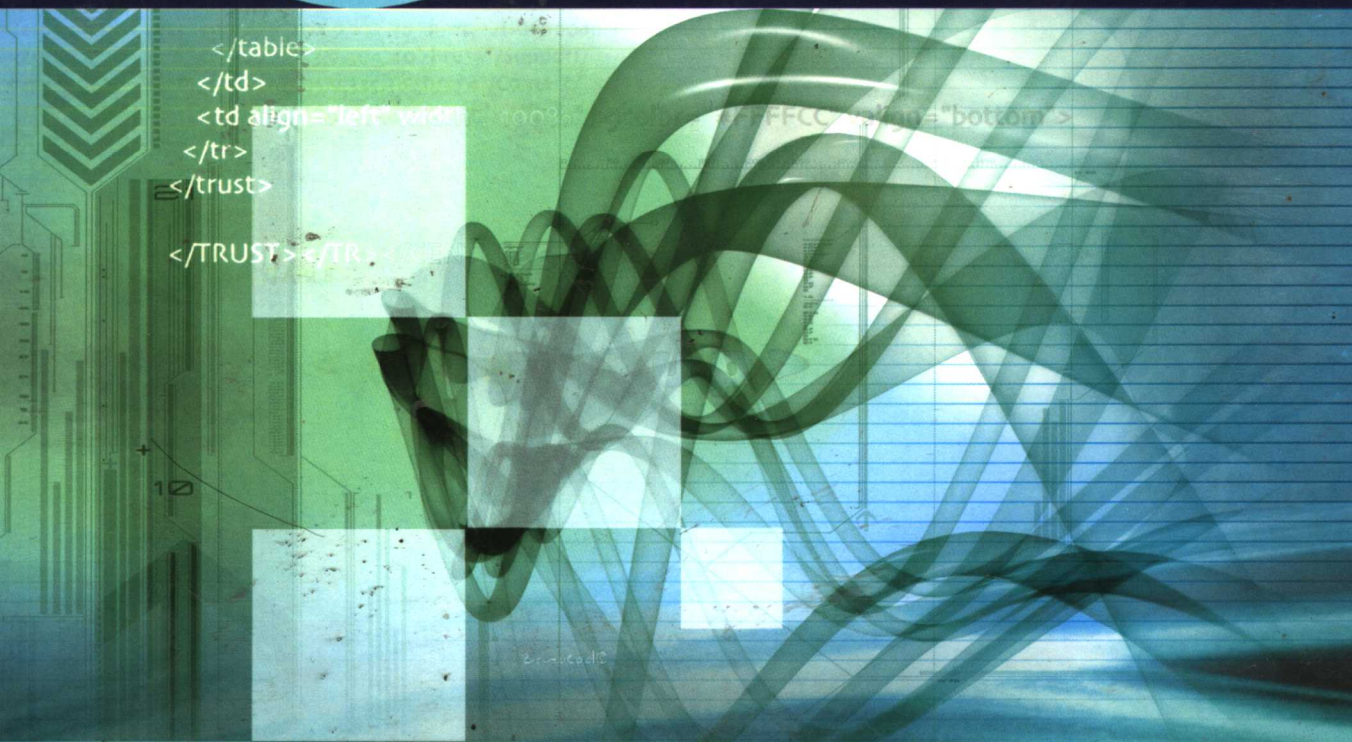
普通高等教育“十一五”规划教材

高等院校计算机科学与技术系列教材

算法设计与分析

田翠华 著



冶金工业出版社

普通高等教育“十一五”规划教材
高等院校计算机科学与技术系列教材

算法设计与分析

田翠华 著

北 京
冶 金 工 业 出 版 社

内 容 简 介

本书是根据普通高等教育“十一五”国家级规划教材的指导精神而编写的。

本书讲解算法的设计与分析的相关知识，内容包括三大部分。第一部分介绍算法基本概念和算法基础知识；第二部分介绍一些经典算法，包括分治法、贪心法、动态规划、回溯法、分支限界法、概率算法、NP 问题、近似算法、新型算法以及并行算法；第三部分是上机实训，选择一些实际问题，编写程序上机实现，通过实际动手编程解决问题来加强学生的实践能力。另外，书中还配有大量的习题，以便读者检验和强化所学知识，起到事半功倍的作用。

本书内容详实、结构清晰，不仅适合作为普通高校本科生和研究生的教材，也适合作为理工学科的工程技术人员在实际工作过程中使用的技术参考书。

图书在版编目（CIP）数据

算法设计与分析 / 田翠华著. —北京：冶金工业出版社，2007.8

普通高等教育“十一五”规划教材

ISBN 978-7-5024-4361-0

I. 算… II. 田… III. ①电子计算机—算法设计—高等学校—教材②电子计算机—算法分析—高等学校—教材 IV. TP301.6

中国版本图书馆 CIP 数据核字（2007）第 109127 号

出版人 曹胜利（北京沙滩嵩祝院北巷 39 号，邮编 100009）

责任编辑 程志宏

ISBN 978-7-5024-4361-0

广州锦昌印务有限公司印刷；冶金工业出版社发行；各地新华书店经销

2007 年 8 月第 1 版第 1 次印刷

787mm × 1092mm 1/16； 16.5 印张； 379 千字； 256 页

30.00 元

冶金工业出版社发行部 电话：（010）64044283 **传真：**（010）64027893

冶金书店 地址：北京东四西大街 46 号（100711） **电话：**（010）65289081

（本社图书如有印装质量问题，本社发行部负责退换）

作者简介

田翠华：辽宁沈阳人，硕士，博士在读，讲师，聘任副教授，硕士研究生导师，计算机算法与网格专家。2002 年硕士研究生毕业于沈阳理工大学计算机软件与理论专业。出版了学术著作《计算机网络与通信技术》与专著《算法设计与分析》，公开发表学术论文 16 篇，其中，EI 检索 3 篇，核心期刊 6 篇。先后出席了在新疆大学、大连海事大学召开的国际学术会议，并作了专题学术报告，在基于 WSRF 的交通信息服务网格和并行算法以及网络技术等方面有较大的学术影响。所讲授的课程《算法设计与分析》、《编译技术》、《计算机网络》、《网络英语》(双语教学)、《网格计算》和《数据安全与密码学》等课程，都是理论与实践紧密结合、深受学生欢迎的课程。目前是沈阳计算机学会会员，沈阳工业大学软件教研室教师。

电子信箱：rosswindy@163.com

◎ 责任编辑/程志宏
◎ 封面设计/凌　波

前　言

一、关于本书

本书是根据普通高等教育“十一五”国家级规划教材的指导精神而编写的。

算法的设计与分析是计算机科学的核心问题之一，也是计算机科学与技术专业本科及研究生的一门重要的专业基础课。其内容是研究计算机领域及其相关领域中的一些非数值计算的常用算法。通过学习，使学生掌握算法设计的常用方法，以便去解决计算机科学与工程领域中较为复杂的实际问题。此外对分析算法，估计算法的时间与空间复杂性也作一些介绍，但不作为重点。

算法知识理论性较强，涉及的范围又很广，给学习和理解造成困难。本书的编写条理清晰、内容详实、逻辑严谨、深入浅出，利于算法知识的教与学。此外，本书中的算法均用自然语言来表述其思路，再以类 C 语言来描述，程序结构清楚，构思精巧，对程序代码做了必要的注释，力求简洁明了、通俗易懂。

二、本书结构

本书介绍算法的设计与分析的相关知识，包括三个部分。

第一部分（第 1~2 章）：介绍算法基本概念和算法基础知识。

第 1 章：算法概述。对分析算法的抽象表示、算法渐进复杂度以及如何对算法进行设计与分析作了简要的阐述。

第 2 章：递归技术。学好递归技术为以后各章问题算法的求解打下坚实的基础，是必须要掌握的。

第二部分（第 3~12 章）：经典算法的介绍，主要有分治法、贪心法、动态规划、回溯法、分枝限界法、概率算法、NP 问题、近似算法、新型算法、并行算法。

第 3 章：分治法。它作为一种算法设计策略是非常有价值的，可以求解许多算法问题。

第 4 章：贪心法。这也是一种重要的算法设计策略，它与动态规划算法的设计思想有一定的联系，但其效率更高。按贪心法设计出的许多算法能产生最优解。其中有许多典型问题和典型算法可供学习和使用。

第 5 章：动态规划。以具体实例详述动态规划算法的设计思想、适用性以及算法的设计要点。

第 6 章：回溯法。介绍其概念以及所要解决的问题。

第 7 章：分支限界法。介绍其概念以及所要解决的问题，它同第 6 章的回溯法适合于处理难解问题，其解题的思想各具特色，值得学习和掌握。

第 8 章：概率算法概述。介绍了概率算法的概念及其可以解决的一些常见的问题。

第 9 章：NP 问题。NP 完全问题是计算机科学理论中最重要的问题，本章简要介绍了 P 类与 NP 类问题以及 NP 完全问题。

第 10 章：近似算法。介绍了近似算法的概念及其可以解决的一些常见的问题。

第 11 章：新型算法。介绍了一些最近流行起来的算法，如加密算法等。

第 12 章：并行算法。并行算法也是最近流行的一种算法，本章简要介绍了并行算法及其编程实现。

第三部分（第 13 章）：是前面章节所学算法的实际应用，通过多个实例使读者掌握算法的设计与分析中的思想，并学以致用。

三、本书特点

本书具有以下特点：

（1）注重基础知识，理论联系实际，丰富详实。

（2）内容上，科学先进，充分体现计算机学科知识更新要快的特点，及时更新知识，与时俱进，确保教材处于学科前沿，以拓宽学生知识面，培养学生的创新能力。

（3）结构上，清晰、严谨、精炼、深入浅出。每章开头有内容提要，给出重点和难点；正文全面、简练、主次分明；结尾有本章小结，归纳、总结与体会。

四、适用对象

本书不仅适合作为普通高校本科生和研究生的教材，也适合作为理工学科的工程技术人员在实际工作过程中的技术参考书。

在本书的编写过程中，得到了爱人田洪涛，同事贾威、关沫和王博，教研室书记尹铁源，主任王宏生，院长刘丽钧等的关心帮助，特别是软件学院牛连强院长的热心指导，在此表示衷心的感谢！

由于时间仓促，作者水平有限，书中疏漏之处在所难免，欢迎广大读者批评指正，以使其能在使用中不断地得到改进，日臻完善。联系方式如下：

电子邮箱：service@cnbook.net

网址：www.cnbook.net

本书电子教案、习题参考答案可在该网站下载，此外，该网站还有一些其他相关书籍的介绍，可以方便读者选购参考。

编　者

2007 年 7 月

目　录

第 1 章　算法概述 1

1.1　算法概念 1
1.1.1　什么是算法 1
1.1.2　抽象表达算法机制 2
1.2　算法的复杂度 4
1.2.1　算法三性态 4
1.2.2　算法复杂度 5
1.3　算法设计与分析的步骤 12
1.3.1　利用算法进行问题求解的过程 12
1.3.2　如何设计算法 13
1.3.3　如何表示算法 14
1.3.4　如何确认算法 14
1.3.5　如何分析算法 14
1.4　算法描述语言简介 16
1.4.1　C 语言中的标准数据类型 16
1.4.2　C 语言中的运算符 17
1.4.3　C 语言中的语句简介 18
小结 20
习题一 20
一、选择题 20
二、填空题 21
三、简答题 22
四、计算题 22
五、上机题 23

第 2 章　递归技术 24

2.1　递归技术概述 24
2.1.1　什么是递归技术 24
2.1.2　递归技术的基本思想 26
2.2　Hanoi 塔问题 27
2.3　递归方程的建立与求解 28
2.3.1　递推法 29
2.3.2　生成函数法 30
2.3.3　特征方程法 31
2.3.4　数学归纳法 32
2.3.5　不规则解法 33
2.4　递归消除 33
2.4.1　简单递归消除 34
2.4.2　基于栈的递归消除 36
小结 38
习题二 38
一、选择题 38
二、填空题 38
三、简答题 38
四、计算题 38
五、上机题 39

第 3 章　分治法 40

3.1　分治法概述 40
3.1.1　什么是分治法 40
3.1.2　分治法的基本思想 41
3.1.3　分治法的基本要素 41
3.2　二分检索技术 41
3.2.1　二分检索算法描述 42
3.2.2　最坏情况分析 42
3.2.3　平均复杂度分析 44
3.2.4　以比较为基础的检索时间下界 45
3.3　找第 K 小元素 47
3.3.1　分划点 m 的选取 47
3.3.2　随机选择算法 49
3.4　分治乘法 50
3.4.1　大整数相乘 50
3.4.2　多项式乘法 51
3.4.3　矩阵乘法 58
3.5　棋盘覆盖 59
3.6　分治合并排序 61
3.6.1　什么是合并 62
3.6.2　合并排序的基本思想 62
3.6.3　二路合并排序算法 63
3.7　分治快速排序 65

3.7.1 快速排序的基本思想 65
3.7.2 示例 67
3.8 常见分治 68
3.8.1 快速傅立叶变换 68
3.8.2 傅立叶变换的逆变换 70
3.8.3 利用傅立叶理论求解多项式相乘 71
小结 72
习题三 72
一、选择题 72
二、填空题 72
三、简答题 72
四、计算题 72
五、上机题 73

第 4 章 贪心法 74

4.1 贪心算法概述 74
4.1.1 什么是贪心法 74
4.1.2 贪心法的基本思想 75
4.1.3 贪心法基本要素 75
4.2 背包问题 77
4.3 磁带存储 81
4.3.1 单磁带最优存储 81
4.3.2 多磁带最优存储 83
4.4 作业调度 84
4.4.1 顺序选择法 85
4.4.2 最大期限选择法 86
4.5 启发式算法 88
4.6 贪心法的理论基础 91
4.6.1 拟阵 91
4.6.2 带权拟阵的贪心算法 92
4.6.3 任务时间表问题 94
4.7 常见贪心算法问题 97
4.7.1 最优装载 97
4.7.2 哈夫曼编码 99
4.7.3 单源最短路径 100
4.7.4 最小生成树 101
小结 103
习题四 104
一、选择题 104
二、填空题 104
三、简答题 104
四、计算题 104
五、上机题 105

第 5 章 动态规划 106

5.1 动态规划概述 106
5.1.1 什么是动态规划 106
5.1.2 动态规划的基本思想 106
5.1.3 动态规划的基本要素 108
5.2 最短路径 109
5.3 0/1 背包问题 112
5.4 多矩阵乘积 120
5.5 货郎担问题 125
5.6 资源分配 127
小结 130
习题五 130
一、选择题 130
二、填空题 130
三、简答题 131
四、计算题 131
五、上机题 131

第 6 章 回溯法 132

6.1 概述 132
6.1.1 什么是回溯法 132
6.1.2 回溯法的基本思想 133
6.1.3 回溯法的算法框架与符号 133
6.2 n-皇后问题 135
6.3 图的 m 着色问题 140
6.3.1 图 m 着色问题的回溯法求解 140
6.3.2 图 m 着色问题的递归回溯算法 140
6.4 批处理作业调度问题 141
6.5 其他常见回溯法问题 142
6.5.1 最大团问题 142
6.5.2 旅行售货员问题 143
6.5.3 连续邮资问题 144

6.5.4 电路板排列问题 145
小结 147
习题六 147
一、选择题 147
二、填空题 147
三、简答题 147
四、计算题 147
五、上机题 147

第 7 章 分支限界法 148

7.1 概述 148
7.1.1 什么是分支限界法 148
7.1.2 分支限界的基本思想 148
7.2 复杂的有限期作业调度问题 149
7.3 贷郎担问题的分支限界法 151
7.4 其他分支限界问题 154
7.4.1 布线问题 154
7.4.2 0/1 背包问题 157
7.4.3 单源最短路径的分支限界法 160
7.5 分支限界法与回溯法的比较 162
小结 163
习题七 163
一、选择题 163
二、填空题 163
三、简答题 163
四、计算题 164
五、上机题 164

第 8 章 概率算法概述 165

8.1 概率算法概述 165
8.1.1 什么是概率算法 165
8.1.2 概率算法的基本思想 166
8.2 数值概率算法 167
8.3 蒙特卡罗算法 168
8.4 其他概率算法 169
8.4.1 舍伍德算法 169
8.4.2 拉斯维加斯算法 170
小结 173
习题八 173
一、选择题 173
二、填空题 173
三、简答题 173
四、计算题 173
五、上机题 173

第 9 章 NP 问题 174

9.1 NP 问题概述 174
9.2 P 类与 NP 类问题 175
9.2.1 非确定性图灵机 175
9.2.2 P 类与 NP 类语言 176
9.2.3 多项式时间验证 177
9.3 NP 完全问题 178
9.3.1 多项式时间变换 178
9.3.2 Cook 定理 179
9.4 一些典型的 NP 完全问题 182
9.4.1 合取范式的可满足性问题 CNF - SAT 183
9.4.2 三元合取范式的可满足性问题 3 - SAT 183
9.4.3 团问题 CLIQUE 184
9.4.4 顶点覆盖问题 VERTEX - COVER 185
9.4.5 子集和问题 SUBSET - SUM 186
9.4.6 哈密顿回路问题 HAM - CYCLE 188
9.4.7 旅行售货员问题 TSP 191
小结 192
习题九 192
一、选择题 192
二、填空题 192
三、简答题 192
四、计算题 192
五、上机题 193

第 10 章 近似算法 194

10.1 近似算法概述 194
10.1.1 什么是近似算法 194
10.1.2 近似算法的基本思想及性能 194

10.2 顶点覆盖问题的近似算法 196
10.3 集合覆盖问题的近似算法 198
10.4 子集合问题的近似算法 199
10.4.1 子集和问题的指数时间算法 200
10.4.2 子集合问题的完全多项式时间近似格式 200
小结 202
习题十 202
一、选择题 202
二、填空题 202
三、简答题 203
四、计算题 203
五、上机题 203
第 11 章 新型算法 204
11.1 加密算法概述 204
11.2 初等数论 205
11.3 DES 以及 3 重 DES 算法 208
11.3.1 DES 算法 208
11.3.2 三重 DES 算法 210
11.4 AES 算法 213
11.4.1 Rijndael 加密算法描述 214
11.4.2 Rijndael 解密算法描述 215
11.5 RSA 算法 215
小结 216
习题十一 216
一、选择题 216
二、填空题 217
三、简答题 217
四、计算题 217
五、上机题 217
第 12 章 并行算法 218
12.1 并行算法概述 218
12.2 并行计算机 218
12.2.1 并行计算机分类 218
12.2.2 并行计算机的处理机互连方式 222
12.2.3 并行计算模型 224
12.3 并行算法 228
12.3.1 数据并行模型 229
12.3.2 消息传递模型 230
12.3.3 共享变量模型 231
12.4 并行算法编程实现 231
12.4.1 枚举排序 232
12.4.2 快速排序 233
12.4.3 并行正则采样排序 234
小结 235
习题十二 236
一、选择题 236
二、填空题 236
三、简答题 236
四、计算题 236
五、上机题 236
第 13 章 上机实训 237
13.1 递归技术应用 237
13.2 运用贪心算法求解实际问题 239
13.2.1 套汇问题 239
13.2.2 汽车加油问题 239
13.3 动态规划法应用 240
13.3.1 最好费用购物 240
13.3.2 租用游艇问题 242
13.4 回溯法应用 242
13.4.1 重排九宫问题 243
13.4.2 智力拼图问题 247
13.5 分支限界法应用 251
13.5.1 0/1 问题的栈式分支限界法 251
13.5.2 用最大堆存储活结点的优先队列式分支限界法 253
小结 255
参考文献 256

第 1 章　算法概述

算法是计算机学科的一个重要分支，计算机科学的基础，更是程序设计的基石。学习算法，一方面需要学习求解计算领域中典型问题的各种有效算法，在遇到问题时能灵活地应用所掌握的方法技巧；另一方面还要学习设计新算法和分析算法性能的方法，当没有现成可用的算法时，能够创造出有效的问题求解方法。

本章学习目标：

（1）掌握算法的概念；

（2）掌握算法的复杂度分析与效率分析；

（3）了解算法的设计与分析步骤；

（4）了解算法描述语言 C 语言的基本使用。

1.1　算法概念

从算法的概念着手，本节主要讲述为什么学习算法以及抽象表达算法机制。

1.1.1　什么是算法

对于计算机科学来说，算法（Algorithm）的概念是至关重要的。例如在一个大型软件系统的开发中，设计出有效的算法将起决定性的作用。通俗地讲，算法是指解决问题的一种方法或一个过程，如图 1-1 所示。

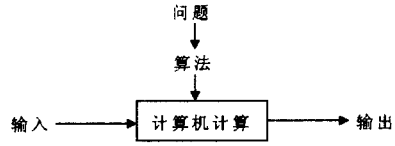


图 1-1　运用算法解决问题的过程

从这个角度来讲，算法的研究由来已久。中国的珠算口诀可以说是非常典型的算法，它将复杂的计算（如除法）描述为一系列的算珠拨动操作。还有，计算两个整数的最大公约数的辗转相除法（欧几里德算法）也是算法早期研究的最重要成果。

更严格地讲，算法是由若干条指令组成的有穷序列，且满足下述几条性质：

（1）输入（input）：有零个或多个由外部提供的量作为算法的输入。

（2）输出（output）：算法产生至少一个量作为输出。

（3）确定性（definiteness）：组成算法的每条指令是清晰的，不能有二义性，是无歧义的。

（4）可行性（effectiveness）：算法中每条指令都有明确的定义，是可行的，可在有限的时间内做完。

（5）有穷性（finiteness）：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的，即算法必须总能在执行有限步骤之后终止。

所有算法都必须具有以上 5 个性质。算法的输入是一个算法在开始前所需的最初的量，

这些输入取自特定的值域。算法可以没有输入，但算法至少应产生一个输出，否则算法便失去了它存在的意义。算法是一组指令序列。一方面，每条指令的作用必须是明确、无歧义的。在算法中不允许出现诸如“计算 5+3 或计算 7－3”这样让计算机自己选择的指令。另一方面，算法的每条指令必须是可行的。对一个计算机算法而言，可行性要求一条算法指令应当最终能够由执行有限条的计算机指令来实现。例如，一般的整数算术运算是可行的，但如果像 3÷7 这样的值就需要由无穷的十进制展开的实数表示，就不是可行的。因此，概括地说，算法是由一系列明确定义的基本指令序列所描述的，求解特定问题的过程。它能够对合法的输入，在有限时间内产生所要求的输出。如果取消了有穷性的限制，则只能称为计算机实现的计算过程（computational procedure）。当一个算法使用计算机程序设计语言描述时，就是程序（program）。

算法必须可终止，而计算机程序并没有这一限制。例如，一个操作系统是一个程序，却不是一个算法，一旦运行，只要计算机不关闭，操作系统程序就不会终止运行。所以，操作系统程序是使用计算机语言描述的一个计算过程。算法与程序（Program）不同。程序是算法用某种程序设计语言的具体实现。程序可以不满足算法的性质（5）。例如操作系统，它是一个在无限循环中执行的程序，因而不是一个算法。然而我们可把操作系统的各种任务看成是一些单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

算法是计算机学科的一个重要分支，计算机科学的基础，更是程序的基石。学习算法，一方面需要学习求解计算领域中典型问题的各种有效算法，在遇到问题时能灵活地应用所掌握的方法技巧；另一方面还要学习设计新算法和分析算法性能的方法，当没有现成可用的算法时，能够创造出有效的问题求解方法。

1.1.2 抽象表达算法机制

算法层出不穷，变化万千，其对象数据和结果数据名目繁多，不胜枚举。最基本的有布尔值数据、字符数据、整数和实数等；稍复杂的有向量、矩阵、记录等；更复杂的有集合、树和图，还有声音、图形、图像等数据。

算法的运算种类五花八门、多姿多彩。最基本的有赋值运算、算术运算、逻辑运算和关系运算等；稍复杂的有算术表达式和逻辑表达式等；更复杂的有函数值计算、向量运算、矩阵运算、集合运算，以及表、栈、队列、树和图上的运算等；此外，还可能有以上列举的运算的复合和嵌套。

高级程序设计语言在数据、运算和控制三方面的表达中引入许多使之十分接近算法语言的概念和工具，具有抽象表达算法的能力。高级程序设计语言的主要好处是：

（1）高级语言更接近算法语言，易学、易掌握，一般工程技术人员只需要几周时间的培训就可以胜任程序员的工作；

（2）高级语言为程序员提供了结构化程序设计的环境和工具，使得设计出来的程序可读性好，可维护性强，可靠性高；

（3）高级语言不依赖于机器语言，与具体的计算机硬件关系不大，因而所写出来的程序可移植性好、重用率高；

（4）把复杂琐碎的事务交给编译程序，所以自动化程度高，开发周期短，程序员可

以集中时间和精力从事更重要的创造性劳动，提高程序质量。

算法从非形式的自然语言表达形式转换为形式化的高级语言是一个复杂的过程，仍然要做很多繁杂琐碎的事情，因而需要进一步抽象。

对于一个具体的数学问题，想设计它的算法，应该先选用一个适合此问题的数据模型。接下来，要搞清楚此问题的数据模型在已知条件下的初态和要求的终态，以及这两个状态之间的隐含关系。然后确定从数据模型的已知初态到达要求的终态所需的运算步骤。这些运算步骤就构成了求解该问题的具体算法。

按照自顶向下逐步分解求精的原则，在确定运算步骤的时候，应该考虑算法顶层的运算步骤，然后再考虑底层的运算步骤。定义在数据模型级上的运算步骤被定义成顶层运算步骤，或称之为宏观步骤。它们是组成算法的主干部分，而表达这部分算法通常使用非形式化的自然语言。这其中所涉及的数据是数据模型中的变量，暂不关心其数据结构；而所涉及的运算将数据模型中数据变量或者作为运算对象，或作为运算结果，甚至是二者兼而为之，这些运算被简称为定义在数据模型上的运算。由于暂时不需要关心变量的数据结构，这些运算都带有抽象的性质，而不含运算细节。顶层抽象运算的具体实现被称作底层运算步骤。它们是依赖于数据模型结构及其具体表示。所以，数据模型的具体表示和定义在该数据模型上的运算的具体实现是底层运算步骤的两个主要部分。将顶层运算步骤称为宏观步骤，那么底层运算就可以被认为是微观运算。二者的关系是：底层运算是顶层运算的细化，底层运算为顶层运算服务；顶层决定底层，顶层又是通过底层调用实现的。为了将顶层算法与底层算法相隔开，使二者在设计时独立，必须抽象化二者的接口。限制底层只能通过接口为顶层服务，顶层也只能通过接口调用底层运算。此接口就是抽象数据类型（Abstract Data Types，ADT）。

算法设计的重要概念之一就是抽象数据类型。实际上，它是算法的一个数据模型和定义在该模型上的作为算法构件的一组运算。这样定义抽象数据类型就明确地把数据模型与该模型上的运算紧密地联系起来。事实确实如此，一方面，数据模型上的运算依赖于数据模型的具体表示，数据模型上的运算以数据模型中的数据变量为运算对象和/或运算结果；另一方面，确定了数据模型的具体表示，也确定了数据模型上运算的具体实现，运算的效率就随之而定了。那么，问题是如何选择数据模型的具体表示使该模型上各种运算效率都尽可能高呢？显然，针对不同的运算组，为达到使该运算组中所有运算的效率都尽可能高的目的，其相应的数据模型的具体表示将有所不同。在这个前提下，数据模型的具体表示又要依赖于数据模型上定义的运算。特别地，当不同运算的效率互相牵制时，事先还须将所有的运算使用频度做相应的排序，使所选择的数据模型的具体表示首先保证使用频度较高的运算效率较高。抽象数据类型概念产生的背景和依据就是数据模型的定义在该模型上的运算之间存在的这种密不可分的联系。

使用抽象数据类型带给算法设计的好处主要有：

（1）算法顶层设计与底层实现分离，使得在进行顶层设计时不考虑它所用到的数据，运算表示和实现；反过来，在表示数据和实现底层运算时，只要定义清楚抽象数据类型而不必考虑在什么场合引用它。这样做就使算法设计的复杂性降低了，同时条理性也随之增强，既有助于迅速开发出程序原型，又使开发过程少出差错，程序可靠性提高。

（2）算法设计与数据结构设计相隔开，允许数据结构自由地选择，从其中做比较，

优化算法的效率。

（3）数据模型和该模型上的运算统一在抽象数据类型中，反映它们之间内在的互相依赖和互相制约的关系，便于空间和时间耗费的折衷，灵活地满足用户要求。

（4）由于顶层设计和底层实现局部化，在设计中出现的差错问题也是局部的，因而容易查找以及容易纠正，在设计中经常要做的增、删、改也都是局部的，因而也都容易进行。因此，用抽象数据类型表述的算法具有很好的可维护性。

（5）算法自然呈现模块化，抽象数据类型的表示和实现可以封装，便于移植和重用。

（6）为自顶向下逐步求精和模块化提供了有效途径和工具。

（7）算法结构清晰，层次分明，便于算法正确的证明和复杂性的分析。

1.2 算法的复杂度

解决同一个问题可以有多个算法，如何确定某个算法是最优的呢？这就要看它的复杂性，一个好的算法不仅要具有正确性(correctness)、简明性(simplicity)、高效性(efficiency)，还要具备最优性（ optimality)。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，所需资源越多，我们就说该算法的复杂性越高；反之，则该算法的复杂性越低。最重要的计算机资源是时间和空间（即存储器）资源。因此，算法的复杂性有时间复杂性和空间复杂性之分。一般情况，要选择算法复杂性低的算法。

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，是在选用算法时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。

本节主要讲述算法的复杂度概念、算法三性态以及对算法复杂度的分析。

1.2.1 算法三性态

通常，分析算法要从它的三个性态考虑，即最好性态、最坏性态和平均性态。对于某些算法，即使问题规模相同，如果输入数据不同，其时间开销也不同。例如，现在要从一个 n 元的一维数组中找出一个给定的 K（假设该数组中有且仅有一个元素值为 K)。顺序检索法将从第一个元素开始，依次检索每一个元素，直到找到 K 为止。一旦找到了 K，算法也就完成了。要找到给定的 K，必须检查每一个元素的值。这样，顺序检索法的时间开销可能在很大的一个范围内浮动。数组中的第一个元素可能恰恰就是 K，于是只要检索一个元素就行了。在这种情况下，运行时间很短。这叫做算法的最好情况（ best case ）——顺序检索算法不可能执行比检索一个元素更少的操作。另一种情况，如果数组的最后一个元素是 K，运行时间就会相当长，因为这个算法要检查所有的 n 个元素。这是算法的最坏情况（ worst case ）——该算法不可能检索 n 个以上的元素。如果用一个程序来实现顺序检索法并用该程序对许多不同的 n 元数组检索，或者在同一个数组中检索不同的 K 值，就会发现，平均检索到整个数组的一半就能找到 K。也就是说，这种算法平均要检索 n/2 个元素，我们称之为算法的平均情况（ average case ）时间代价。

分析一种算法时，应该研究最好、最坏还是平均情况呢？一般来说，我们对最好情况没有多大兴趣，因为它发生的概率太小，而且对于条件的考虑太乐观了。换言之，最好情

况不能作为算法性能的代表。不过，也有一小部分情况，最好情况分析是有用的——尤其是当最好情况出现概率较大的时候，可以用最好情况运行时间来分析该算法，非常迅速而有效。

那么最坏情况呢？分析最坏情况有一个好处：它能让你知道算法至少能运行得多快。这一点在实时系统中尤其重要，例如空运处理系统。在这个系统中，一个“绝大部分”情况下能管理 n 架飞机的算法，如果它不能在规定的时间内管理来自于同一方向的 n 架飞机，那么它是不能被接受的。那么，最坏情况复杂度 T(n)可表示如下：

$$T(n)=\max_{I\in D_n} t(I)$$

即 T(n)是算法在任何规模为 n 输入时所执行的基本运算的最大次数。可以看出，T(n)的计算较平均复杂度 $T_{avg}(n)$的确定往往方便得多。

在另外一些情况下，特别是程序要对许多不同的输入运行多次时，最坏情况分析就不适合用来衡量一种算法的性能了。通常人们会更希望知道平均情况的时间代价，也就是说，当输入规模为 n 时算法的“典型”表现。可惜，平均情况分析并不总是可行的。首先，它要求人们清楚数据是如何分布的。例如，上面提到过顺序检索算法平均情况下要检查数组中一半的元素。但是这是基于 K 在数组中每个位置出现概率相等的假设之上的。如果这个假设不成立，那么算法的平均情况就不一定是检查一半的元素。

总之，在实时系统中，我们比较关注最坏情况算法分析。在其他情况下，通常考虑平均情况，只要知道计算平均情况所需要的输入数据的分布即可；否则，就只能求助于最坏情况分析了。

1.2.2 算法复杂度

算法的复杂度是算法运行所需要的计算机资源的量，需要时间资源的量被称为时间复杂度（time complexity），需要的空间资源的量被称为空间复杂度（space complexity）。这个量应该集中反映算法的效率，并从运行该算法的实际计算机中抽象出来。换句话说，这个量应该是只依赖于要解的问题的规模、算法的输入和算法本身的函数。如果分别用 N、I 和 A 表示算法要解的问题的规模、算法的输入和算法本身，而且用 C 表示复杂度（complexity），那么，应该有 C = F(N，I，A)，其中 F(N，I，A)是一个由 N、I 和 A 确定的三元函数。如果把时间复杂度和空间复杂度分开，并分别用 T 和 S 来表示，应该有：T = T(N，I，A)和 S = S(N，I，A)。通常，让 A 隐含在复杂性函数名当中，因而将 T 和 S 分别简写为 T = T(N，I)和 S = S(N，I)。

1. 时间复杂度

现在研究时间复杂度，所面临的问题是如何将复杂性函数具体化，即对于给定的 N、I 和 A，如何导出 T(N，I)和 S(N，I)的数学表达式，来给出计算 T(N，I)和 S(N，I)的法则。下面以 T(N，I)为例，将复杂性函数具体化。

根据 T(N，I)的概念，它应该是算法在一台抽象的计算机上运行所需要的时间。设此抽象的计算机所提供的基本运算有 k 种，它们分别记为 O_1，O_2，…，O_k。又设每执行一次这些基本运算所需要时间分别为 t_1，t_2，…，t_k。对于给定的算法 A，设经统计，用到基本运算 O_i 的次数为 e_i，i = 1，2，…，k。很清楚，对于每一个 i，$1\leqslant i\leqslant k$，e_i 是 N 和 I 的函

数，即 $e_i = e_i(N, I)$。那么有：

$$T(N,I)=\sum_{i=1}^{k} t_i e_i(N,I)$$

其中，$t_i(i=1, 2, \cdots, k)$是与 N 和 I 无关的常数。

显然，不可能对规模 N 的每一种合法的输入 I 都去统计 $e_i(N, I)$，$i=1, 2, \cdots, k$。因此 T(N, I)的表达式还要进一步简化，或者说，只能在规模为 N 的某些或某类有代表性的合法输入中统计相应的 e_i，$i=1, 2, \cdots, k$，评价其时间复杂性。

考虑时间复杂性的三性态，并分别记最坏情况、最好情况和平均情况下的时间复杂度为 Tmax(N)、Tmin(N)和 Tavg(N)。在数学上有：

$$T\max(N)=\max_{I\in D_N} T(N,I)=\max_{I\in D_N}\sum_{i=1}^{k} t_i e_i(N,I)=\sum_{i=1}^{k} t_i e_i(N,I^*)=T(N,I^*)$$

$$T\min(N)=\min_{I\in D_N} T(N,I)=\min_{I\in D_N}\sum_{i=1}^{k} t_i e_i(N,I)=\sum_{i=1}^{k} t_i e_i(N,\bar{I})=T(N,\bar{I})$$

$$T\text{avsg}(N)=\sum_{I\in D_N} P(I)T(N,I)=\sum_{I\in D_N} P(I)\sum_{i=1}^{k} t_i e_i(N,I)$$

式中，D_N 是规模为 N 的合法输入的集合；I*是 D_N 中一个使 T(N, I*)达到 Tmax(N)的合法输入；$\bar{I}$ 是 D_N 中一个使 T(N, $\bar{I}$)达到 Tmin(N)的合法输入；而 P(I)是在算法的应用中出现输入 I 的概率。

以上三种情况下的时间复杂性从不同角度来反映算法的效率，各有其局限性，也各有各的用处。实践表明可操作性最好且最有实际价值的是最坏情况下的时间复杂性。

显然，对于算法工作量的度量，应该有助于认清一个算法的好坏，有助于比较同一问题的各种算法之优劣，以便可以确定一种算法的效率是否比另一些更高。当然，如果能够选择算法的实际执行时间，这种比较将非常方便。然而，这种时间却受到诸多因素的影响，如程序所依赖的算法、问题的规模和输入的数据、计算机系统性能及所用的机器语言等等。所以，用实际执行时间作为标准并不客观。因此，以考察算法所做的基本运算的数目来代替其实际执行时间，可以说明这种做法是合理的。

另外，一个算法往往由各种各样的运算构成，如加、减、乘、除和两个元素比较等等。对于不同的运算，计算机执行每种运算的时间也不相同，有的少些，有的则多些。这样，就需要在算法中找出一种（有时也可多于一种）运算，使该算法的运行时间与此种运算的次数相匹配。也就是说，为了分析一个算法，需要将一种对于所研究的问题来说是“基本的”运算分离出来，忽略其他运算或细节（这些其余的运算细节可称为“薄记”工作），而只计算算法所执行的基本运算的次数。

对于一个问题，通常可以选择一种基本运算。所谓基本运算就是可以用来衡量算法运行时间的主要运算。但有时会发现，还有一些其他的运算，其数目也是巨大而不可忽视的。例如，计算两矩阵的积时，如果有一种算法，只做了少量乘法，而做了大量加法，那么，把基本运算定为乘法和加法才是合理的。这样，尽管乘法和加法的运算时间不同，但总体不至于有太多的偏差，否则有可能丢失一些关于两种算法的相对优点的有用信息（例如可以设计做较多的加法而做较少的乘法来改进算法）。

只要合理地选择基本运算，是算法执行的时间大致和基本运算次数成比例，就有了一