中国计算机学会教育专业委员会 推荐
全国高等学校计算机教育研究会 出版
高等学校规划教材

算法与数据结构
(第二版)

傅清祥　王晓东　编著

计算机学科教学计划2000

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

高等学校规划教材

算法与数据结构

（第二版）

傅清祥　王晓东　编著

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是《计算机学科教学计划 1993》的配套教材之一。它覆盖了《计算机学科教学计划 1993》中开列的关于算法与数据结构主科目的所有知识单元。其主要内容有:算法与数据结构的概念、抽象数据类型(ADT)、基于序列的 ADT(如表,栈,队列和串等)、反映层次关系的 ADT(如树,堆和各种平衡树等)、关于集合的 ADT(如字典,优先队列和并查集等)、算法设计的策略与技巧、排序与选择算法、图的算法、问题的计算复杂性、并行算法。

全书强调"算法"与"数据结构"之间密不可分的联系,因而强调融数据类型与定义在数据类型上的运算于一体的抽象数据类型,为面向对象的程序设计方法打下扎实的基础。

本书以知识单元为基本构件,具有可拆卸性和可重组性,内容丰富,表述详细,适合不同类型的院校按照不同的培养规格组织教学,其中基础部分可作为计算机学科各专业本科生的教材,高级专题部分可作为高年级本科生或研究生的教材。

图书在版编目(CIP)数据

算法与数据结构/傅清祥编著 . -2 版 . -北京:电子工业出版社, 2001.8
高等学校规划教材
ISBN 7-5053-6792-7

Ⅰ.算… Ⅱ.傅… Ⅲ.①电子计算机-计算方法-高等学校-教材②数据结构-高等学校-教材
Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(2001)第 042915 号

丛 书 名:高等学校规划教材
书　　名:**算法与数据结构(第二版)**
编 著 者:傅清祥　王晓东
责任编辑:赵家鹏
排版制作:电子工业出版社计算机排版室
印 刷 者:北京市大中印刷厂
出版发行:电子工业出版社　http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销:各地新华书店
开　　本:787×1092　1/16　印张:30　字数:764.8 千字
版　　次:2001 年 8 月第 2 版　2001 年 12 月第 2 次印刷
书　　号:ISBN 7-5053-6792-7 / TP·3821
印　　数:5 000 册　　定价:34.00 元

出 版 说 明

中国计算机学会教育专业委员会和全国高等学校计算机教育研究会(以下简称"两会"),为了适应培养我国21世纪计算机各类人材的需要,根据学科技术发展的总趋势,结合我国高等学校教育工作的现状,立足培养的学生能跟上国际计算机科学技术发展水平,于1993年5月参照ACM和IEEE/CS联合教程专题组1990年12月发表的《Computing Curricula 1991》,制定了《计算机学科教学计划1993》,并组织编写与其配套的首批18种教材。现推荐给国内有关院校,作为组织教学的参考。

《计算机学科教学计划1993》是从计算机学科的发展和社会需要出发提出的最基本的公共要求,不是针对某一具体专业(如计算机软件或计算机及应用专业),因此它适用于不同类型的学校(理科、工科及其他学科)、不同专业(计算机各专业)的本科教学。各校可以根据自己的培养目标和教学条件有选择地组织制定不同的教学计划,设置不同的课程。本教学计划的思想是将计算机学科领域的知识,分解为九个主科目(算法与数据结构、计算机体系结构、人工智能与机器人学、数据库与信息检索、人-机通信、数值与符号计算、操作系统、程序设计语言、软件方法学与工程)作为学科的公共要求;对计算机学科的教学归结为理论(数学)、抽象(实验)和设计(工程)三个过程,并强调专业教学一定要与社会需要相结合。另外,还提出了贯穿于计算机学科重复出现的十二个基本概念,在深层次上统一了计算机学科,对这些概念的理解和应用能力,是本科毕业生成为成熟的计算机学科工作者的重要标志。

为了保证这套教材的编审和出版质量,两会成立了教材编委会,制定了编写要求和编审程序。编委会对编者提出的编写大纲进行了讨论,其中一些关键性和难度较大的教材还进行了多次讨论。并且组织了部分编委对教材的质量和进度分片落实,有的教材在编审过程中召开了部分讲课教师座谈会,广泛听取意见。参加这套教材的编审者都是在该领域第一线从事教学和科研工作多年,学术水平较高,教学经验丰富,治学态度严谨的教师。这套教材的出版得到了电子工业出版社的积极支持。他们把这套教材列为出版社的重点图书出版,并制定了专门的编审出版暂行规定和出版流程,组织了专门的编辑和协调机构。

这套教材的编审出版凝聚了参加这套教材编审教师和关心这套教材的教师、参与编辑和出版工作者、以及编委会成员的汗水,他们为此作出了努力。

这套教材还得到电子工业部计算机专业教学指导委员会的支持,其中11本被选入1996~2000年全国工科电子类专业规划教材。

限于水平和经验,这套教材肯定还会有缺点和不足,希望使用教材的单位、教师和同学积极提出批评建议,共同为提高教学质量而努力。

中国计算机学会教育专业委员会
全国高等学校计算机教育研究会

教材编审委员会成员名单

前　言

本书是全国高等学校计算机教育研究会和中国计算机学会教育专业委员会联合制定的《计算机学科教学计划 1993》的配套系列教材中的一本，旨在专门讲解作为计算机学科公共要求的九个主科目之一——算法与数据结构。

特色：

本书按照《计算机学科教学计划 1993》的教学大纲编写，既考虑到我国国情，又努力与国际上计算机学科的教学要求接轨。强调"算法"与"数据结构"之间密不可分的联系，因而强调融数据类型与定义在该类型上的运算于一体的抽象数据类型，为面向对象的程序设计方法奠定基础；体现计算机学科方法论的理论、抽象和设计三个过程，知识面较宽，且有一定的深度；反复再现计算机学科中用到的大问题的复杂性、效率、抽象的层次、重用、折衷等带有普遍性的概念，让读者在更深的层次上掌握算法与数据结构这一主科目。本书还介绍了国内同类教材中至今尚未介绍过的有关算法与数据结构的一些新知识、新成果。

全书以知识单元为基本构件，各知识单元相对独立，具有可拆卸性和可重组性，便于不同类型的院校按照不同的培养规格组织教学。

内容：

全书共十一章。第一章引出算法与数据结构的基本概念（如算法的复杂性、抽象数据类型（ADT）等）；第二章到第五章由浅入深依次介绍基于序列的 ADT（如表、栈、队列和串等）、反映层次关系的 ADT（如堆、红黑树、2-3 树、AVL 树和检索树等）和表达集合的 ADT（如字典、优先队列和并查集等）；第六章系统地阐述算法设计的策略和技巧；第七、八章分别介绍在实际应用中很基本的排序算法和图的算法；第九章讨论问题的计算复杂性和问题按复杂性的分类；第十章介绍并行计算模型和设计并行算法的一些基本技术；最后，第十一章讲解四个高级专题。

表述：

为了使选用本书的教师容易教，学生容易学，软、硬件研制开发人员查阅时容易懂，本书在表述上采取四条措施。

1. 对于每一个比较抽象的概念，都举实例加以说明；对于每个比较抽象的定理都有具体的应用例证帮助理解；对于每一个比较复杂的算法都有简单的算例示范。

2. 大量运用图表（全书包含 300 多个图表），使概念、定理、算法变得形象直观。

3. 定理的证明尽量初等，公式的推导尽量详细。

4. 用类 Pascal 作为算法的描述语言。考虑到本书描述算法的目的在于教学，在于让读者容易抓住算法的基本思想和弄清算法的基本步骤，而不是要提供可直接上机运行的源代码，我们认为用类 Pascal 来描述算法更符合我们的目的，因为它比较简明，又比较接近自然语言。

用法：

根据我们的经验，本书的第一、二、三、四、五、七、八章可作为计算机学科大专高年级和本科低年级的教材，而第六、九、十、十一章可作为本科高年级和研究生的教材。全书可供自学者系统地自学，也可作为计算机软、硬件研制开发人员查阅算法和数据结构的参考手册。

分工：

本书由傅清祥和王晓东合作编著。第一、三、四、六、七、十一章由傅清祥撰写，第二、五、八、九、十章由王晓东撰写。最后由傅清祥负责统稿。文稿的录入、校对、排版以及图表的整理由林志庆、田俊、倪一涛、高明、谢攀、刘涛等同志承担。

差错：

一本70多万字的书，不可能没有缺点和错误。我们热诚欢迎读者批评指正，那怕是表述不当或笔误。对于读者在阅读中发现的缺点和错误，请通过书信或电子邮件告诉我们，我们一方面将表示感谢，另一方面将及时整理成勘误表放在Internet的福州大学站点上供读者查询更正。我们同时热诚欢迎读者对本书的任何建设性意见，以便将来需要再版时改进。我们的通信地址是福州市工业路523号福州大学计算机科学系；我们的E-mail地址分别是 qxfu @ fzu.edu.cn 和 wangxd @ fzu.edu.cn。

致谢：

我们首先感谢杨文龙、张吉锋两位教授和朱毅副编审对我们承接编著本书任务的鼓励，以及在成书过程中给予的指导和帮助；感谢电子工业部计算机专业教学指导委员会对本书的重视，把本书列为计算机专业"95"规划教材。

本书先后经过曹厚隆教授的一审和朱关铭教授的二审。我们对他们一丝不苟的负责精神和所提出的许多宝贵意见表示由衷的谢意。我们还感谢福州大学计算机科学系《算法与数据结构》重点课程工作小组同事们的帮助和计算机应用重点实验室为我们提供上机条件；感谢叶先健博士从美国为我们寄来宝贵的参考资料。

编著者

于福州大学

2001年4月

目　录

第一章　绪论 …… (1)
　第一节　算法的复杂性 …… (1)
　　一、比较两对算法的效率 …… (1)
　　二、复杂性的计量 …… (4)
　　三、复杂性的渐近性态及其阶 …… (7)
　　四、复杂性渐近阶的重要性 …… (10)
　　五、算法复杂性渐近阶的分析 …… (12)
　　六、递归方程解的渐近阶的求法 …… (16)
　第二节　算法表达中的抽象机制 …… (26)
　　一、从机器语言到高级语言的抽象 …… (27)
　　二、抽象数据类型 …… (30)
　　三、使用抽象数据类型带来的好处 …… (35)
　　四、数据结构、数据类型和抽象数据类型 …… (35)
　习题 …… (37)

第二章　表 …… (40)
　第一节　ADT 表 …… (40)
　第二节　表的实现 …… (42)
　　一、表的数组实现 …… (42)
　　二、表的指针实现 …… (45)
　　三、表的游标实现 …… (48)
　　四、循环链表 …… (51)
　　五、双链表 …… (52)
　第三节　栈 …… (54)
　　一、ADT 栈 …… (54)
　　二、栈的数组实现 …… (55)
　　三、栈的指针实现 …… (57)
　第四节　队列 …… (58)
　　一、ADT 队列 …… (58)
　　二、用指针实现队列 …… (59)
　　三、用循环数组实现队列 …… (60)
　第五节　映射 …… (64)
　　一、ADT 映射 …… (64)
　　二、用数组实现映射 …… (64)
　　三、用表实现映射 …… (65)
　习题 …… (67)

第三章　串 …… (71)
　第一节　ADT 串 …… (71)

第二节　串的实现 …… (72)
一、串的数组实现 …… (72)
二、串的指针实现 …… (73)
三、串的块链表示法 …… (73)
四、串的堆结构 …… (74)
第三节　模式匹配 …… (75)
一、朴素的模式匹配算法 …… (75)
二、模式匹配的 KMP 算法 …… (76)
习题 …… (82)

第四章　树 …… (84)
第一节　树的定义 …… (84)
第二节　二叉树 …… (86)
第三节　树的遍历 …… (88)
第四节　ADT 树 …… (90)
第五节　树的实现方法 …… (92)
一、父亲数组表示法 …… (92)
二、儿子链表表示法 …… (93)
三、左儿子右兄弟表示法 …… (95)
第六节　二叉树的实现及其应用 …… (96)
一、二叉树的顺序存储结构 …… (96)
二、二叉树的结点度表示法 …… (98)
三、二叉树的链式存储结构 …… (98)
四、果园或森林的二叉树表示 …… (99)
五、线索二叉树 …… (100)
六、二叉树的应用 …… (102)
习题 …… (104)

第五章　集合 …… (108)
第一节　以集合为基础的抽象数据类型 …… (108)
一、集合的定义和记号 …… (108)
二、定义在集合上的基本运算 …… (109)
三、集合的简单表示法 …… (109)
第二节　字典 …… (113)
一、实现字典的简单方法 …… (113)
二、用散列表实现字典 …… (114)
三、用散列表实现映射 …… (123)
第三节　有序字典 …… (124)
一、有序字典的定义 …… (125)
二、用数组实现有序字典 …… (125)
三、用二叉搜索树实现有序字典 …… (127)
第四节　平衡树 …… (134)
一、红黑树 …… (134)
二、2-3 树 …… (147)
第五节　优先队列 …… (156)
一、优先队列的定义 …… (156)

二、 优先队列的字典式实现 …… (157)
三、 优先级树和堆 …… (159)
四、 用数组实现堆 …… (161)
第六节 并查集 …… (163)
一、 并查集的定义及其简单实现 …… (163)
二、 并查集的快速实现 …… (165)
三、 并查集的树实现 …… (167)
第七节 检索树 …… (169)
一、 检索树与检索树结点 …… (169)
二、 用数组表示检索树结点 …… (170)
三、 用链接表表示检索树结点 …… (171)
四、 检索树的效率 …… (171)
习题 …… (172)

第六章 算法设计策略与技巧 …… (177)
第一节 递归技术与分治法 …… (177)
一、 递归技术 …… (177)
二、 分治法的基本思想 …… (181)
三、 大整数的乘法 …… (182)
四、 Strassen 矩阵乘法 …… (184)
五、 最接近点对问题 …… (186)
六、 循环赛日程表 …… (190)
第二节 动态规划 …… (191)
一、 计算矩阵连乘积 …… (191)
二、 动态规划算法的基本要素 …… (196)
三、 最长公共子序列 …… (199)
四、 凸多边形的最优三角剖分 …… (203)
第三节 贪心算法 …… (206)
一、 活动安排问题 …… (206)
二、 贪心算法的基本要素 …… (209)
三、 哈夫曼编码 …… (211)
四、 贪心算法的理论基础 …… (215)
第四节 回溯法 …… (220)
一、 回溯法的--般描述 …… (220)
二、 n 后问题 …… (225)
三、 子集和问题 …… (227)
四、 图的 m-着色问题 …… (230)
五、 回溯法的效率分析 …… (233)
第五节 限界剪枝法 …… (236)
一、 最小耗费搜索法 …… (236)
二、 限界与剪枝 …… (241)
三、 旅行售货员问题 …… (245)
习题 …… (248)

第七章 排序与选择 …… (253)
第一节 简单排序算法 …… (253)
一、 冒泡排序 …… (253)

二、 插入排序 …… (254)
三、 选择排序 …… (254)
四、 简单排序算法的计算复杂性 …… (255)
第二节 快速排序 …… (256)
一、 算法的基本思想及其实现 …… (256)
二、 快速排序的性能 …… (258)
三、 随机快速排序算法 …… (258)
第三节 堆排序 …… (259)
一、 堆排序算法的基本思想及其实现 …… (259)
二、 堆排序算法的计算复杂性 …… (261)
第四节 线性时间排序 …… (261)
一、 计数排序 …… (262)
二、 桶排序 …… (263)
三、 基数排序 …… (264)
第五节 中位数与第 k 小元素 …… (267)
一、 平均情况下的线性时间选择算法 …… (267)
二、 最坏情况下的线性时间选择算法 …… (269)
习题 …… (271)

第八章 图 …… (275)
第一节 图的基本概念 …… (275)
一、 图及其相关术语 …… (275)
二、 抽象数据类型 ADT 图 …… (277)
第二节 图的表示法 …… (278)
一、 邻接矩阵表示法 …… (278)
二、 邻接表表示法 …… (279)
第三节 图的遍历 …… (281)
一、 深度优先搜索 …… (281)
二、 广度优先搜索 …… (283)
第四节 图的连通性 …… (284)
一、 深度优先生成森林 …… (284)
二、 无圈有向图 …… (285)
三、 有向图的强连通分支 …… (286)
四、 无向图的割点和双连通分支 …… (287)
第五节 最小生成树 …… (288)
一、 最小生成树性质 …… (289)
二、 Prim 算法 …… (289)
三、 Kruskal 算法 …… (291)
第六节 最短路径 …… (293)
一、 单源最短路径 …… (293)
二、 所有顶点对之间的最短路径 …… (295)
第七节 图匹配 …… (298)
习题 …… (299)

第九章 问题的计算复杂性 …… (304)
第一节 计算模型 …… (304)

一、 随机存取机 RAM …… (305)
二、 随机存取存储程序机 RASP …… (310)
三、 RAM 模型的变形与简化 …… (314)
四、 图灵机 …… (317)
五、 图灵机模型与 RAM 模型的关系 …… (320)
第二节 问题的计算时间下界 …… (322)
一、 问题的输入、 输出及平凡下界 …… (323)
二、 信息论下界 …… (323)
三、 对手论证方法 …… (324)
四、 Ben_Or 下界定理 …… (330)
五、 问题变换与计算复杂性归约 …… (334)
第三节 P 类与 NP 类 …… (337)
一、 非确定性图灵机 …… (338)
二、 P 类与 NP 类语言 …… (339)
三、 “证书”与 VP 类语言 …… (341)
四、 问题和语言 …… (342)
第四节 NP-完全性 …… (343)
一、 多项式时间变换与 NP-完全问题 …… (343)
二、 Cook 定理 …… (344)
三、 几个典型的 NP-完全问题 …… (347)
第五节 NP-完全问题的近似解法 …… (357)
一、 近似算法的性能 …… (358)
二、 顶点覆盖问题的近似算法 …… (358)
三、 旅行售货员问题的近似算法 …… (360)
四、 集合覆盖问题的近似算法 …… (362)
五、 子集和问题的近似算法 …… (365)
习题 …… (369)

第十章 并行算法 …… (372)
第一节 并行计算模型 …… (372)
一、 PRAM 模型 …… (372)
二、 同步与控制 …… (374)
三、 并行算法的表达 …… (374)
四、 并行算法的性能指标 …… (375)
五、 运行时间和工作量有效性 …… (375)
第二节 并行算法的基本设计技术 …… (376)
一、 平衡树方法 …… (376)
二、 指针跳越技术 …… (377)
三、 欧拉回路技术 …… (380)
四、 并行分治法 …… (382)
五、 划分原理 …… (385)
六、 流水线技术 …… (386)
七、 接力技术 …… (388)
八、 递归的并行随机消元法 …… (390)
九、 确定性破对称技术 …… (393)
第三节 EREW 算法与 CRCW 算法的速度比较 …… (398)
一、 并发读对提高速度的作用 …… (398)

二、 并发写对提高速度的作用 …… (400)
三、 CRCW 算法速度的上界 …… (401)
习题 …… (403)

第十一章 高级专题 …… (406)
第一节 算法的分摊时间分析 …… (406)
一、 累计方法 …… (407)
二、 记账方法 …… (409)
三、 势能方法 …… (410)
四、 自适应二叉搜索树 …… (412)
第二节 可并优先队列 …… (418)
一、 可并优先队列的定义 …… (418)
二、 用二项堆实现可并优先队列 …… (419)
三、 用 Fibonacci 堆实现可并优先队列 …… (430)
第三节 数据结构的扩充与联合 …… (442)
一、 动态选择树——红黑树的扩充 …… (442)
二、 数据结构扩充的方法 …… (446)
三、 区间树 …… (447)
四、 数据结构的联合 …… (451)
第四节 静态数据结构的动态化方法 …… (452)
一、 可分解搜索问题 …… (453)
二、 静态数据结构的半动态化 …… (454)
三、 静态数据结构的另一种半动态化方法 …… (458)
四、 静态数据结构的全动态变换 …… (460)
五、 其他动态化方法 …… (461)
习题 …… (462)
参考文献 …… (465)

第一章 绪 论

第一节 算法的复杂性

我们开篇就讲算法的复杂性是因为它是算法效率的度量，是评价算法优劣的重要依据。它所提供的概念和方法，本书到处都要用到。

一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上面，所需要的资源越多，我们就说该算法的复杂性越高；反之，所需要的资源越少，我们就说该算法的复杂性越低。

计算机的资源，最重要的是时间和空间(即存储器)资源。因而，算法的复杂性有时间复杂性和空间复杂性之分。

不言而喻，对于任意给定的问题，设计出复杂性尽可能低的算法是我们在设计算法时追求的一个重要目标；另一方面，当给定的问题已有多种算法时，选择其中复杂性最低者，是我们在选用算法时应遵循的一个重要准则。因此，算法的复杂性分析对算法的设计或选用有着重要的指导意义和实用价值。

关于算法的复杂性，有两个问题要弄清楚：

(1)用怎样的一个量来表达一个算法的复杂性；

(2)对于给定的一个算法，怎样具体计算它的复杂性。

让我们从比较两对具体算法的效率开始。

一、 比较两对算法的效率

考虑问题 1：已知不重复且已经按从小到大排好的 m 个整数的数组 $A[1..m]$(为简单起见，还设 $m=2^k$，k 是一个确定的非负整数)。对于给定的整数 c，要求寻找一个下标 i，使得 $A[i]=c$；若找不到，则返回一个 0 。

问题 1 的一个简单的算法是：从头到尾扫描数组 A。照此，或者扫到 A 的第 i 个分量，经检测满足 $A[i]=c$；或者扫到 A 的最后一个分量，经检测仍然不满足 $A[i]=c$。我们用一个函数 search 来表达这个算法：

```
function search (c:integer):integer;
var j:integer;
begin
  j:=1;{初始化 j}
  {在还没有到达 A 的最后一个分量且等于 c 的分量还没有找到时，查找下一个分量并进行检测}
  while (A[j]<c) and (j<m) do j:=j+1;
  if A[j]=c then search:=j {在数组 A 中找到等于 c 的分量，且此分量的下标为 j}
```

```
    else search := 0 {在数组 A 中找不到等于 c 的分量}
  end;
```

容易看出，在最坏的情况下，这个算法要检测 A 的所有 m 个分量才能判断在 A 中找不到等于 c 的分量。

解决问题 1 的另一个算法利用到已知条件中 A 已排好序的性质。它首先拿 A 的中间分量 $A[m/2]$ 与 c 比较，如果 $A[m/2]=c$ 则解已找到。如果 $A[m/2]>c$，则 c 只可能在 $A[1],A[2],\cdots,A[m/2-1]$ 中，因而下一步只要在 $A[1],A[2],\cdots,A[m/2-1]$ 中继续查找；如果 $A[m/2]<c$，则 c 只可能在 $A[m/2+1],A[m/2+2],\cdots,A[m]$ 中，因而下一步只要在 $A[m/2+1],A[m/2+2],\cdots,A[m]$ 继续查找。不管哪一种情形，都把下一步需要继续查找的范围缩小了一半。再拿这一半的子数组的中间分量与 c 比较，重复上述步骤。照此重复下去，总有一个时候，或者找到一个 i 使得 $A[i]=c$，或者子数组为空（即子数组下界大于上界）。前一种情形找到了等于 c 的分量，后一种情形则找不到。

这个新算法因为有反复把供查找的数组分成两半，然后在其中一半继续查找的特征，我们称为二分查找算法。它可以用函数 b_search 来表达：

```
  function b_search (c:integer):integer;
  var L,U,I:integer;
      {U 和 L 分别是要查找的数组下标的上界和下界}
      found:boolean;
  begin
    L := 1,U := m;{初始化数组的下、上界}
    found := false;
    {当前要查找的范围是 A[L],…,A[U]。当等于 c 的分量还没有找到且 U≥L 时，继续查找}
    while (not found) and (U≥L) do
      begin
        I := (U+L) div 2;{找数组的居中分量}
        if c=A[I] then found := true
        else if c>A[I] then L := I+1
          else U := I-1
      end;
    if found then b_search := I
      else b_search := 0
  end;
```

容易理解，在最坏情况下，最多只要检测 A 中的 $k(=\log m)+1$ 个分量，就可以判断 c 是否在 A 中。这里 $\log m$ 表示 m 的以 2 为底的对数即 $\log_2 m$。

算法 search 和 b_search 解的是同一个问题，但在最坏的情况下（所给定的 c 不在 A 中），两个算法所需要检测的分量的个数却大不相同，前者要 $m=2^k$ 个，后者只要 $k+1$ 个。可见算法 b_search 比 search 高效得多。或者换句话说，算法 b_search 的时间复杂性比 search 要低得多。

考虑另一个问题，即问题 2：对于给定的非负整数 N，要求计算出菲波纳契（Fibonacci）数列的前 $N+1$ 项：

$a_0, a_1, a_2, \cdots, a_N$

其中 $a_0=0, a_1=1, a_i=a_{i-1}+a_{i-2}, i\geqslant 2$ (1.1.1)

解这个问题也有多种算法。其中比较简单的一个算法是写一个计算一般项 a_k 的函数 A，然后靠反复地调用 A，产生 a_0 到 a_N。这一算法可用一个过程 Seq1 来表达：

```
procedure Seq1(N:integer);
var  i:integer;
function  A(k:integer):integer;
begin
  if k=0 then A:=0
  else if k=1 then A:=1
  else A:=A(k-1)+A(k-2)
end;
begin
  if N<0 then error
  else for i:=0 to N do writeln(A(i))
end;
```

其中 error 是一个出错处理过程。因为与我们的主题没有什么关系，我们不去关心它的细节，而只是在需要时引用它，往后再在算法(程序)中出现时也不再说明。

算法 Seq1 没有很好地利用斐波纳契数列中项间的递推关系式(1.1.1)。下面是另一个算法。由式(1.1.1)可见，除了 a_0 和 a_1 外，数列的每一项都是它的前面紧挨着的两项之和。换句话说，在得到数列的前 i 项之后，只要把最近得到的第 i 项和第 $i-1$ 项相加就可以得到第 $i+1$ 项。重复这样做，让 i 跑遍 2 到 N，便产生出所要的数列。我们用过程 Seq2 来表达这一算法：

```
procedure Seq2(N:integer);
var L0,L1,i,temp:integer;
{L0 和 L1 始终表示已经得到的数列的最后两项，即 a0,a1,…,ai 中的 ai-1和 ai}
begin
if N<0 then error
  else
  begin
    L0:=0; L1=1;{初始化 L0 和 L1}
    for i:=0 to N do
      begin
        writeln(L0);{输出 ai}
        temp:=L1;
        L1:=L0+L1;
        L0:=temp
      end
  end
end;
```

我们看到，在 Seq1 中，对于每一个 $k(0\leqslant k<N)$，a_k 被重复计算多次。图 1-1 是一个直观的

说明。它告诉我们，仅在计算 a_5 时，$A(3)$，$A(2)$，$A(1)$和 $A(0)$就分别被重复调用 2，3，5 和 3 次，即 a_3，a_2，a_1，a_0 分别被重复计算了 2，3，5 和 3 次。不用说，如果可以避免不必要的重复，算法的效率将会明显提高。

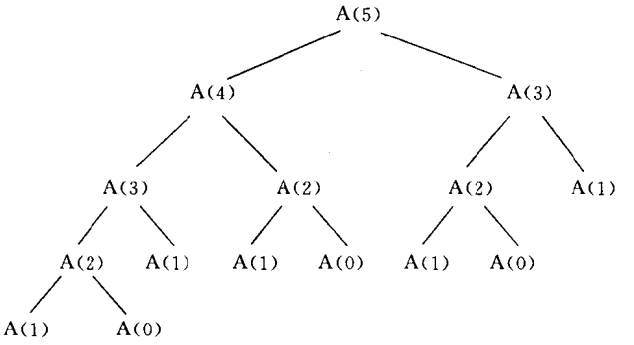

图 1-1　算法 Seq1 在计算 a_5 时调用 A 的情况

相比之下，算法 Seq2 就避免了上述的重复计算，即对每一个 $k(0\leqslant k<N)$，a_k 只计算一次。因而很明显，Seq2 的效率要比 Seq1 高。

以上列举的两对算法说明：解同一个问题，算法不同，则计算的工作量就不同，所需要的计算时间随之不同，即复杂性不同。

有实验表明，运行这两对算法的时间曲线分别如图 1-2 和图 1-3 所示。

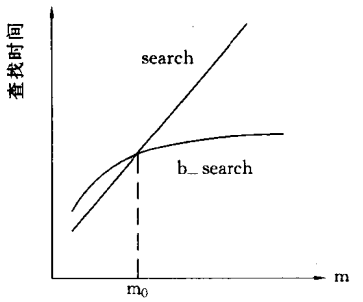

图 1-2　search 与 b_search 的比较

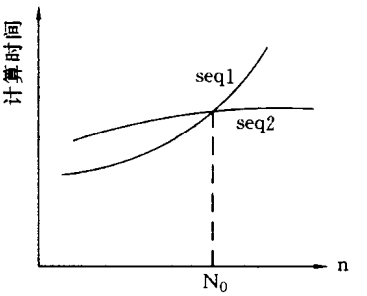

图 1-3　Seq1 与 Seq2 的比较

图 1-2 表明，当 m 适当大($m>m_0$)时，算法 b_search 就比 search 省时，而且当 m 更大时，节省的时间急剧增加。同样地，图 1-3 表明，当 N 适当大($N>N_0$)时，Seq2 比 Seq1 省时，而且当 N 更大时，两者的时耗悬殊越来越显著。

不过，应该指出：用实例的运行时间来度量算法的时间复杂性并不合适，因为这个实例时间与运行该算法的实际计算机的性能有关。换句话说，这个实例时间不单纯反映算法的效率而是反映包括运行该算法的计算机在内的综合效率。我们引入复杂性的概念是为了比较解决同一个问题的不同算法本身的效率，而不想去比较运行该算法的计算机的性能。因而，不应该取算法运行的实例时间作为算法复杂性的尺度。我们希望，尽量单纯地反应作为算法精髓的计算方法本身的效率，而且在不实际运行该算法的情况下就能分析出它所需要的时间和空间。

二、复杂性的计量

算法的复杂性是算法运行所需要的计算机资源的量，需要的时间资源的量称为时间复杂性，需要的空间(即存储器)资源的量称为空间复杂性。这个量应该集中反映算法中所采用的方