HZ BOOKS
华章教育

计 算 机 科 学 丛 书

PEARSON

原书第3版

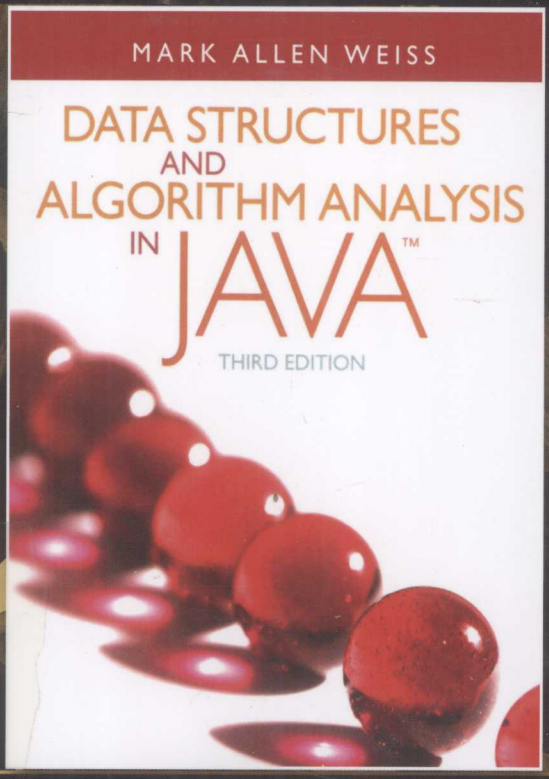
数据结构与算法分析

Java语言描述

[美] 马克·艾伦·维斯（Mark Allen Weiss）
佛罗里达国际大学
著 冯舜玺 陈越 译

Data Structures and Algorithm Analysis in Java

Third Edition



机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

原书第3版

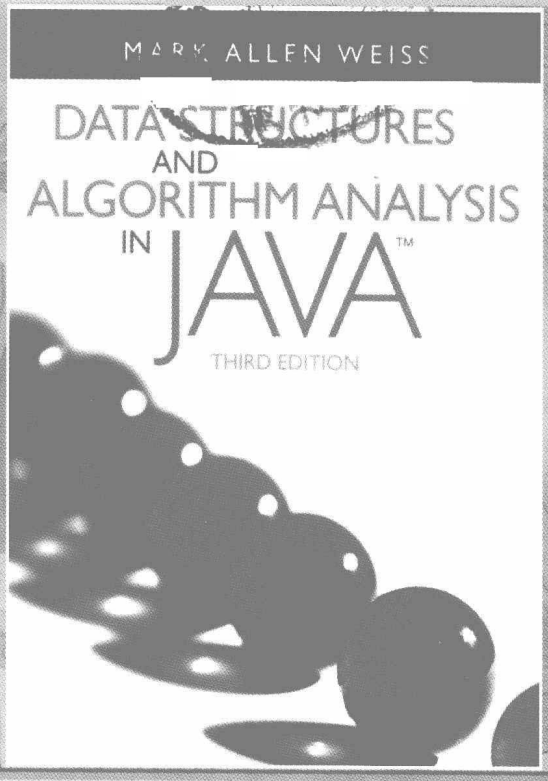
数据结构与算法分析

Java语言描述

[美] 马克·艾伦·维斯（Mark Allen Weiss） 著 冯舜玺 陈越 译
佛罗里达国际大学

Data Structures and Algorithm Analysis in Java

Third Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

数据结构与算法分析：Java 语言描述（原书第 3 版）/（美）维斯（Weiss，M. A.）著；冯舜玺，陈越译．—北京：机械工业出版社，2016.2
（计算机科学丛书）
书名原文：Data Structures and Algorithm Analysis in Java，Third Edition

ISBN 978-7-111-52839-5

I. 数… II. ①维… ②冯… ③陈… 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 021222 号

本书版权登记号：图字：01-2012-2646

本书是国外经典教材，使用卓越的 Java 编程语言作为实现工具，讨论数据结构（组织大量数据的方法）和算法分析（对算法运行时间的估计）。第 3 版的主要新增内容包括 AVL 树删除算法、布谷鸟散列、跳房子散列、基数排序、后缀树和后缀数组等，并对全书代码进行了更新。

本书要求读者具备一定的编程基础，适合作为计算机相关专业高年级本科生和研究生教材，也可供广大程序员参考。

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：曲　熠　　责任校对：董纪丽
印　　刷：中国电影出版社印刷厂　　版　　次：2016 年 3 月第 1 版第 1 次印刷
开　　本：185mm×260mm　1/16　　印　　张：26
书　　号：ISBN 978-7-111-52839-5　　定　　价：69.00 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

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www. hzbook. com
电子邮件：hzjsj@ hzbook. com
联系电话：(010)88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章科技图书出版中心

前　言

Data Structures and Algorithm Analysis in Java, Third Edition

本书目标

本书新的 Java 版论述数据结构——组织大量数据的方法，以及算法分析——算法运行时间的估计。随着计算机的速度越来越快，对于能够处理大量输入数据的程序的需求变得日益迫切。可是，由于在输入量很大的时候程序的低效率变得非常明显，因此这又要求对效率问题给予更仔细的关注。通过在实际编程之前对算法的分析，我们可以确定某个特定的解法是否可行。例如，查阅本书中一些特定的问题，可以看到我们如何通过巧妙的实现，将其处理大量数据的时间限制从几个世纪减至不到 1 秒。因此，我们在提出所有算法和数据结构时都会阐释其运行时间。在某些情况下，对于影响实现的运行时间的一些微小细节都需要认真探究。

一旦确定了解法，接着就要编写程序。随着计算机功能的日益强大，它们必须解决的问题也变得更加庞大和复杂，这就要求我们开发更加复杂的程序。本书的目的是同时教授学生良好的程序设计技巧和算法分析能力，使得他们能够以最高的效率开发出这种程序。

本书适用于高级数据结构(CS7)课程或是第一年研究生的算法分析课程。学生应该掌握一些中级编程知识，包括基于对象的程序设计和递归等内容，并具备一些离散数学的背景。

第 3 版中最显著的变化

第 3 版订正了大量的错误，也修改了很多地方，以使内容更加清晰。此外还有以下修订：

- 第 4 章包括了 AVL 树的删除算法——这也是读者经常需要的内容。
- 第 5 章进行了大量修改和扩充，现在包含两种新算法：布谷鸟散列(cuckoo hashing)和跳房子散列(hopscotch hashing)。此外还增加了一节讨论通用散列法。
- 第 7 章现在包含了基数排序的内容，并且增加了一节讨论下界的证明。
- 第 8 章用到 Seidel 和 Sharir 提出的新的并查集分析，并且证明了 $O(M\alpha(M, N))$界，而不是前一版中比较弱的 $O(M\log^* N)$界。
- 第 12 章增加了后缀树和后缀数组的内容，包括 Karkkainen 和 Sanders 提出的构造后缀数组的线性时间算法(附带实现)。关于确定性跳跃表和 AA 树的章节被删除。
- 通篇代码已做更新，使用了 Java 7 的菱形运算符。

处理方法

虽然本书的内容大部分都与语言无关，但是，程序设计还是需要使用某种特定的语言。正如书名所示，我们为本书选择了 Java。

人们常常将 Java 和 C ++ 比较。Java 具有许多优点，程序员常常把 Java 看成是一种比 C ++ 更安全、更具有可移植性并且更容易使用的语言。因此，这使得它成为讨论和实现基础数据结构的一种优秀的核心语言。Java 的其他重要的方面，诸如线程和 GUI(图形用户界面)，虽然很重要，但是本书并不需要，因此也就不再讨论。

完整的 Java 和 C ++ 版数据结构均在互联网上提供。我们采用相似的编码约定以使得这两种语言之间的对等性更加明显。

内容概述

第1章包含离散数学和递归的一些复习材料。我相信熟练掌握递归的唯一办法是反复不断地研读一些好的用法。因此，除第5章外，递归遍及本书每一章的例子之中。第1章还介绍了一些相关内容，作为对Java中“继承”的复习，包括对Java泛型的讨论。

第2章讨论算法分析，阐述渐近分析及其主要缺点，提供了许多例子，包括对对数级运行时间的深入分析。我们通过直观地把递归程序转变成迭代程序，对一些简单递归程序进行了分析。更复杂的分治程序也在此介绍，不过有些分析(求解递推关系)要推迟到第7章再进行详细讨论。

第3章介绍表、栈和队列。包括对Collections API `ArrayList`类和`LinkedList`类的讨论，提供了Collections API `ArrayList`类和`LinkedList`类的一个重要子集的若干实现。

第4章讨论树，重点是查找树，包括外部查找树(B-树)。UNIX文件系统和表达式树是作为例子来介绍的。这一章还介绍了AVL树和伸展树。查找树实现细节的更仔细的处理可在第12章找到。树的另外一些内容(如文件压缩和博弈树)推迟到第10章讨论。外部介质上的数据结构作为若干章中的最后论题来考虑。对于Collections API `TreeSet`类和`TreeMap`类的讨论，则通过一个重要的例子来展示三种单独的映射在求解同一个问题中的使用。

第5章讨论散列表，既包括经典算法，如分离链接法和线性及平方探测法，同时也包括几个新算法，如布谷鸟散列和跳房子散列。本章还讨论了通用散列法，并且在章末讨论了可扩散列。

第6章是关于优先队列的。二叉堆也在这里讲授，还有些附加的材料论述优先队列某些理论上有趣的实现方法。斐波那契堆在第11章讨论，配对堆在第12章讨论。

第7章论述排序。这一章特别关注编程细节和分析。所有重要的通用排序算法均在该章进行了讨论和比较。此外，还对四种排序算法做了详细的分析，它们是插入排序、希尔排序、堆排序以及快速排序。这一版新增的是基数排序以及对选择类问题的下界的证明。本章末尾讨论了外部排序。

第8章讨论不相交集算法并证明其运行时间。分析部分是新的。这是简短且特殊的一章，如果不讨论Kruskal算法则可跳过该章。

第9章讲授图论算法。图论算法之所以有趣，不仅因为它们在实践中经常出现，而且还因为它们的运行时间强烈地依赖于数据结构的恰当使用。实际上，所有标准算法都和适用的数据结构、伪代码以及运行时间的分析一起介绍。为了恰当地理解这些问题，我们对复杂性理论(包括NP-完全性和不可判定性)进行了简短的讨论。

第10章通过考察一般性的问题求解技术来介绍算法设计。本章通过大量的例子来增强理解。这一章及后面各章使用的伪代码使得读者在理解例子时不会被实现的细节所困扰。

第11章处理摊还分析，主要分析三种数据结构，它们分别在第4章、第6章以及本章(斐波那契堆)介绍。

第12章讨论查找树算法、后缀树和数组、k-d树和配对堆。不同于其他各章，本章给出了查找树和配对堆完整且仔细的实现。材料的安排使得教师可以把一些内容纳入其他各章的讨论之中。例如，第12章中的自顶向下红黑树可以和(第4章的)AVL树一起讨论。

第1~9章为大多数一学期的数据结构课程提供了足够的材料。如果时间允许，那么第10章也可以包括进来。研究生的算法分析课程可以使用第7~11章的内容。第11章所分析的高级数据结构可以很容易地被前面各章所提及。第9章里所讨论的NP-完全性太过简短，不适用于这样的课程。另外再用一部NP-完全性方面的著作作为本教材的补充可能是比较有益的。

练习

每章末尾提供的练习与正文中所述内容的顺序相一致。最后的一些练习是对应整章而不是针对特定的某一节的。难度较大的练习标有一个星号，更具挑战的练习标有两个星号。

参考文献

参考文献列于每章的最后。通常，这些参考文献或者是具有历史意义的、给出书中材料的原始出处，或者阐述对书中给出的结果的扩展和改进。有些文献为一些练习提供了解法。

补充材料

下面的补充材料在 www. pearsonhighered. com/cssupport 对所有读者公开：

- 例子程序的源代码

此外，下述材料仅提供给经培生教师资源中心(Pearson's Instructor Resource Center，IRC)(www. pearsonhighered. com/irc)认可的教师。有意者请访问 IRC 或联系培生的校园代表以获得访问权限。[⊖]

- 部分练习的解答
- 来自本书的一些附图

致谢

在本书的准备过程中，我得到了许多人的帮助，有些已在本书的其他版本中列出，感谢大家。

一如既往地，培生的专家们的努力使得本书的写作过程更加轻松。我愿在此感谢我的编辑 Michael Hirsch 以及制作编辑 Pat Brown。我还要感谢 Abinaya Rajendran 和她在 Integra Software Services 的同事，感谢他们使最后的散稿成书的出色工作。贤妻 Jill 所做的每一件事情都值得我特别感谢。

最后，我还想感谢发来 E-mail 并指出前面各版中错误和矛盾之处的广大读者。我的网页 www. cis. fiu. edu/ ~ weiss 包含更新后的源代码(用 Java 和 C ++ 编写)、勘误表以及提交问题报告的链接。

M. A. W.
佛罗里达州迈阿密市

⊖ 关于本书教辅资源，用书教师可向培生教育出版集团北京代表处申请，电话：010-5735 5169/5735 5171，电子邮件：service. cn@ pearson. com。——编辑注

第1章

Data Structures and Algorithm Analysis in Java, Third Edition

引　论

在这一章，我们阐述本书的目的和目标并简要复习离散数学以及程序设计的一些概念。我们将要：

- 看到程序对于合理的大量输入的运行性能与其在适量输入下运行性能的同等重要性。
- 概括为本书其余部分所需要的基本的数学基础。
- 简要复习递归。
- 概括用于本书的 Java 语言的某些重要特点。

1.1　本书讨论的内容

设有一组 N 个数而要确定其中第 k 个最大者，我们称之为**选择问题**(selection problem)。大多数学习过一两门程序设计课程的学生写一个解决这种问题的程序不会有什么困难。“明显的”解决方法是相当多的。

该问题的一种解法就是将这 N 个数读进一个数组中，再通过某种简单的算法，比如冒泡排序法，以递减顺序将数组排序，然后返回位置 k 上的元素。

稍微好一点的算法可以先把前 k 个元素读入数组并(以递减的顺序)对其排序。接着，将剩下的元素再逐个读入。当新元素被读到时，如果它小于数组中的第 k 个元素则忽略之，否则就将其放到数组中正确的位置上，同时将数组中的一个元素挤出数组。当算法终止时，位于第 k 个位置上的元素作为答案返回。

这两种算法编码都很简单，建议读者试一试。此时我们自然要问：哪个算法更好？哪个算法更重要？还是两个算法都足够好？使用三千万个元素的随机文件和 $k = 15\,000\,000$ 进行模拟将发现，两个算法在合理的时间量内均不能结束；每种算法都需要计算机处理若干天才能算完(虽然最后还是给出了正确的答案)。在第 7 章将讨论另一种算法，该算法将在一秒钟左右给出问题的解。因此，虽然我们提出的两个算法都能算出结果，但是它们不能被认为是好的算法，因为对于第三种算法能够在合理的时间内处理的输入数据量而言，这两种算法是完全不切实际的。

第二个问题是解决一个流行的字谜。输入是由一些字母构成的一个二维数组以及一组单词组成。目标是要找出字谜中的单词，这些单词可能是水平、垂直或沿对角线上任何方向放置的。作为例子，图 1-1 所示的字谜由单词 this、two、fat 和 that 组成。单词 this 从第一行第一列的位置即(1，1)处开始并延伸至(1，4)；单词 two 从(1，1)到(3，1)；fat 从(4，1)到(2，3)；而 that 则从(4，4)到(1，1)。

	1	2	3	4
1	t	h	i	s
2	w	a	t	s
3	o	a	h	g
4	f	g	d	t

图 1-1　字谜示例

现在至少也有两种直观的算法来求解这个问题。对单词表中的每个单词，我们检查每一个有序三元组(行、列、方向)验证是否有单词存在。这需要大量嵌套的 `for` 循环，但它基本上是直观的算法。

也可以这样，对于每一个尚未越出谜板边缘的有序四元组(行、列、方向、字符数)我们可以测试是否所指的单词在单词表中。这也导致使用大量嵌套的 `for` 循环。如果在任意单词中的最大字符数已知，那么该算法有可能节省一些时间。

上述两种方法相对来说都不难编码并可求解通常发表于杂志上的许多现实的字谜游戏。这些字谜通常有16行16列以及40个左右的单词。然而，假设我们把字谜变成为只给字谜板(puzzle board)而单词表基本上是一本英语词典，则上面提出的两种解法均需要相当长的时间来解决这个问题，从而这两种方法都是不可接受的。不过，这样的问题还是有可能在数秒内解决的，甚至单词表可以很大。

在许多问题当中，一个重要的观念是：写出一个工作程序并不够。如果这个程序在巨大的数据集上运行，那么运行时间就变成了重要的问题。我们将在本书看到对于大量的输入如何估计程序的运行时间，尤其是如何在尚未具体编码的情况下比较两个程序的运行时间。我们还将看到彻底改进程序速度以及确定程序瓶颈的方法。这些方法将使我们能够发现需要我们集中精力努力优化的那些代码段。

1.2 数学知识复习

2 本节列出一些需要记忆或是能够推导出的基本公式，并从推导过程复习基本的证明方法。

1.2.1 指数

$$X^A X^B = X^{A+B}$$

$$\frac{X^A}{X^B} = X^{A-B}$$

$$(X^A)^B = X^{AB}$$

$$X^N + X^N = 2X^N \neq X^{2N}$$

$$2^N + 2^N = 2^{N+1}$$

1.2.2 对数

在计算机科学中，除非有特别的声明，否则所有的对数都是以2为底的。

定义1.1 $X^A = B$ 当且仅当 $\log_X B = A$。

由该定义可以得到几个方便的等式。

定理1.1

$$\log_A B = \frac{\log_C B}{\log_C A};\ A,\ B,\ C > 0,\ A \neq 1$$

证明：

令 $X = \log_C B$，$Y = \log_C A$，以及 $Z = \log_A B$。此时由对数的定义，$C^X = B$，$C^Y = A$ 以及 $A^Z = B$，联合这三个等式则产生 $(C^Y)^Z = C^X = B$。因此，$X = YZ$，这意味着 $Z = X/Y$，定理得证。 □

定理1.2

$$\log AB = \log A + \log B;\ A,\ B > 0$$

证明：

令 $X = \log B$，$Y = \log A$，以及 $Z = \log AB$。此时由于假设默认的底为2，$2^X = A$，$2^Y = B$，及 $2^Z = AB$，联合最后的三个等式则有 $2^X 2^Y = 2^Z = AB$。因此 $X + Y = Z$，这就证明了该定理。 □

其他一些有用的公式如下，它们都能够用类似的方法推导。

$$\log A/B = \log A - \log B$$

$$\log(A^B) = B\log A$$

$\log X < X$ 对所有的 $X > 0$ 成立

3 $$\log 1 = 0,\ \log 2 = 1,\ \log 1024 = 10,\ \log 1048576 = 20$$

1.2.3 级数

最容易记忆的公式是

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

和

$$\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$$

在第二个公式中，如果 $0 < A < 1$，则

$$\sum_{i=0}^{N} A^i \leqslant \frac{1}{1 - A}$$

当 N 趋于 ∞ 时该和趋向于 $1/(1-A)$，这些公式是“几何级数”公式。

我们可以用下面的方法推导关于 $\sum_{i=0}^{\infty} A^i (0 < A < 1)$ 的公式。令 S 是其和。此时

$$S = 1 + A + A^2 + A^3 + A^4 + A^5 + \cdots$$

于是

$$AS = A + A^2 + A^3 + A^4 + A^5 + \cdots$$

如果我们将这两个方程相减(这种运算只允许对收敛级数进行)，等号右边所有的项相消，只留下 1：

$$S - AS = 1$$

即

$$S = \frac{1}{1 - A}$$

可以用相同的方法计算 $\sum_{i=1}^{\infty} i/2^i$，它是一个经常出现的和。我们写成

$$S = \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \frac{5}{2^5} + \cdots$$

用 2 乘之得到

$$2S = 1 + \frac{2}{2} + \frac{3}{2^2} + \frac{4}{2^3} + \frac{5}{2^4} + \frac{6}{2^5} + \cdots$$

4

将这两个方程相减得到

$$S = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \cdots$$

因此，$S = 2$。

分析中另一种常用类型的级数是算术级数。任何这样的级数都可以从基本公式计算其值。

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

例如，为求出和 $2 + 5 + 8 + \cdots + (3k - 1)$，将其改写为 $3(1 + 2 + 3 + \cdots + k) - (1 + 1 + 1 + \cdots + 1)$，显然，它就是 $3k(k+1)/2 - k$。另一种记忆的方法则是将第一项与最后一项相加(和为 $3k + 1$)，第二项与倒数第二项相加(和也是 $3k + 1$)，等等。由于有 $k/2$ 个这样的数对，因此总和就是 $k(3k+1)/2$，这与前面的答案相同。

现在介绍下面两个公式，不过它们就没有那么常见了。

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

$$\sum_{i=1}^{N} i^k \approx \frac{N^{k+1}}{|k+1|} \quad k \neq -1$$

当 $k = -1$ 时，后一个公式不成立。此时我们需要下面的公式，这个公式在计算机科学中的使用要远比在数学其他科目中使用得多。数 H_N 叫作调和数，其和叫作调和和。下面近似式中的误差趋向于 $\gamma \approx 0.57\,721\,566$，称为**欧拉常数**(Euler's constant)。

$$H_N = \sum_{i=1}^{N} \frac{1}{i} \approx \log_e N$$

以下两个公式只不过是一般的代数运算：

$$\sum_{i=1}^{N} f(N) = Nf(N)$$

$$\sum_{i=n_0}^{N} f(i) = \sum_{i=1}^{N} f(i) - \sum_{i=1}^{n_0-1} f(i)$$

1.2.4 模运算

如果 N 整除 $A-B$，那么就说 A 与 B 模 N 同余，记为 $A \equiv B(\bmod N)$。直观地看，这意味着无论是 A 还是 B 被 N 去除，所得余数都是相同的。于是，$81 \equiv 61 \equiv 1(\bmod 10)$。如同等号的情形
5 一样，若 $A \equiv B(\bmod N)$，则 $A+C \equiv B+C(\bmod N)$ 以及 $AD \equiv BD(\bmod N)$。

有许多定理适用模运算，其中有些特别需要用到数论来证明。我们将尽量少使用模运算，这样，前面的一些定理也就足够了。

1.2.5 证明的方法

证明数据结构分析中的结论的两种最常用的方法是归纳法证明和反证法证明（偶尔也被迫用到只有教授们才使用的证明）。证明一个定理不成立的最好的方法是举出一个反例。

归纳法证明

由归纳法进行的证明有两个标准的部分。第一步是证明**基准情形**（base case），就是确定定理对于某个（某些）小的（通常是退化的）值的正确性；这一步几乎总是很简单的。接着，进行**归纳假设**（inductive hypothesis）。一般说来，它指的是假设定理对直到某个有限数 k 的所有的情况都是成立的。然后使用这个假设证明定理对下一个值（通常是 $k+1$）也是成立的。至此定理得证（在 k 是有限的情形下）。

作为一个例子，我们证明斐波那契数，$F_0=1$，$F_1=1$，$F_2=2$，$F_3=3$，$F_4=5$，…，$F_i=F_{i-1}+F_{i-2}$，满足对 $i \geqslant 1$，有 $F_i < (5/3)^i$（有些定义规定 $F_0=0$，这只不过将该级数做了一次平移）。为了证明这个不等式，我们首先验证定理对简单的情形成立。容易验证 $F_1=1<5/3$ 及 $F_2=2<25/9$，这就证明了基准情形。假设定理对于 $i=1$，2，…，k 成立，这就是归纳假设。为了证明定理，我们需要证明 $F_{k+1} < (5/3)^{k+1}$。根据定义得到

$$F_{k+1} = F_k + F_{k-1}$$

将归纳假设用于等号右边，得到

$$\begin{aligned} F_{k+1} &< (5/3)^k + (5/3)^{k-1} < (3/5)(5/3)^{k+1} + (3/5)^2(5/3)^{k+1} \\ &= (3/5)(5/3)^{k+1} + (9/25)(5/3)^{k+1} \end{aligned}$$

6

化简后为

$$F_{k+1} < (3/5+9/25)(5/3)^{k+1} = (24/25)(5/3)^{k+1} < (5/3)^{k+1}$$

这就证明了这个定理。

作为第二个例子，我们建立下面的定理。

定理 1.3

如果 $N \geqslant 1$，则 $\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$。

证明：

用数学归纳法证明。对于基准情形，容易看到，当 $N=1$ 时定理成立。对于归纳假设，设定理对 $1 \leqslant k \leqslant N$ 成立。我们将在该假设下证明定理对于 $N+1$ 也是成立的。我们有

$$\sum_{i=1}^{N+1} i^2 = \sum_{i=1}^{N} i^2 + (N+1)^2$$

应用归纳假设得到

$$\begin{aligned} \sum_{i=1}^{N+1} i^2 &= \frac{N(N+1)(2N+1)}{6} + (N+1)^2 = (N+1)\left[\frac{N(2N+1)}{6} + (N+1)\right] \\ &= (N+1)\frac{2N^2+7N+6}{6} = \frac{(N+1)(N+2)(2N+3)}{6} \end{aligned}$$

因此

$$\sum_{i=1}^{N+1} i^2 = \frac{(N+1)[(N+1)+1][2(N+1)+1]}{6}$$

定理得证。□

通过反例证明

公式 $F_k \leqslant k^2$ 不成立。证明这个结论的最容易的方法就是计算 $F_{11} = 144 > 11^2$。

反证法证明

反证法证明是通过假设定理不成立，然后证明该假设导致某个已知的性质不成立，从而原假设是错误的。一个经典的例子是证明存在无穷多个素数。为了证明这个结论，我们假设定理 7
不成立。于是，存在某个最大的素数 P_k。令 P_1，P_2，…，P_k 是依序排列的所有素数并考虑

$$N = P_1 P_2 P_3 \cdots P_k + 1$$

显然，N 是比 P_k 大的数，根据假设 N 不是素数。可是，P_1，P_2，…，P_k 都不能整除 N，因为除得的结果总有余数 1。这就产生一个矛盾，因为每一个整数或者是素数，或者是素数的乘积。因此，P_k 是最大素数的原假设是不成立的，这正意味着定理成立。

1.3　递归简论

我们熟悉的大多数数学函数都是由一个简单公式来描述的。例如，我们可以利用公式

$$C = 5(F - 32)/9$$

将华氏温度转换成摄氏温度。有了这个公式，写一个 Java 方法就太简单了。除去程序中的说明和大括号外，这一行的公式正好翻译成一行 Java 程序。

有时候数学函数以不太标准的形式来定义。例如，我们可以在非负整数集上定义一个函数 f，它满足 $f(0) = 0$ 且 $f(x) = 2f(x-1) + x^2$。从这个定义我们看到 $f(1) = 1$，$f(2) = 6$，$f(3) = 21$，以及 $f(4) = 58$。当一个函数用它自己来定义时就称为是**递归**(recursive)的。Java 允许函数是递归的。⊖

```
1       public static int f( int x )
2       {
3           if( x == 0 )
4               return 0;
5           else
6               return 2 * f( x - 1 ) + x * x;
7       }
```

图 1-2　一个递归方法

但重要的是要记住，Java 提供的仅仅是遵循递归思想的一种尝试。不是所有的数学递归函数都能被有效地(或正确地)由 Java 的递归模拟来实现。上面例子说的是递归函数 f 应该只用几行就能表示出来，正如非递归函数一样。图 1-2 指出了函数 f 的递归实现。

第 3 行和第 4 行处理**基准情况**(base case)，即此时函数的值可以直接算出而不用求助递归。正如 $f(x) = 2f(x-1) + x^2$ 若没有 $f(0) = 0$ 这个事实在数学上没有意义一样，Java 的递归方法若无基准情况也是毫无意义的。第 6 行执行的是递归调用。

关于递归，有几个重要并且可能会被混淆的概念。一个常见的问题是：它是否就是循环推理(circular logic)？答案是：虽然我们定义一个方法用的是这个方法本身，但是我们并没有用方法本身定义该方法的一个特定的实例。换句话说，通过使用 $f(5)$ 来得到 $f(5)$ 的值才是循环的。通过使用 $f(4)$ 得到 $f(5)$ 的值不是循环的，当然，除非 $f(4)$ 的求值又要用到对 $f(5)$ 的计算。两个最重要的问题恐怕就是如何做和为什么做的问题了。如何和为什么的问题将在第 3 章正式解 8
决。这里，我们将给出一个不完全的描述。

实际上，递归调用在处理上与其他调用没有什么不同。如果以参数 4 的值调用函数 f，那么程序的第 6 行要求计算 $2 * f(3) + 4 * 4$。这样，就要执行一个计算 $f(3)$ 的调用，而这又导致计算 $2 * f(2) + 3 * 3$。因此，又要执行另一个计算 $f(2)$ 的调用，而这意味着必须求出 $2 * f(1) +$

⊖ 对于数值计算使用递归通常不是个好主意。我们在解释基本概念时已经说过。

2 * 2 的值。为此，通过计算 2 * $f(0)$ + 1 * 1 而得到 $f(1)$。此时，$f(0)$必须被赋值。由于这属于基准情况，因此我们事先知道 $f(0) = 0$。从而 $f(1)$的计算得以完成，其结果为 1。然后，$f(2)$、$f(3)$以及最后 $f(4)$的值都能够计算出来。跟踪挂起的函数调用(这些调用已经开始但是正等待着递归调用来完成)以及它们的变量的记录工作都是由计算机自动完成的。然而，重要的问题在于，递归调用将反复进行直到基准情形出现。例如，计算 $f(-1)$的值将导致调用 $f(-2)$、$f(-3)$等等。由于这将不可能出现基准情形，因此程序也就不可能算出答案。偶尔还可能发生更加微妙的错误，我们将其展示在图 1-3 中。图 1-3 中程序的这种错误是第 6 行上的 bad(1)定义为 bad(1)。显然，实际上 bad(1)究竟是多少，这个定义给不出任何线索。因此，计算机将会反复调用 bad(1)以期解出它的值。最后，计算机簿记系统将占满内存空间，程序崩溃。一般情形下，我们会说该方法对一个特殊情形无效，而在其他情形是正确的。但此处这么说则不正确，因为 bad(2)调用 bad(1)。因此，bad(2)也不能求出值来。不仅如此，bad(3)、bad(4)和 bad(5)都要调用 bad(2)，
9 bad(2)算不出值，它们的值也就不能求出。事实上，除了 0 之外，这个程序对 n 的任何非负值都无效。对于递归程序，不存在像“特殊情形”这样的情况。

```
1    public static int bad( int n )
2    {
3        if( n == 0 )
4            return 0;
5        else
6            return bad( n / 3 + 1 ) + n - 1;
7    }
```

图 1-3 无终止递归方法

上面的讨论导致递归的前两个基本法则：

1. *基准情形*(base case)。必须总要有某些基准的情形，它们不用递归就能求解。

2. *不断推进*(making progress)。对于那些要递归求解的情形，递归调用必须总能够朝着一个基准情形推进。

在本书中我们将用递归解决一些问题。作为非数学应用的一个例子，考虑一本大词典。词典中的词都是用其他的词定义的。当查一个单词的时候，我们不是总能理解对该词的解释，于是我们不得不再查找解释中的一些词。同样，对这些词中的某些地方我们又不理解，因此还要继续这种查找。因为词典是有限的，所以实际上或者我们最终要查到一处，明白了此处解释中所有的单词(从而理解这里的解释，并按照查找的路径回查其余的解释)或者我们发现这些解释形成一个循环，无法理解其最终含义，或者在解释中需要我们理解的某个单词不在这本词典里。

我们理解这些单词的递归策略如下：如果知道一个单词的含义，那么就算我们成功；否则，就在词典里查找这个单词。如果我们理解对该词解释中的所有的单词，那么又算我们成功；否则，通过递归查找一些我们不认识的单词来“算出”对该单词解释的含义。如果词典编纂得完美无瑕，那么这个过程就能够终止；如果其中一个单词没有查到或是循环定义(解释)，那么这个过程则循环不定。

打印输出整数

设有一个正整数 n 并希望把它打印出来。我们的例程的名字为 printOut(n)。假设仅有的现成 I/O 例程将只处理单个数字并将其输出到终端。我们为这种例程命名为 printDigit；例如，printDigit(4)将输出 4 到终端。

递归将为该问题提供一个非常漂亮的解。要打印 76234，我们首先需要打印出 7623，然后再打印出 4。第二步用语句 printDigit(n% 10)很容易完成，但是第一步却不比原问题简单多少。它实际上是同一个问题，因此可以用语句 printOut(n/10)递归地解决它。

这告诉我们如何去解决一般的问题，不过我们仍然需要确认程序不是循环不定的。由于我们尚未定义一个基准情况，因此很清楚，我们仍然还有些事情要做。如果 $0 \leq n < 10$，那么基准情形就是 printDigit(n)。现在，printOut(n)已对每一个从 0 到 9 的正整数定义，而更大
10 的正整数则用较小的正整数定义。因此，不存在循环的问题。整个方法在图 1-4 中指出。