计算机算法设计与分析基础

JISUANJI SUANFA SHEJI YU FENXI JICHU

JISUANJI

欧中亚　张起荣 / 主编

电子科技大学出版社

计算机算法设计与分析基础

JISUANJI SUANFA SHEJI YU FENXI JICHU

JISUANJI

欧中亚　张起荣 / 主编

电子科技大学出版社

图书在版编目（CIP）数据

计算机算法设计与分析基础 / 欧中亚，张起荣主编.
— 成都：电子科技大学出版社，2017.4
ISBN 978-7-5647-4289-8

Ⅰ. ①计… Ⅱ. ①欧… ②张… Ⅲ. ①计算机算法－研究 Ⅳ. ① TP301.6

中国版本图书馆 CIP 数据核字（2017）第 068017 号

计算机算法设计与分析基础

欧中亚 张起荣 主 编

出 版：电子科技大学出版社（成都市一环路东一段 159 号电子信息产业大厦 邮编：610051）
策划编辑：罗 雅
责任编辑：卢 莉
主 页：www.uestcp.com.cn
电子邮箱：uestcp@uestcp.com.cn
发 行：新华书店经销
印 刷：四川永先数码印刷有限公司
成品尺寸：185mm×260mm 印张 12.5 字数 319 千字
版 次：2017 年 4 月第一版
印 次：2017 年 4 月第一次印刷
书 号：ISBN 978-7-5647-4289-8
定 价：37.00 元

前　言

算法设计与分析是计算机科学永恒不变的主题，是计算机专业的核心课程之一。这门课程旨在培养学生以算法设计和分析为手段，时间高效、空间高效地解决问题的意识和能力，以及简洁、清晰、准确表达算法及其分析过程的技能，提高学生的算法修养，培养创新意识，培育研究能力。通过这门课程的学习，学生可以理解算法和算法复杂度的概念，掌握评判算法优劣的标准，较系统地掌握分治算法、动态规划算法、贪心算法、搜索策略、随机算法、近似算法和在线算法等算法设计技术，掌握循环不变量方法、反例方法、平摊分析方法、概率分析方法、近似度分析方法和竞争度分析方法等算法分析技术，为进一步深造打下良好的算法基础。

本书以算法设计与分析方法为主线，着重强调对算法设计与分析基本技术的掌握，舍弃了在数据结构、集合论与图论等先修课程中学生已经掌握的基本算法，包括绝大部分排序算法、字符串匹配算法和基本图论算法。本书对每一种算法设计与分析技术，均先通过简单的例子介绍技术的原理和要点，再通过一系列的例子阐述使用该技术时需要注意的各个侧面。

本书还特别强调算法设计过程依赖于对问题的分析和对问题特征的把握，即算法设计通常从求解问题的蛮力算法开始，逐步观察和分析问题的特征并结合恰当的算法设计技术，逐渐设计出高效的算法。每章均包含了这样的算法设计实例，以引导读者建立从蛮力法入手，逐步分析问题特征，再利用问题特征设计算法的思维过程，让读者理解问题特征分析、基本算法和算法设计与分析之间的关系，体会从无到有、精益求精的算法设计过程，培养创新意识和研究能力。

为了适应21世纪我国培养计算机各类人才的需要，本书结合我国教育工作的现状，追踪国际计算机科学技术的发展水平，以算法设计策略为知识单元，系统地介绍计算机算法的设计方法与分析技巧，以期为读者提供一个广泛扎实的计算机的算法知识基础。

由于作者水平有限，书中不当之处在所难免，敬请读者批评指正。

编　者

目　录

第1章　绪　论

1.1　算法的基本概念

算法的概念在计算机科学领域几乎无处不在，在各种计算机软件系统的实现中，算法设计往往处于核心地位。例如，操作系统是现代计算机系统中不可缺少的系统软件，操作系统的各个任务都是一个单独的问题，每个问题由操作系统中的一个子程序根据特定的算法来实现。用什么方法来设计算法、如何判定一个算法的优劣、所设计的算法需要占用多少时间资源和空间资源，在实现一个软件系统时，都是必须予以解决的重要问题。

1.1.1　为什么要学习算法

1. 用计算机求解任何问题都离不开程序设计，而程序设计的核心是算法设计

一般来说，对程序设计的研究可以分为4个层次：算法、方法学、语言和工具，其中算法研究位于最高层次。算法对程序设计的指导可以延续几年甚至几十年，它不依赖于方法学、语言和工具的发展与变化。例如，用于数据存储和检索的Hash算法产生于20世纪50年代，用于排序的快速排序算法发明于20世纪60年代，但它们至今仍被人们广为使用，可是程序设计方法已经从结构化发展到面向对象，程序设计语言也变化了几代，至于编程工具，很难维持3年不变。所以，对于从事计算机专业的人士来说，学习算法是非常必要的。

2. 学习算法还能够提高人们分析问题的能力

算法可以看作是解决问题的一类特殊方法——它不是问题的答案，而是经过精确定义的用来获得答案的求解过程。因此，无论是否涉及计算机，特定的算法设计技术都可以看作是问题求解的有效策略。著名的计算机科学家克努特(Donald Knuth)是这样论述这个问题的：“受过良好训练的计算机科学家知道如何处理算法，如何构造算法、操作算法、理解算法以及分析算法，这些知识远不只是为了编写良好的计算机程序而准备的。算法是一种一般性的智能工具，一定有助于我们对其他学科的理解，不管是化学、语言学、音乐还是另外的学科。为什么算法会有这种作用呢？我们可以这样理解：人们常说，一个人只有把知识教给别人，才能真正掌握它。实际上，一个人只有把知识教给计算机，才能真正掌握它，也就是说，将知识表述为一种算法比起简单地按照常规去理解事物，用算法将其形式化会使我们获得更加深刻的理解。”

算法研究的核心问题是时间(速度)问题。人们可能有这样的疑问：既然计算机硬件技术的发展使得计算机的性能不断提高，算法的研究还有必要吗？

计算机的功能越强大,人们就越想去尝试更复杂的问题,而更复杂的问题需要更大的计算量。现代计算技术在计算能力和存储容量上的革命仅仅提供了计算更复杂问题的有效工具,无论硬件性能如何提高,算法研究始终是推动计算机技术发展的关键。下面看几个例子。

(1)检索技术

20 世纪 50—60 年代,检索的对象是规模比较小的数据集合。例如,编译系统中的符号表,表中的记录个数一般在几十至数百这样的数量级。

20 世纪 70—80 年代,数据管理采用数据库技术,数据库的规模在 k 级或 M 级,检索算法的研究在这个时期取得了巨大的进展。

20 世纪 90 年代以来,Internet 引起计算机应用的急速发展,海量数据的处理技术成为研究的热点,而且数据驻留的存储介质、数据的存储方法以及数据的传输技术也发生了许多变化,这些变化使得检索算法的研究更为复杂,也更为重要。

近年来,智能检索技术成为基于 Web 信息检索的研究热点。使用搜索引擎进行 Web 信息检索时,经常看到一些搜索引擎前 50 个搜索结果中几乎有一半来自同一个站点的不同页面,这是检索系统缺乏智能化的一种表现。另外,在传统的 Web 信息检索服务中,信息的传输是按 pull 的模式进行的,即用户找信息。而采用 push 的方式,是信息找用户,用户不必进行任何信息检索,就能方便地获得自己感兴趣的信息,这就是智能信息推送技术。这些新技术的每一项重要进步都与算法研究的突破有关。

(2)压缩与解压缩

随着多媒体技术的发展,计算机的处理对象由原来的字符发展到图像、图形、音频、视频等多媒体数字化信息,这些信息数字化后,其特点就是数据量非常庞大,同时,处理多媒体所需的高速传输速度也是计算机总线所不能承受的。因此,对多媒体数据的存储和传输都要求对数据进行压缩。声音文件的 MP3 压缩技术说明了压缩与解压缩算法研究的巨大成功,一个播放 3 ~ 4min 歌曲的 MP3 文件通常只需 3MB 左右的磁盘空间。

(3)信息安全与数据加密

在计算机应用迅猛发展的同时,也面临着各种各样的威胁。一位酒店经理曾经描述了这样一种可能性:“如果我能破坏网络的安全性,想想你在网络上预订酒店房间所提供的信息吧! 我可以得到你的名字、地址、电话号码和信用卡号码,我知道你现在的位置,将要去哪儿,何时去,我也知道你支付了多少钱,我已经得到足够的信息来盗用你的信用卡!”这的确是一个可怕的情景。所以,在电子商务中,信息安全是最关键的问题,保证信息安全的一个方法就是对需要保密的数据进行加密。在这个领域,数据加密算法的研究是绝对必需的,其必要性与计算机性能的提高无关。

1.1.2 算法及其重要特性

算法(Algorithm)被公认为是计算机科学的基石。通俗地讲,算法是解决问题的方法。严格地说,算法是对特定问题求解步骤的一种描述,是指令的有限序列,此外,算法还必须满足下列 5 个重要特性:输入、输出、有穷性、确定性、可行性。

1.1.3　算法的描述方法

算法设计者在构思和设计了一个算法之后，必须清楚准确地将所设计的求解步骤记录下来，即描述算法。常用的描述算法的方法有自然语言、流程图、程序设计语言和伪代码等。

1.1.4　算法设计的一般过程

算法是问题的解决方案，这个解决方案本身并不是问题的答案，而是能获得答案的指令序列。不言而喻，由于实际问题千奇百怪，问题求解的方法千变万化，所以，算法的设计过程是一个灵活的充满智慧的过程，它要求设计人员根据实际情况，具体问题具体分析。可以肯定的是，发明（或发现）算法是一个非常有创造性和值得付出精力的过程。在设计算法时，遵循下列步骤可以在一定程度上指导算法的设计。

1. 理解问题

在面对一个算法任务时，算法设计者往往不能准确地理解要求他做的是什么，对算法希望实现什么只有一个大致的想法就匆忙地落笔写算法，其后果往往是写出的算法漏洞百出。在设计算法时需要做的第一件事情就是完全理解要解决的问题，仔细阅读问题的描述，手工处理一些小例子。对设计算法来说，这是一项重要的技能：准确地理解算法的输入是什么？要求算法做的是什么？即明确算法的入口和出口，这是设计算法的切入点。

2. 预测所有可能的输入

算法的输入确定了该算法所解问题的一个实例。一般而言，对于问题 P，总有其相应的实例集 I，则算法 A 若是问题 P 的算法，意味着把 P 的任意实例 input $\in$ I 作为算法 A 的输入，都能得到问题 P 的正确输出。预测算法所有可能的输入，包括合法的输入和非法的输入。事实上，无法保证一个算法（或程序）永远不会遇到一个错误的输入，一个对大部分输入都运行正确而只有一个输入不行的算法，就像一颗等待爆炸的炸弹。这绝不是危言耸听，有大量这种引起灾难性后果的案例。倒如，许多年以前，整个 AT&T 的长途电话网崩溃，造成了几十亿美元的直接损失。原因只是一段程序的设计者认为他的代码能一直传送正确的参数值，可是有一天，一个不应该有的值作为参数传递了，导致了整个北美电话系统的崩溃。

如果养成习惯，首先考虑问题和它的数据，然后列举出算法必须处理的所有特殊情况，那么可以更快速成功地构造算法。

3. 在精确解和近似解间做选择

计算机科学的研究目标是用计算机来求解人类所面临的各种问题。但是，有些问题无法求得精确解，例如计算 π 值、求平方根、解非线性方程、求定积分等；有些问题由于其固有的复杂性，求精确解需要花费太长的时间，其中最著名的要算旅行商问题（即 TSP 问题），此时，只能求出近似解。有时需要根据问题以及问题所受的资源限制，在精确解和近似解间做选择。

4. 确定适当的数据结构

确定数据结构通常包括对问题实伪的数据进行组织和重构，以及为完成算法所设计的辅助数据结构。

5. 算法设计技术

现在,设计算法的必要条件都已经具备了,如何设计一个算法来解决一个特定的问题呢? 这正是本书讨论的主题。算法设计技术(Algorithm Design Technique,也称算法设计策略)是设计算法的一般性方法,可用于解决不同计算领域的多种问题。本书讨论的算法设计技术已经被证明是对算法设计非常有用的通用技术,包括分治法、动态规划法、贪心法、回溯法、分支限界法、概率算法、近似算法等。这些算法设计技术构成了一组强有力的工具,在为新问题(即没有令人满意的已知算法可以解决的问题)设计算法时,可以运用这些技术设计出新的算法。算法设计技术作为问题求解的一般性策略,在解决计算机领域以外的问题时,也能发挥相当大的作用,读者在日后的学习和工作中将会发现学习算法设计技术的好处。

6. 描述算法

在构思和设计了一个算法之后,必须清楚准确地将所设计的求解步骤记录下来,即描述算法。

7. 跟踪算法

逻辑错误无法由计算机检测出来,因为计算机只会执行程序,而不会理解动机。经验和研究都表明,发现算法(或程序)中的逻辑错误的重要方法就是系统地跟踪算法。跟踪必须要用"心和手"来进行,跟踪者要像计算机一样,用一组输入值来执行该算法,并且这组输入值要最大可能地暴露算法中的错误。即使有几十年经验的高级软件工程师,也经常利用此方法查找算法中的逻辑错误。

8. 分析算法的效率

9. 根据算法编写代码

算法设计的一般过程如图 1 - 1 所示。需要强调的是,一个好算法是反复努力和重新修正的结果,所以,即使足够幸运地得到了一个貌似完美的算法,也应该尝试着改进它。

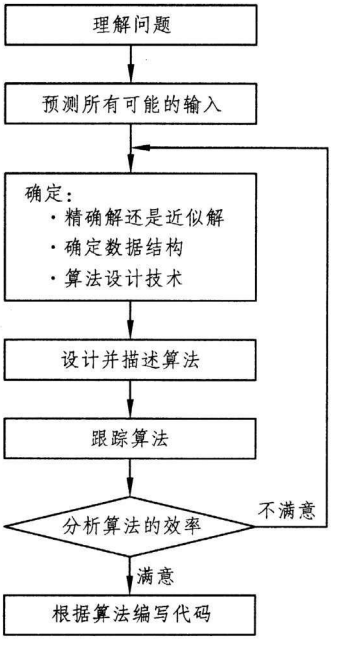


图 1 - 1　算法设计的一般过程

1.2　算法分析

算法分析(Algorithm Analysis)指的是对算法所需要的两种计算机资源——时间和空间——进行估算,所需要的资源越多,该算法的复杂性就越高。不言而喻,对于任何给定的问题,设计出复杂性尽可能低的算法是设计算法时追求的一个重要目标;另一方面,当给定的问题有多种解法时,选择其中复杂性最低者,是选用算法时遵循的一个重要准则。随着计算机硬件性能的提高,一般情况下,算法所需要的额外空间已不是我们需要关注的重点了,但是对算法时间效率的要求仍然是计算机科学不变的主题。本书重点讨论算法时间复杂性(Time Complexity)的分析,对空间复杂性(Space Complexity)的分析是类似的。

1.2.1　渐近符号

算法的复杂性是运行算法所需要的计算机资源的量,这个量应该集中反映算法的效率,而从运行该算法的实际计算机中抽取出来。撇开与计算机软、硬件有关的因素,影响算法时间代价的最主要因素是问题规模。问题规模(Problem scope)是指输入量的多少,一般来说,它可以从问题描述中得到。例如,对一个具有 n 个整数的数组进行排序,问题规模是 n。一个显而易见的事实是:几乎所有的算法,对于规模更大的输入都需要运行更长的时间。例如,需要更多时间来对更大的数组排序。所以运行算法所需要的时间 T 是问题规模 n 的函数,记作 $T(n)$。

要精确地表示算法的运行时间函数常常是很困难的,即使能够给出,也可能是个相当复杂的函数,函数的求解本身也是相当复杂的。考虑到算法分析的主要目的在于比较求解同一个问题的不同算法的效率,为了客观地反映一个算法的运行时间,可以用算法中基本语句的执行次数来度量算法的工作量。基本语句(Basic statement)是执行次数与整个算法的执行时间成正比的语句,基本语句对算法运行时间的贡献最大,是算法中最重要的操作。这种衡量效率的方法得出的不是时间量,而是一种增长趋势的度量。换言之,只考察当问题规模充分大时,算法中基本语句的执行次数在渐近意义下的阶,通常使用大 O、大 Ω(Omega,大写 Ω,小写 ω)和 Θ(Theta,大写 Θ,小写 θ)3 种渐近符号表示。

1. 大 O 符号

定义 1-1　若存在两个正的常数 c 和 n_0,对于任意 $n \geqslant n_0$,都有 $T(n) \leqslant cf(n)$,则称 $T(n) = O(f(n))$[或称算法在 $O(f(n))$ 中]。

大 O 符号用来描述增长率的上限,表示 $T(n)$ 的增长最多像 $f(n)$ 增长得那样快,也就是说,当输入规模为 n 时,算法消耗时间的最大值,就是上限的阶越低,结果就越有价值。大 O 符号的含义如图 1-2 所示,为了说明这个定义,将问题规模 n 扩展为实数。

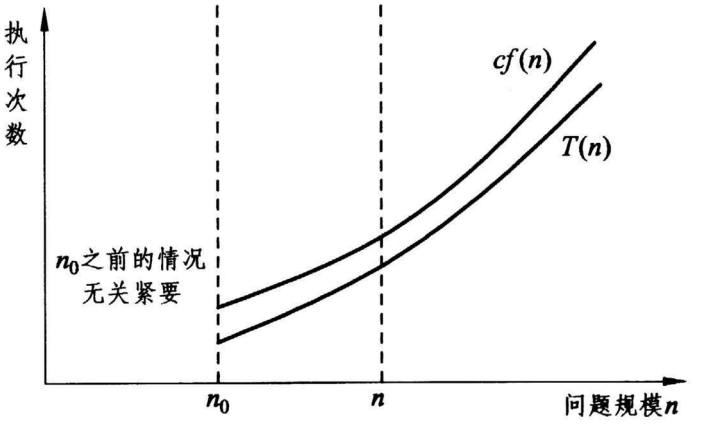


图 1-2　大 O 符号的含义

应该注意的是,定义 1-2 给了很大的自由度来选择常量 c 和 n_0 特定值,例如,下列推导都是合理的

$100n+5 \leqslant 100n+n(\text{当 } n \geqslant 5)=101n=O(n)(c=101, n_0=5)$

$100n+5 \leqslant 100n+5n(\text{当 } n \geqslant 1)=105n=O(n)(c=105, n_0=1)$

2. 大 Ω 符号

定义 1-2　若存在两个正的常数 c 和 n_0,对于任意 $n \geqslant n_0$,都有 $T(n) \geqslant cg(n)$,则称 $T(n)=\Omega(g(n))$ 中)。

大 Ω 符号用来描述增长率的下限,也就是说,当输入规模为 n 时,算法消耗时间的最小值。与大 O 符号对称,这个下限的阶越高,结果就越有价值。

大 Ω 符号的含义如图 1-3 所示。

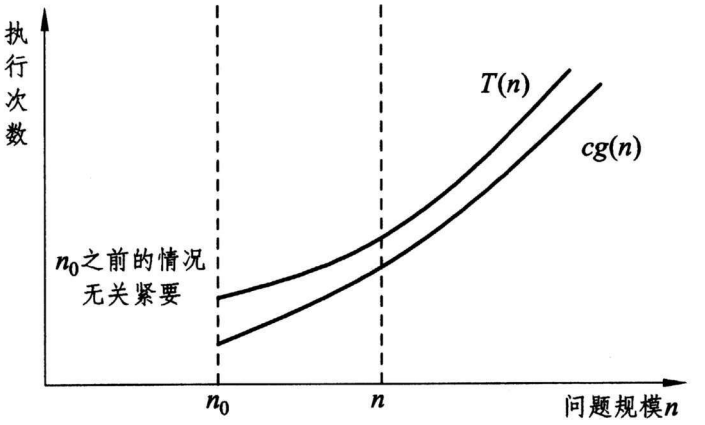


图 1-3　大 Ω 符号的含义

大 Ω 符号常用来分析某个问题或某类算法的时间下界。例如,矩阵乘法问题的时间下界为 $\Omega(n^2)$(平凡下界),是指任何两个 $n \times n$ 矩阵相乘的算法的时间复杂性不会小于即 n^2,基于比较的排序算法的时间下界为 Ω(即 $n\log_2 n$),是指无法设计出基于比较的排序算法,其时间复杂性小于 $n\log_2 n$。

大 Ω 符号常常与大 O 符号配合以证明某问题的一个特定算法是该问题的最优算法,或是该问题中的某算法类中的最优算法。

3. Θ 符号

定义 1 - 3　若存在 3 个正的常数 c_1、c_2 和 n_0，对于任意 $n \geqslant n_0$，都有 $c_1 f(n) \geqslant T(n) \geqslant c_2 f(n)$，则称 $T(n) = \Theta(f(n))$。

Θ 符号意味着 $T(n)$ 与 $f(n)$ 同阶，用来表示算法的精确阶。Θ 符号的含义如图1 - 4所示。

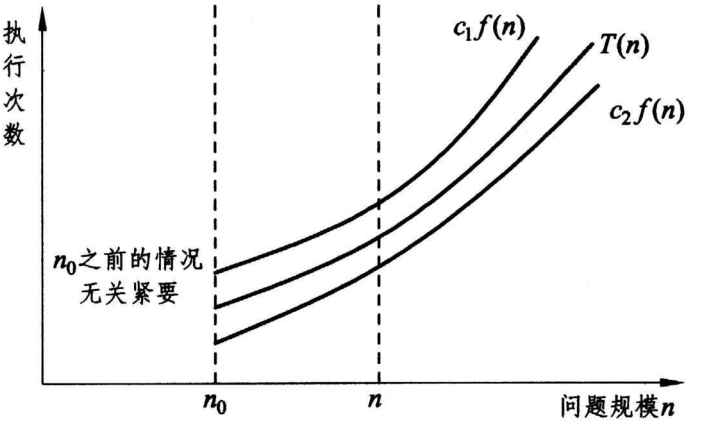


图 1 - 4　Θ 符号的意义

下面举例说明大 O、大 Ω 和 Θ 三种渐近符号的使用。

例 1　$T(n) = 3n - 1$

解答：

当 $n \geqslant 1$ 时，$3n - 1 \leqslant 3n = O(n)$（放大）

当 $n \geqslant 1$ 时，$3n - 1 \geqslant 3n - n = 2n = \Omega(n)$（缩小）

当 $n \geqslant 1$ 时，$3n \geqslant 3n - 1 = 2n + n - 1 \geqslant 2n$，则 $3n - 1 = \Theta(n)$

例 2　$T(n) = 5n^2 + 8n + 1$

解答：

当 $n \geqslant 1$ 时，$5n^2 + 8n + 1 \leqslant 5n^2 + 8n + n = 5n^2 + 9n \leqslant 5n^2 + 9n^2 \leqslant 14n^2 = O(n^2)$

当 $n \geqslant 1$ 时，$5n^2 + 8n + 1 \geqslant 5n^2 = \Omega(n^2)$

当 $n \geqslant 1$ 时，$14n^2 \geqslant 5n^2 + 8n + 1 \geqslant 5n^2$，则 $5n^2 + 8n + 1 = \Theta(n^2)$

定理 1 - 1　若 $T(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0 (a_m > 0)$，则有 $T(n) + O(n^m)$，且 $T(n) = \Omega(n^m)$，因此，有 $T(n) = \Theta(n^m)$。

1.2.2　非递归算法的分析

从算法是否递归调用的角度来说，可以将算法分为非递归算法和递归算法。对非递归算法时间复杂性的分析，关键是建立一个代表算法运行时间的求和表达式，然后用渐近符号表示这个求和表达式。

例 3　在一个整型数组中查找最小值元素，具体算法如下：

```
int ArrayMin(int a[],int n)
{
int min = a[0];
```

```
for(int i=1;i<n;i++)
{
if(a[i]<min)
min=a[i];
}
return min;
}
```

在例3中,问题规模显然是数组中的元素个数,执行最频繁的操作是在for循环中,循环体中包含两条语句:比较和赋值,应该把哪一个作为基本语句呢?由于每做一次循环都会进行一次比较,而赋值语句却不一定执行,所以,应该把比较运算作为该算法的基本语句。接下来考虑基本语句的执行次数,由于每执行一次循环就会做一次比较,而循环变量 i 从1到 $n-1$ 之间的每个值都会做一次循环,可得到如下求和表达式:

$$T(n)=\sum_{i=1}^{n-1}1$$

用渐近符号表示这个求和表达式:

$$T(n)=\sum_{i=1}^{n-1}1=n-1=O(n)$$

非递归算法分析的一般步骤如下。

(1)决定用哪个(或哪些)参数作为算法问题规模的度量。在大多数情况下,问题规模是很容易确定的,可以从问题的描述中得到。

(2)找出算法中的基本语句。算法中执行次数最多的语句就是基本语句,通常是最内层循环的循环体。

(3)检查基本语句的执行次数是否只依赖于问题规模。如果基本语句的执行次数还依赖于其他一些特性(如数据的初始分布),则最好情况、最坏情况和平均情况的效率需要分别研究。

(4)建立基本语句执行次数的求和表达式。计算基本语句执行的次数,建立一个代表算法运行时间的求和表达式。

(5)用渐近符号表示这个求和表达式。计算基本语句执行次数的数量级,用大 O 符号来描述算法增长率的上限。

1.2.3 递归算法的分析

递归算法实际上是一种分而治之的方法,它把复杂问题分解为若干个简单问题来求解。对递归算法时间复杂性的分析,关键是根据递归过程建立递推关系式,然后求解这个递推关系式。下面介绍几种求解递推关系式的技术。

1. 猜测技术

猜测技术首先对递推关系式估计一个上限,然后试着证明它正确。如果给出了一个正确的上限估计,经过归纳证明就可以验证事实。如果证明成功,那么就试着收缩上限。如果证明失败,那么就放松限制再试着证明,一旦上限符合要求就可以结束了。当只求解算法的

近似复杂性时这是一种很有用的技术。

例 4　使用猜测技术分析二路归并排序算法的时间复杂性。

二路归并排序是将一个长度为 n 的记录序列分成两部分，分别对每一部分完成归并排序，最后把两个子序列合并到一起。其运行时间用下面的递推式描述：

$$T(n) = \begin{cases} 1 & n = 2 \\ 2T(n/2) + n & n > 2 \end{cases}$$

也就是说，在序列长度为 n 的情况下，算法的代价是序列长度为 $n/2$ 时代价的 2 倍（对归并排序的递归调用）加上 n（把两个子序列合并在一起）。

假定 $T(n) \leqslant n^2$，并证明这个猜测是正确的。在证明中，为了计算方便，假定$n = 2^k$。

对于最基本的情况，$T(2) = 1 \leqslant 2^2$；对于所有 $i \leqslant n$，假设 $T(i) \leqslant i^2$，而

$$T(2n) = 2T(n) + 2n \leqslant 2n^2 + 2n \leqslant 4n^2 = (2n)^2$$

由此，$T(n) = O(n^2)$ 成立。

$O(n^2)$ 是一个最小上限吗？如果猜测更小一些，例如对于某个常数 c，$T(n) \leqslant cn$，很明显，这样做不行。所以，真正的代价一定在 n 和 n^2 之间。

现在试一试 $T(n) \leqslant n\log_2 n$。

对于最基本的情况，$T(2) = 1 \leqslant (2\log_2 2) = 2$；对于所有 $i \leqslant n$，假设及 $T(i) \leqslant i\log_2 i$，而

$$T(2n) \leqslant 2T(n) + 2(n) \leqslant 2n\log_2 n = 2n(\log_2 n + 1) \leqslant 2n\log_2(2n)$$

这就是我们要证明的，$T(n) = O(n\log_2 n)$

2. 扩展递归技术

扩展递归技术是将递推关系式中等式右边的项根据递推式进行替换，这称为扩展。扩展后的项被再次扩展，以此下去，会得到一个求和表达式，然后就可以借助于求和技术了。

例 5　使用扩展递归技术分析下面递推式的时间复杂性。

$$T(n)\begin{cases} 7 & n = 1 \\ 2T(n/2) + 5n^2 & n > 1 \end{cases}$$

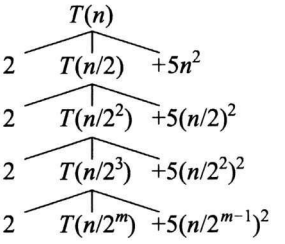
为了简单起见，假定 $n = 2^k$。将递推式像下面这样扩展：

$$\begin{aligned} T(n) &= 2T(n/2) + 5n^2 \\ &= 2(2T(n/4) + 5(n/2)^2) + 5n^2 \\ &= 2(2(2T(n/8) + 5(n/4)^2) + 5(n/2)^2) + 5n^2 \\ &\cdots \\ &= 2^k T(1) + 2^{k-1}5\left(\frac{n}{2^{k-1}}\right)^2 + \cdots + 2 \times 5\left(\frac{n}{2}\right)^2 + 5n^2 \end{aligned}$$

最后这个表达式可以使用如下的求和表示：

$$\begin{aligned} T(n) &= 7n + 5\sum_{i=0}^{k-1}\left(\frac{n^2}{2^i}\right) = 7n + 5n^2\left(2 - \frac{1}{2^{k-1}}\right) \\ &= 7n + 5n^2\left(2 - \frac{2}{n}\right) = 10n^2 - 3n \leqslant 10n^2 = O(n^2) \end{aligned}$$

其扩展过程也可以直观地描述为



3. 通用分治递推式

递归算法分析的第3种技术是利用通用分治递推式：

$$T(n)\begin{cases} c & n = 1 \\ aT(n/b) + cn^2 & n > 1 \end{cases}$$

其中，a、b、c、k 都是常数。这个递推式描述了大小为 n 的原问题分成若干个大小为 n/b 的子问题，其中 a 个子问题需要求解，而 cn^k 是合并各个子问题的解需要的工作量。下面使用扩展递归技术对通用分治递推式进行推导，并假定 $n = b^m$。

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + cn^k \\ &= a\left(aT\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^k\right) + cn^k \\ &\cdots \\ &= a^m T(1) + a^{m-1}c\left(\frac{n}{b^{m-1}}\right)^k + \cdots + ac\left(\frac{n}{b}\right)^k + cn^k \\ &= c\sum_{i=0}^{m} a^{m-i}\left(\frac{n}{b^{m-i}}\right)^k \\ &= c\sum_{i=0}^{m} a^{m-i}b^{ik} \\ &= ca^m\sum_{i=0}^{m}\left(\frac{b^k}{a}\right)^i \end{aligned}$$

这个求和是一个几何级数，其值依赖于比率 $r = \dfrac{b^k}{a}$，注意到 $a^m = a^{\log_b n} = n^{\log_b a}$，有以下3种情况：

(1) $r < 1$：$\sum_{i=0}^{m} r^i < \dfrac{1}{1-r}$，由于 $a^m = n^{\log_b a}$，所以 $T(n) = O(n^{\log_b a})$。

(2) $r = 1$：$\sum_{i=0}^{m} r^i = m + 1 = \log_b n + 1$，由于 $a^m = n^{\log_b a} = n^2$，所以 $T(n) = O(n^k \log_b n)$。

(3) $r > 1$：$\sum_{i=0}^{m} r^i = \dfrac{r^{m+1} - 1}{r - 1} = O(r^m)$，所以，$T(n) = O(a^m r^m) = O(b^{km}) = O(n^k)$。

对通用分治递推式的推导概括为下面的主定理：

$$T(n)\begin{cases} O(n^{\log_b a}) & a > b^k \\ O(n^k \log_b n) & a = b^k \\ O(n^k) & a < b^k \end{cases}$$

1.2.4　算法的后验分析

前面我们介绍了如何对非递归算法和递归算法进行数学分析，这些分析技术能够在数量级上对算法进行精确度量。但是，数学不是万能的，实际上，许多貌似简单的算法很难用数学的精确性和严格性来分析，尤其在做平均效率分析时。

算法的后验分析（Posteriori）也称算法的实验分析，它是一种事后计算的方法，通常需要将算法转换为对应的程序并上机运行。其一般步骤如下。

（1）明确实验目的。

在对算法进行实验分析时，可能会有不同的目的，例如，检验算法效率理论分析的正确性；比较相同问题的不同算法或相同算法的不同实现间的效率，等等。实验的设计依赖于实验者要寻求什么答案。

（2）决定度量算法效率的方法，为实验准备算法的程序实现。

实验目的有时会影响甚至会决定如何对算法的效率进行度量。一般来说，有以下两种度量方法。

①计数法。在算法中的适当位置插入一些计数器，来度量算法中基本语句（或某些关键语句）的执行次数。

②计时法。记录某个特定程序段的运行时间，可以在程序段的开始处和结束处查询系统时间，然后计算这两个时间的差。在用计时法时需要注意，在分时系统中，所记录的时间很可能包含了 CPU 运行其他程序的时间（例如系统程序），而实验应该记录的是专门用于执行特定程序段的时间。例如，在 UNIX 中将这个时间称为用户时间，time 命令就提供了这个功能。

（3）决定输入样本，生成实验数据对于某些典型的算法（例如 TSP 问题），研究人员已经制定了一系列输入实例作为测试的基准（如 TSPLIB），但大多数情况下，需要实验人员自己确定实验的输入样本。一般来说，通常需要确定以下几点。

①样本的规模。一种可借鉴的方法是先从一个较小的样本规模开始，如果有必要再加大样本规模。

②样本的范围。一般来说，输入样本的范围不要小得没有意义，也不要过分大。此外，还要设计一个能在所选择的样本范围内产生输入数据的程序。

③样本的模式。输入样本可以符合一定的模式，也可随机产生。根据一个模式改变输入样本的好处是可以分析这种改变带来的影响，例如，如果一个样本的规模每次都会翻倍，可以通过计算 $T(2n)/T(n)$，考查该比率揭示的算法性能是否符合一个基本的效率类型。

如果对于相同规模的不同输入实例，实验数据和实验结果会有很大不同，则需要考虑是否包括同样规模的多个不同输入实例。例如排序算法，对于同样数据集合的不同初始排列，算法的时间性能会有很大差别。

（4）对输入样本运行算法对应的程序，记录得到的实验数据。

作为实验结果的数据需要记录下来，通常用表格或者散点图记录实验数据，散点图就是在笛卡儿坐标系中用点将数据标出。以表格呈现数据的优点是直观、清晰，可以方便地对数据进行计算，以散点图呈现数据的优点是可以确定算法的效率类型。例如表 1－1 是对某算