攻读计算机科学/信息技术学位而感到欣慰。

作为准备参加计算机科学/信息技术专业选拔考试的学生，本书完整而详细地涵盖了所有必需的主题，在撰写本书时，就着眼于帮助正在准备这些考试的学生。

本书对攻读工程学位的学生和研究生都非常有用。在所有的章节中，你会发现本书更强调问题及其分析，而不是理论的阐述。每一章将首先阐述必要的理论基础，然后再给出问题集。书中大约有 700 个算法问题及相应的解。

对于许多问题，本书提供了多个具有不同复杂度的解决方法。我们从蛮力法开始，逐步引入问题的最佳解决方法。对于每一个问题，我们试图知晓算法所需的运行时间和内存空间。

建议读者至少完整地阅读本书一遍以便充分理解所有的主题。在随后的阅读中，你可以直接选择任何一章阅读和参考。即便经过足够的校阅，书中出现小纰漏也在所难免。如果发现了任何此类错误，www. CarrerMonk. com 网站将予以更新，请经常关注本网站以便及时了解任何勘误、新问题和解决方法。此外，请提供宝贵建议至 Info@ CarrerMonk. com。

祝愿你一切顺利。我相信你会发现本书很有用。

致谢

感谢我的父母，他们为我所做的一切无法衡量，是他们给予的无私的爱、提供的安

定的成长环境和坚持不懈的传统价值观，教会了我赞美和拥抱生活。他们是这世界上最好的父母和榜样，他们使我明白信念、勤奋和决心能够让任何事成为可能！

本书的撰写得到了许多人的帮助，没有他们的帮助本书不可能完成。感谢他们为改进本书终稿所做出的努力。需要说明的是，我已经尽最大努力纠正了审稿人所指出的错误并准确地对各种协议和机制进行了描述。我个人对书中出现的任何其他错误负责。

首先，感谢那些在本书撰写过程中陪我度过难关的人，感谢所有给予我支持的人，感谢所有参与讨论、阅读、编写和提出宝贵意见的人，感谢所有允许我引用他们的评论并协助我编辑、校对和设计本书的人。特别地，我要感谢如下人员：

- Mohan Mullapudi，印度理工学院孟买分校，架构师，dataRPM Pvt. Ltd.
- Navin Kumar Jaiswal，资深咨询师，Juniper Networks Inc.
- Kishore Kumar Jinka，印度理工学院孟买分校
- A. Vamshi Krishna，印度理工学院坎普尔分校，Mentor Graphics Inc.
- Hirak Chatterjee，Yahoo Inc.
- Kondrakunta Murali Krishna，科技学士，技术主管，HCL
- Chaganti Siva Rama Krishna Prasad，创始人，StockMonks Pvt. Ltd.
- Naveen Valsakumar，联合创始人，NotionPress Pvt. Ltd.
- Ramanaiah，讲师，龙树科技学院，MLG

最后，感谢 Guntur Vikas 学院主任 Y. V. Gopala Krishna Murthy 教授、Ayub Khan 教授（ACE 工程学院）、T. R. C. Bose（APTransco 前任主任）、Ch. Venkateswara Rao VNR Vignanajyothi（工程学院，Hyderabad）、Ch. Venkata Narasaiah（IPS）、Yarapathineni Lakshmaiah（Manchikallu，Gurazala），以及所有在本项目期间帮助过我和家人的所有好心人。

——Narasimha Karumanchi
印度理工学院孟买分校理科硕士
CareerMonk. com 创始人

目　录 *Contents*

第1章 Chapter 1

绪 论

本章的目的是阐述算法分析的重要性、它们的表示法和关系，并尽可能求解多个问题。首先，让我们重点关注算法的基本要素、分析的重要性，然后再逐步讨论上述提及的其他主题。在完成本章的学习后，能够分析任意给定算法的复杂度(特别是递归函数)。

1.1 变量

在开始讨论变量的定义之前，先来看看以前学习过的数学方程式，它与变量是有关联的。许多人从初中阶段就学会了求解许多数学方程式。例如，考虑如下方程式：

$$x^2 + 2y - 2 = 1$$

我们不必关心这个方程式用在什么地方。需要理解的重点是，这个方程式包含一些名称(x 和 y)，它们能保存值(数据)。即*名称*(x 和 y)是用来表示数据的占位符。类似地，在计算机科学中，也需要一些东西来保存数据，*变量*就是实现这一功能的方式。

1.2 数据类型

在上述方程式中，变量 x 和 y 能表示任意值，例如整数(10、20)、实数(0.23、5.5)或仅仅是 0 和 1。为了求解该方程式，需要将它们与其所能表示的数据值的类型关联起来。在计算机科学中，*数据类型*就用于这一目的。

编程语言中的*数据类型*是指具有预定义值的一个数据集合。典型的数据类型有：整数、浮点数、字符、字符串等。

计算机内存中全由 0 和 1 填充。如果直接用 0 和 1 来编码求解问题是非常困难的。为了帮助程序员，编程语言和编译器提供了数据类型。

例如，*整数*(integer)数据类型占 2 字节(具体的字节数与编译器有关)，*浮点*(float)数据类型占 4 字节等。这就是说，在内存中将 2 个字节组合起来(16 位)可称为整数。类似

地，将 4 个字节组合起来(32 位)可称为浮点数。数据类型可以减少编码的工作量。在顶层，有两种数据类型：

- 系统定义的数据类型(也称为基本数据类型)。
- 用户定义的数据类型。

1. 系统定义的数据类型(基本数据类型)

系统定义的数据类型叫作基本数据类型。许多编程语言提供的基本数据类型有：int、float、char、double、bool 等。每一个基本数据类型分配的位数与编程语言、编译器和操作系统有关。对于相同的基本数据类型，不同的编程语言可能使用不同的大小。根据数据类型的大小变化，总的有效数据值(值域)也是变化的。

例如，int 可能占 2 字节或 4 字节。如果占 2 字节(16 位)，则总的取值范围为 $-32\ 768 \sim 32\ 767(-2^{15} \sim 2^{15}-1)$；如果占 4 字节(32 位)，那么总的取值范围为 $-2\ 147\ 483\ 648 \sim 2\ 147\ 483\ 648(-2^{31} \sim 2^{31}-1)$。其他数据类型也是这样。

2. 用户定义的数据类型

如果系统定义的数据类型不够，那么大多数编程语言允许用户定义自己的数据类型，称为用户定义的数据类型。用户定义的数据类型的经典实例是：C/C++ 中的结构体和 Java 中的类。

例如，下面的代码片段把许多系统定义的数据类型组合在一起，然后用新的名字“newType”将其表示为用户定义的数据类型。这样可以更加灵活和方便地处理计算机内存。

```
public class newType {
   public int data1;
   public int data 2;
   private float data3;
   ...
   private char data;
   // 操作
}
```

1.3 数据结构

基于上述讨论，一旦变量中有数据，就需要一种操纵这些数据的机制来求解问题。*数据结构*(data structure)就是计算机中存储和组织数据的一种特定方式，它将使得数据处理更加有效。一个*数据结构*就是一种组织和存储数据的特定形式。常用的数据结构包括数组、文件、链表、栈、队列、树和图等。

根据元素的组织方式，数据结构可以分为两种类型：

1) *线性数据结构*：可以按线性次序访问元素，但它并不强制将所有元素连续地存储在一起(例如，链表)。例如，链表、栈和队列。

2) *非线性数据结构*：这种数据结构的元素是以非线性次序来存储和访问的。例如，树和图。

1.4 抽象数据类型

在定义抽象数据类型前，先从不同的视角分析系统定义的数据类型。我们都知道，在默认情况下，所有基本数据类型(int、float 等)都支持基本运算，如加法和减法。系统

实现了这些基本数据类型的基本运算。对于用户自定义的数据类型，也需要定义相应的运算。在实际使用这些运算时需要实现这些运算。即，通常用户定义的数据类型需要与它们的运算一起定义。

为简化求解问题的过程，将数据结构和相关的运算组合起来，将其称为*抽象数据类型*(ADT)。一个 ADT 包含两个部分：

1) 数据的声明。

2) 运算的声明。

常用 ADT 包括：链表、栈、队列、优先队列、二叉树、字典、并查集(并和查找)、散列表、图和许多其他类型。例如，栈使用后进先出(LIFO)机制将数据存储到数据结构中，最后插入栈的元素第一个被删除。它的常用操作有：创建栈、入栈、出栈、查找栈顶元素、在栈中查找某一个元素等。

当定义 ADT 时，不需要考虑其实现细节。仅当需要使用它们时才考虑具体的实现。不同的 ADT 类型适合不同的应用。有些是用于高度专业化的特定任务。到本书结束时，我们将探讨许多 ADT。读者也能够学会将相关数据结构与需要解决的问题进行关联。

1.5 什么是算法

考虑一个煎蛋的问题。为了煎蛋，通常有如下步骤：

1) 准备一个平底锅。

2) 准备油。

a) 有油吗?

i. 如果有，就放油到锅中。

ii. 如果没有，需要去买油吗?

① 如果是，那么就出门买油。

② 如果否，那么就停止煎蛋。

3) 打开炉子……

对于一个给定的问题(煎蛋)，我们所要做的就是通过一步一步的过程来解决它。算法的形式化定义就是：

算法就是用一条接一条的指令来解决给定的问题。

注意： *不需要证明算法的每一步。*

1.6 为什么需要算法分析

从城市 A 到城市 B，可以有许多种方式：可以坐飞机、坐汽车、坐火车，也可以骑自行车。根据可用性和方便性，可以选择最合适的一种方式。类似地，在计算机科学领域中，对于同一个问题，也存在多个有效算法来解决它(例如，排序问题就有许多算法，如插入排序、选择排序、快速排序等其他多种算法)。算法分析能够帮助我们确定在时间和空间开销方面哪一种算法是有效的。

1.7 算法分析的目的

算法分析的目标是根据运行时间及其他的一些因素(如内存、开发者的工作量等)来比较算法(或解决方案)的优劣。

1.8 什么是运行时间分析

当问题的规模(输入规模)增大时，它研究问题的处理时间是如何增加的。输入规模是指输入元素的个数。根据问题的类型，输入可能有不同的类型。下面是常用的输入类型：

- 数组的大小。
- 多项式的次数。
- 矩阵中元素的个数。
- 用二进制数表示的输入的位数。
- 图中的顶点和边。

1.9 如何比较算法

为了比较算法，首先定义几个客观评价指标。

执行时间 它不是一个好的评价指标，因为执行时间与特定的计算机有关。

执行的语句数 它不是一个好的评价指标，因为执行的语句数与编程语言有关，也与程序员个人的编程风格有关。

理想的解决方案 假设用一个函数(如 $f(n)$)来表示一个算法的运行时间，该函数的输入参数就是问题的规模 n。然后比较这些不同函数所对应的运行时间。这种比较与机器时间、编程风格等无关。

1.10 什么是增长率

所谓增长率就是随着输入规模的增加，算法运行时间增加的速度，它是输入规模的函数。假设你去商店买一辆小车和一辆自行车，这时你遇到了你的朋友，当他问你买什么东西时，通常你会这么回答他，我来买小车。这是因为比起自行车的花费，小车的花费要大得多得多(自行车的花费被小车的花费忽略掉了)。

$$\text{总花费} = \text{小车的花费} + \text{自行车的花费}$$

$$\text{总花费} \approx \text{小车的花费(近似值)}$$

对于上述例子，我们可以把总成本写成小车花费和自行车花费的函数。对于一个给定的函数，我们忽略了相对微不足道的低阶因变量(当输入规模 n 很大时)。例如，一个由 n^4、$2n^2$、$100n$ 和 500 相加组成的函数表达式，其函数值近似为 n^4。因为 n^4 的增长率最大。

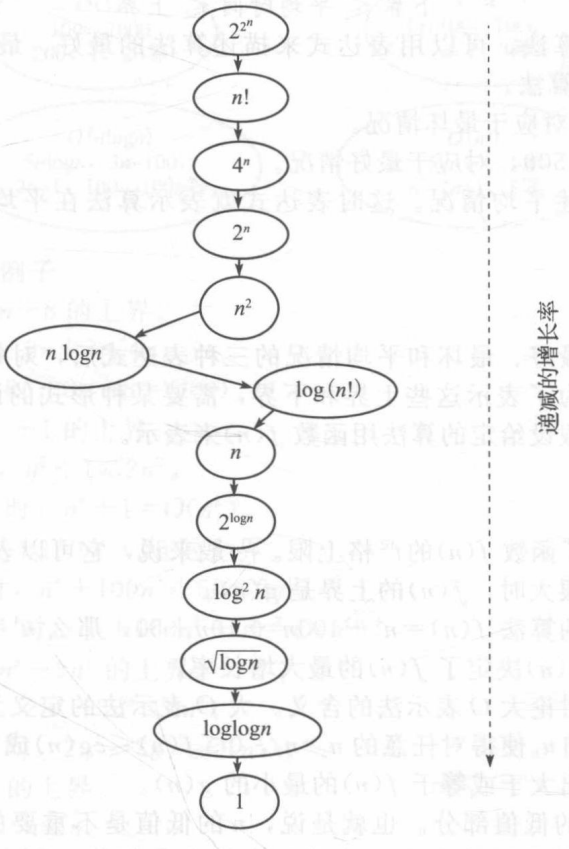
$$n^4 + 2n^2 + 100n + 500 \approx n^4$$

1.11 常用的增长率

下面给出的是一些常用的增长率，它们将在之后的章节中进行介绍。

时间复杂度	名称	实例	时间复杂度	名称	实例
1	常数	在链表的前端增加一个元素	n^2	平方	求图中两个顶点之间的最短距离
$\log n$	对数	在有序数组中查找一个元素	n^3	立方	矩阵乘法
n	线性	在无序数组中查找一个元素	2^n	指数	汉诺塔问题的求解
$n\log n$	线性对数	通过分治——归并排序 n 个元素			

下图显示了不同增长率之间的关系。



1.12 分析的类型

为了分析给定的算法，需要知道在什么输入下算法运行时间最短(性能更优)，又是在什么输入下算法运行时间最长。我们知道，算法运行时间可以用表达式来描述。这意味着可以用多个表达式来描述一个算法：其中一个表达式表示该算法所需的最短时间，而另一个表达式则表示该算法所需的最长时间。通常把前一种情况称为算法的最好情况，后者称为算法的最坏情况。为了分析算法，需要某种类型语法，这些语法是渐近分析/表示法的基础。

算法分析有三种类型：

- 最坏情况
  - 定义算法最长运行时间的输入。
  - 这种输入使算法运行最慢。
- 最好情况
  - 定义算法最短运行时间的输入。
  - 这种输入使算法运行最快。
- 平均情况
  - 提供算法运行时间的预测值。

○ 假设输入是随机的，

$$\text{下界} \leqslant \text{平均时间} \leqslant \text{上界}$$

对于一个给定的算法，可以用表达式来描述算法的最好、最坏和平均情况。例如，函数 $f(n)$代表给定的算法。

$f(n)=n^2+500$，对应于最坏情况。

$f(n)=n+100n+500$，对应于最好情况。

同样，也可以描述平均情况。这时表达式就表示算法在平均运行时间(或空间)的输入。

1.13 渐近表示

有了描述算法的最好、最坏和平均情况的三种表达式后，对每种表达式还需要确定算法的上界和下界。为了表示这些上界和下界，需要某种形式的语法，这也就是接下来要讨论的主题。这里假设给定的算法用函数 $f(n)$来表示。

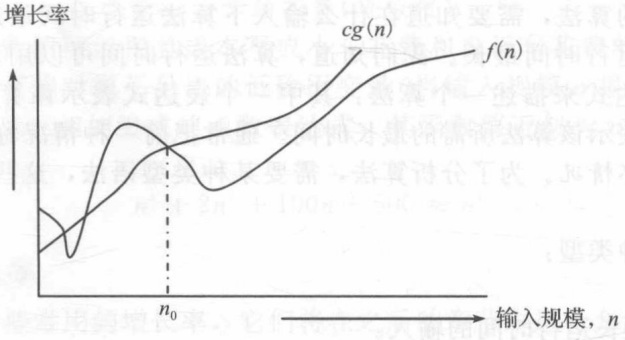
1.14 大 O 表示法

大 O 表示法给出了函数 $f(n)$的严格上限。一般来说，它可以表示为 $f(n)=O(g(n))$。这表示当输入规模 n 很大时，$f(n)$的上界是 $g(n)$。

例如，对于给定的算法 $f(n)=n^4+100n^2+10n+50$，那么 $n^4=g(n)$。这意味着随着问题规模 n 的增大，$g(n)$决定了 $f(n)$的最大增长率。

下面更加详细地讨论大 O 表示法的含义。大 O 表示法的定义为，$O(g(n))=\{f(n)$：存在这样的正常数 c 和 n_0使得对任意的 $n\geqslant n_0$，$0\leqslant f(n)\leqslant cg(n)$成立$\}$，则 $g(n)$是 $f(n)$的渐近上界。目的是求出大于或等于 $f(n)$的最小的 $g(n)$。

通常我们舍弃 n 的低值部分。也就是说，n 的低值是不重要的。在下图中，n_0是一个临界点，在此处需要分析给定算法的增长率的变化规律。而在 n_0的下面增长率可能不同。



1. 大 O 图示法

$O(g(n))$是指所有与 $g(n)$具有相同增长率或比其增长率小的函数的集合。例如，$O(n^2)$包括 $O(1)$、$O(n)$、$O(n\log n)$等。

注意： 只在输入规模 n 很大时才分析算法的复杂度，即当 $n<n_0$时不考虑算法运行时间的增长速率。