北京市高等教育精品教材立项项目

高等学校规划教材

算法设计与问题求解
——编程实践

◎ 李清勇　编著

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

TP301.6/149

2013

北京市高等教育精品教材立项项目

高等学校规划教材

算法设计与问题求解
——编程实践

李清勇　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书是北京市精品教材立项项目，是大学生创新实践课程——“算法设计与实践”课程教材。本书以问题求解为目标，以高级程序设计语言 C/C++为工具，讨论怎样综合运用算法（包括数据结构）知识去分析问题和解决问题。问题驱动、高级语言程序设计、数据结构以及算法设计与分析知识交叉融合是本书的特点。内容包括问题求解与算法分析概述、基本数据结构、高级数据结构、枚举算法、递归与分治、动态规划、贪心算法、搜索算法、图算法、算法分析的实用公式、在线程序评测系统简介等。教材配套有适合理论教学使用的电子课件以及适合实践教学使用的“在线程序评测系统”。

本书适合高等学校计算机、信息与计算科学、信息管理等专业师生使用，也可作为 ACM 程序设计竞赛培训人员的参考书。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

算法设计与问题求解：编程实践/李清勇编著. —北京：电子工业出版社，2013.6
高等学校规划教材
ISBN 978-7-121-20327-5

Ⅰ. ①算… Ⅱ. ①李… Ⅲ. ①电子计算机-算法设计-高等学校-教材 Ⅳ. ①TP301.6

中国版本图书馆 CIP 数据核字（2013）第 094752 号

策划编辑：袁 玺
责任编辑：章海涛 文字编辑：袁 玺
印 刷：北京市李史山胶印厂
装 订：北京市李史山胶印厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：15.5 字数：396.8 千字
印 次：2013 年 6 月第 1 次印刷
定 价：35.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前言

2006年3月，美国卡内基一梅隆大学计算机科学系主任周以真(Jeannette M. Wing)教授在美国计算机权威期刊《Communications of the ACM》上首先提出了计算思维(Computational Thinking)的概念。周教授认为：计算思维是运用计算机科学的基础概念进行问题求解、系统设计以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。计算思维是每个人的基本技能，不仅仅属于计算机科学家。我们应当使每个学生在培养解析能力时不仅掌握阅读、写作和算术(Reading, wRiting and aRithmetic, 3R)，还要学会计算思维。正如印刷出版促进了3R的普及，计算和计算机也以类似的正反馈促进了计算思维的传播，计算机逐渐成为了当今问题求解的最重要工具。

1. 计算机与问题求解

在20世纪40年代，为了求解军事领域复杂的炮弹弹道计算问题，科学家发明了第一台电子计算机“埃尼阿克”(Electronic Numerical Integrator And Computer, ENIAC)。随着计算机计算能力的增强，计算机被广泛应用到了社会生活的各个领域。大到宇宙探测、基因图谱绘制，小到日常工作、生活娱乐，无不需要计算机的支持。

作为“问题求解的一个有力工具”，计算机尽管没有思维，只能机械地执行指令，但它运算速度快、存储容量大、计算精度高。如果能够设计有效的算法和程序，充分利用这些优点，计算机就能成为问题求解的一个利器。

运算速度快是计算机最重要的特点之一。很多问题尽管比较复杂，但仍然存在求解的方法，只是这些方法往往计算量比较大，计算过程较为烦琐，人们难以在可以接受的时间内手工求解。

【例】用1，2，3，4，5，6，7，8，9这9个数字拼成一个9位数，每个数字使用一次且仅用一次，要求得到的9位数的前3位、中间3位和最后3位构成的3位数的比值为1:2:3。例如192384576就是一个符合该要求的数，因为192:384:576=1:2:3。

对于这样的问题，很容易想到的一个求解方案是：列举所有可能的9位数123456789，…，987654321，并逐个验证是否符合比值要求。理论上，这是一个可行的办法，可是几乎没有人愿意这样做。因为这样的9位数总共有9!=362880个，即使每秒验证一个数(对于人工验证，这已经是很快的速度了)，也需要100多个小时。

但是，计算机实现同样的“笨方法”效果就大不一样。考虑用一个数组d保存9位数的各位数字，x，y，z分别代表前3位数、中间3位数和最后3位数，用STL中的函数next_permutation计算9个数字的下一个排列。当某个排列(即一个9位数)满足比例要求$x:y:z=1:2:3$时，则输出该9位数。下面是解决此问题的C++代码，在普通的PC上运行该程序，需要的时间还不到0.1秒。

```
void NineNumber() {
    int x , y , z;
    int d[9]= {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9};
    do{
        x=d [0] * 100+d [1] * 10+d [2];
        y=d [3] * 100+d [4] * 10+d [5];
        z=d [6] * 100+d [7] * 10+d [8];
        if(y == x * 2 && z == x * 3)
            cout << x << y << z << endl ;
    } while (next_permutation(d , d+9)); // STL 中的函数，得到下一个排列
}
```

存储容量大是计算机的另一个重要特点。人们很容易记住 10 以内的两个数的乘积，也就是小学数学中的“九九乘法表”。但如果要求人们记住 100 以内的任意两个数的乘积，普通人可能会觉得“记忆”不够。但对于计算机来说，这真是“小菜一碟”，一个 100×100 的二维数组就可以把“百百乘法表”保存下来。

人们常说的内存、硬盘、光盘、U 盘等都是存储器，可以通俗地理解为计算机的记忆部件。容量的基本单位是**字节**(Byte)，记为 B。其他单位有 KB(1KB＝1024B)、MB(1MB＝1024KB)、GB(1GB＝1024MB)等。容量越大，可以存储的数据就越多，其“记忆力”就越强。“百百乘法表”如果用一个 100×100 的二维 int 型整数(假设一个 int 型整数占 2 字节)数组保存，它仅仅需要占用 20 000 字节(少于 20KB)的空间。相比现在计算机动辄数 GB 的内存容量来说，“百百乘法表”的存储开销微不足道。

需要特别指出的是，运算速度快、存储容量大仅仅是计算机硬件系统的两个突出特点。在实际问题求解过程中，只有硬件平台还远远不够，人们需要针对问题设计不同的算法，并把算法转化为计算机可以运行的程序。

2. 计算机问题求解的知识体系

计算机问题求解的本质是把**特定领域中特定问题的求解过程转换为计算机可以执行的程序**。在这个转换过程中，除了必要的专业知识外，问题求解者还需要掌握计算机算法设计方面的知识，主要包括高级程序设计语言、数据结构和算法。这些知识构成了计算机问题求解的核心知识体系。

在计算机教学体系中，高级程序设计语言、数据结构、算法是相互承接的课程系列。从计算机问题求解的角度看，这三门课程的知识相互交叉、相互支撑。高级程序设计语言和数据结构是算法设计的基础，高效的算法和数据结构需要用某种高级程序设计语言来实现；一个好程序不仅需要“编程小技巧”，更需要合理的数据结构和高效的算法。

程序设计语言是问题求解的基本工具。随着计算机技术的发展，程序设计语言经历了一个从低级程序设计语言到高级程序设计语言的发展历程。机器语言和汇编语言等面向特定的体系结构和指令系统，在计算机发展的早期应用较多。随着形式语言理论、编译技术的发展，与目标机器无关的高级程序设计语言(如 C/C++，Java 等)逐渐成为程序设计的主流。本书约定的程序设计语言是 C/C++。需要注意的是，程序设计语言并不等于程序设计，程序设计的

试读结束：需要全本请在线购买： www.ertongbook.com

目的是表达程序设计者的思想，按照计算机所能理解的方式描述需要让计算机完成的工作，而程序设计语言只是表达这种思想的工具。程序设计的关键之处在于明确数据在计算机中的表达形式，以及确定如何将输入转化为输出的一系列计算步骤，而这些都需要数据结构和算法理论的指导。

数据结构是问题求解的基础要素。数据是信息的载体，无论是待求解问题的输入/输出，还是问题求解过程中产生的中间量；无论是简单的量，比如单个数值，还是复杂的对象，它们在计算机中都以数据的形式进行存储。在问题求解时，为满足数据存储的结构化要求并提高程序执行效率，人们首先面临的问题是怎样合理地组织、存储和加工这些数据。常用的数据结构有线性表、栈、队列、树、二叉树、图、哈希表(散列表)等。数据结构的设计和应用不是一个教条化的生搬硬套过程，同一个问题也许可以运用不同的数据结构求解，而且它们求解的效率往往不尽相同。另外，有些问题可能没有现成的数据结构直接套用，需要人们综合运用基本数据结构组合成新的数据结构。无论是已有数据结构的选择还是新数据结构的设计，人们都需要应用算法设计与分析方面的知识。

算法设计是问题求解的关键要素。简单地说，算法可以理解为把将问题输入转化为问题答案的一系列计算步骤。算法必须满足正确性与复杂性要求。首先，算法执行结果必须正确，它能正确无误地把每一个问题实例的正确答案求解出来。其次，算法的复杂度要适中。计算机系统的资源(包括运行时间和存储空间)是有限的，因此算法必须在有限的资源条件下正确地求解问题。同样的问题，某些算法执行结果可能不正确，某些算法执行结果则正确无误。即使执行结果都正确的不同算法，它们的执行效率可能也不尽相同，如有些算法需要几个小时，甚至几天，有些算法却仅仅需要几秒钟或几分钟。算法设计是一个灵活的、创造性的过程，甚至可以认为是一个艺术创造过程。有些算法是现实生活中人们解决问题时所用办法的升华和抽象；有些算法是数学理论和数学模型的体现和具体化。人们需要掌握经典的算法思想及其应用技巧，也要学会怎样针对特定问题设计和创造新算法。

3. 本书的内容和结构

本书是一本讲述怎样综合运用算法设计理论和技术进行问题求解的实践教材，主要讲述算法设计原理和方法，对运用算法求解问题时涉及的C/C++程序设计细节，尤其是影响算法准确性和复杂性的编程要点和技巧也进行了详细阐述。数据结构往往是算法设计和实现的基础，特别是一些高级数据结构，其本身就体现了很强的算法思维，因此本书不仅仅单独设立一章讲述数据结构，在讨论具体算法时也会交叉讨论相应的数据结构知识。本书包括7章，组织如下：

第1章介绍问题求解和算法的基本概念，然后着重阐述了算法复杂度分析的基本理论和方法。

第2章介绍程序设计和数据结构相关内容，程序设计和数据结构是算法设计的重要支撑，本章重点介绍了程序设计的三个盲点，以及常用的基本数据结构及其用法。

第3章介绍枚举算法。“大道至简”，枚举算法是一种最朴素最简单的算法思想，但在具有卓越运算速度的计算机系统中，它却是常常被忽视的问题求解利器。本章重点阐述了怎样直接和间接运用枚举算法求解问题。

第4章介绍递归和分治算法。“凡治众如治寡，分数是也”，分治策略是分析和解决复杂问

题最常用的策略之一。本章根据分治算法的求解步骤，将分治策略归纳为三类，并结合具体实例阐述每类策略的设计思想、适用范围及实现要点。

第5章介绍动态规划算法。动态规划算法是最具有创造性的一种算法，归约、分治等思维方法都在动态规划算法框架中得到了很好的体现。本章重点讨论基于“划分”和“约简”策略的动态规划算法原理和运用技巧。

第6章介绍贪心算法。这种类似于“瞎子爬山”的策略，如果运用适当，能够快速地产生最优解。本章给出了一些典型的贪心算法问题，并探讨了贪心算法的适用范围。

第7章介绍搜索算法。搜索是求解一些难解问题的常用策略，它把问题求解转换为状态空间图中的路径探索过程，究其本质，搜索是一种枚举和优化策略的综合算法。“运用之妙，存乎一心”，本章以典型问题为例阐述5种经典搜索策略的原理、适用范围以及实现要点。

为便于教学和读者自学，本教材提供有适用于理论教学的课件以及实践学习的配套平台——“北京交通大学在线程序评测系统”(http://acm.bjtu.edu.cn/OnlineJudge，简称BOJ)。BOJ是一个公益性质的计算机问题求解实践平台，也是本书的配套网站。本书的例题和习题都以专题的形式加入BOJ题库中，读者在学习本书时，可登录该系统进行编程求解和自我评测。

目 录

第 1 章 计算机问题求解概述 …… 1

1.1 问题与问题实例 …… 1

1.2 计算机问题求解周期 …… 2

1.3 算法与程序 …… 5

1.4 算法复杂性分析 …… 5

1.4.1 空间复杂性 …… 5

1.4.2 时间复杂性 …… 7

1.4.2.1 时间复杂性的表示 …… 7

1.4.2.2 渐近时间复杂性及其阶 …… 8

1.4.2.3 时间复杂性渐近阶的意义 …… 11

1.4.2.4 算法时间复杂性分析 …… 13

习题 1 …… 14

第 2 章 程序设计语言与数据结构 …… 15

2.1 程序设计语言的“盲点” …… 15

2.1.1 long 不够长 …… 16

2.1.1.1 数据类型的值域 …… 16

2.1.1.2 大整数相加算法 …… 17

2.1.2 double 不够准 …… 18

2.1.2.1 浮点数的存储格式 …… 19

2.1.2.2 浮点数的有效数字 …… 21

2.1.2.3 高精度浮点数处理实例 …… 22

2.1.3 递归不够快 …… 24

2.2 基本数据结构 …… 25

2.2.1 线性表 …… 25

2.2.1.1 线性表的顺序存储结构 …… 26

2.2.1.2 线性表的链式存储结构 …… 27

2.2.2 栈和队列 …… 29

2.2.2.1 栈 …… 29

2.2.2.2 队列 …… 33

2.2.3 树和二叉树 …… 37

2.2.3.1 树 …… 37

2.2.3.2 二叉树 …… 40

2.2.4 优先队列和堆 …… 45

2.2.4.1 优先队列 …………………………………………………………… 45
2.2.4.2 二叉堆 …………………………………………………………… 45
2.2.5 图 ………………………………………………………………………… 46
2.2.5.1 邻接矩阵 ………………………………………………………… 46
2.2.5.2 邻接表 …………………………………………………………… 47
2.3 标准模板库………………………………………………………………… 48
2.3.1 模板的基本概念………………………………………………………… 48
2.3.2 标准模板库概述………………………………………………………… 51
2.3.2.1 算法 ……………………………………………………………… 51
2.3.2.2 容器 ……………………………………………………………… 51
2.3.2.3 迭代器 …………………………………………………………… 52
2.3.3 标准模板库应用………………………………………………………… 53
2.3.3.1 向量(vector) …………………………………………………… 53
2.3.3.2 集合和多重集合(set 和 multiset) …………………………… 54
2.3.3.3 映射和多重映射(map 和 multimap) ………………………… 55
2.3.3.4 堆(heap) ………………………………………………………… 57
2.3.3.5 排序算法 ………………………………………………………… 59
习题 2 ………………………………………………………………………… 65
第 3 章 枚举算法………………………………………………………………… 70
3.1 枚举的基本思想…………………………………………………………… 70
3.2 模糊数字…………………………………………………………………… 71
3.3 m 钱买 n 鸡…………………………………………………………… 73
3.4 真假银币…………………………………………………………………… 75
习题 3 ………………………………………………………………………… 78
第 4 章 递归与分治……………………………………………………………… 84
4.1 递归程序…………………………………………………………………… 84
4.2 分治策略的基本原理……………………………………………………… 88
4.3 合并排序…………………………………………………………………… 90
4.4 逆序对问题………………………………………………………………… 94
4.5 快速排序…………………………………………………………………… 97
4.6 最接近点对问题 ………………………………………………………… 100
4.7 指数运算 ………………………………………………………………… 106
4.8 二分查找 ………………………………………………………………… 107
习题 4 ………………………………………………………………………… 109
第 5 章 动态规划 ……………………………………………………………… 117
5.1 动态规划的基本思想 …………………………………………………… 117
5.1.1 动态规划的基本要素 ……………………………………………… 119
5.1.2 动态规划的求解步骤 ……………………………………………… 119

5.2 矩阵连乘 …… 120
5.3 最优二叉搜索树 …… 126
5.4 多段图最短路径 …… 130
5.5 最长公共子序列 …… 134
5.6 0-1 背包问题 …… 137
5.7 最大上升子序列 …… 140
习题 5 …… 143

第 6 章 贪心算法 …… 150
6.1 贪心算法的基本要素 …… 150
6.2 活动安排问题 …… 152
6.3 小数背包问题 …… 156
6.4 最优前缀码 …… 159
6.5 单源最短路径 …… 164
6.6 最小生成树 …… 169
6.6.1 Prim 算法 …… 170
6.6.2 Kruskal 算法 …… 173
6.7 贪心算法与动态规划、分治算法的比较 …… 176
习题 6 …… 176

第 7 章 搜索技术 …… 182
7.1 问题的状态空间表示 …… 182
7.2 深度优先搜索 …… 184
7.3 广度优先搜索 …… 187
7.4 回溯算法 …… 188
7.4.1 回溯算法的基本原理和框架程序 …… 189
7.4.2 装载问题的回溯算法 …… 194
7.4.3 圆排列问题 …… 198
7.5 分支限界 …… 201
7.5.1 分支限界法的基本原理 …… 201
7.5.2 装载问题的分支限界法 …… 203
7.6 启发式搜索 …… 207
7.6.1 启发式搜索基本原理 …… 207
7.6.2 装载问题的启发式搜索 …… 210
习题 7 …… 212

附录 A 复杂性分析的数学基础 …… 220

附录 B 常用 C 语言和 STL 函数 …… 229

附录 C 程序设计竞赛和 OnlineJudge 介绍 …… 234

参考文献 …… 237

第 1 章 计算机问题求解概述

学习要点

- 理解问题和问题实例的概念
- 了解问题求解的基本步骤
- 了解算法空间复杂性分析的方法
- 掌握算法时间复杂性分析的方法

从第一台电子计算机“埃尼阿克”诞生，计算机就成为了复杂问题求解的最重要工具。但是计算机没有思维，不能自主解决问题，而只能机械地执行程序。程序是算法用某种程序设计语言的实现结果，怎样设计正确和高效的算法是计算机问题求解的核心。计算机问题求解周期包括哪些重要的步骤？如果给定的问题存在多个算法，我们怎样评价这些算法的性能？另外，IT 公司的工程师们日常讨论算法性能的语言或者“行话”是什么？

1.1 问题与问题实例

问题是需要人们回答的一般性提问，通常含有若干参数，由问题描述、输入条件以及输出要求等要素组成。

问题实例定义为确定问题描述参数后的一个对象。

一个问题的问题描述和输入条件通常包含若干参数，当给定这些参数的一组赋值后，则可以得到一个问题实例。一个问题可以包含若干个问题实例，问题和问题实例的关系类似于面向对象程序设计语言中类和对象的关系。

【例 1-1】正整数求和问题。

问题描述：计算正整数 a 与 b 的和 c。

输入：正整数 a，b($1 \leqslant a$，$b \leqslant 10000$)。

输出：和 c。

在正整数求和问题中，指定 $a=1$，$b=1$，则构成了一个问题实例 1+1；如果令 $a=1000$，$b=1000$，则构成了另外一个问题实例 1000+1000。

虽然对于正整数求和问题，两个问题实例之间的差别不大，但是，对于有些问题，不同问题实例无论是其描述还是求解的复杂性差别都非常大。

【例 1-2】迷宫问题。

问题描述：在一个 $N \times N$ 的棋盘迷宫中，有些格子能通行(用 1 表示)，有些格子不能通行(用 0 表示)，假定棋盘位置 (1，1) 为入口，(N，N) 为出口，且在棋盘中只能横向或者竖向移动。任意给定一棋盘，试问是否存在从入口到出口的路径。

试读结束：需要全本请在线购买： www.ertongbook.com

输入： 正整数 N 表示棋盘的大小，以及 $N \times N$ 的 0－1 矩阵表示每个棋盘格子的状态。

输出： 1，表示存在路径；0，表示不存在。

在迷宫问题中，当 $N=2$，棋盘布局如图 1-1(a)所示(黑色格子表示 0，白色格子表示 1)时，得到一个问题实例(a)。当 $N=10000$，棋盘布局如图 1-1(b)所示时，得到另外一个问题实例(b)。显然，求解问题实例(b)比求解问题实例(a)更加复杂。在棋盘大小(如 $N=10000$)相同而布局不同的情况下，求解的复杂性也可能不一样。比如在图 1-1 中的实例(b)，(c)，(d)中，采用从入口到出口的广度优先搜索算法(搜索技术参阅第 7 章)，实例(b)所需要的时间会比实例(c)要多，实例(d)所需要的时间也会比实例(c)要多。如果采用从出口到入口的广度优先搜索算法，实例(d)所需要的时间比实例(b)和实例(c)要少得多。

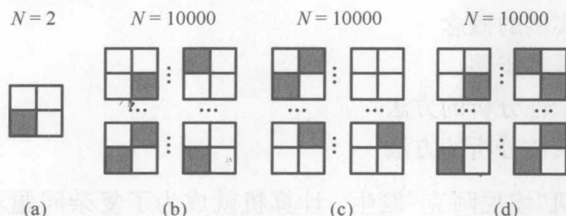

图 1-1　迷宫问题的不同问题实例

值得注意的是，在问题求解时，一个算法能正确而有效地求解某一个问题，严格意义上是指该算法对于该问题的**所有问题实例都能正确而有效地得到答案**，而不是指该算法能正确而有效地求解某一个或者某几个问题实例。

1.2　计算机问题求解周期

在具体讨论问题求解之前，先看看数学家 G·波利亚在 1944 年提出的“怎样解题表”：

……

你以前见过它吗？你是否见过相同的问题而形式稍有不同？

你是否知道与此有关的问题？你是否知道一个可能用得上的定理？

看看未知数！试想出一个具有相同未知数的或相似未知数的熟悉的问题。

这里有一个与你现在的问题有关，且早已解决的问题。

你能不能利用它？你能利用它的结果吗？你能利用它的方法吗？为了能利用它，你是否应该引入某些辅助元素？

你能不能重新叙述这个问题？你能不能用不同的方法重新叙述它？

回到定义去。

如果你不能解决所提出的问题，可先解决一个与此有关的问题。你能不能想出一个更容易着手的有关问题？一个更普遍的问题？一个更特殊的问题？一个类比的问题？你能否解决这个问题的一部分？……如果需要的话，你能不能改变未知数或数据，或者二者都改变，以使新未知数和新数据彼此更接近？

……

尽管这张表是为解决数学问题而设计的，但是它对计算机问题求解具有深刻的启迪意义，在问题求解时离不开类似的分析问题的思维方法。

在设计算法求解特定问题时，算法的准确性和复杂性往往是人们关注的重点。首先，算法执行的结果必须正确。**算法正确是指对于问题界定的所有问题实例，算法执行后都能得到正确的结果**。其次，算法的复杂性要适中。计算机的资源（包括时间和内存）是有限的，所以，算法必须在有限的资源条件下正确地求解问题，更多有关算法复杂性分析的讨论见 1.4 节。可见，用计算机算法求解问题不是一个很容易的过程。

实际上，从一个问题的提出，到计算机可执行的、满足准确性和复杂性要求的程序实现，可以看作是计算机问题求解的一个周期。**问题求解周期**包括问题简化、模型构建、算法设计、程序设计与调试等过程。

问题简化：大多数实际问题涉及的因素很多，在求解之前必须经过简化，得到问题的原型（Prototype）。这个原型应当是没有歧义的，可以用 1.1 节介绍的“问题描述 - 输入 - 输出”标准方法加以定义（本书所讨论的问题都以原型的形式出现）。

模型构建：问题的原型简洁地叙述了问题的条件、限制和求解目标，但是没有表明问题的本质。很多表面上看起来完全不同的问题具有相同的本质。模型构建是一个非常灵活的过程，同一个问题可以构建不同的模型，模型求解的难度也有差异。

得到数学模型以后，只要不是简单到可以直接求解或者套用经典模型的程度，一般都需要进行模型分析，得到初步结论。很多经典模型前人已经做过详细而透彻的研究，学习算法时应当尽量多地积累这样的经典模型及其求解算法。另一方面，如果模型是新的，需要继续进行算法设计，这一步往往比构建模型更有挑战性。

算法设计：由于问题答案最终需要由计算机执行得到，因此，在模型构建和分析后要进行算法设计。这是本书探讨的重点，也是计算机问题求解的核心要素。

程序设计与调试：这是用特定程序设计语言实现算法的过程，属于程序设计的范畴。

例 1-3 展示了一个实际问题的求解过程。

【例 1-3】补丁与 Bug 问题。

Bug 就是人们所说的错误。用户在使用软件时总是希望其错误越少越好，最好没有错误。但是推出一个没有错误的软件几乎不可能，所以很多软件公司都在不断地发布补丁（有时这种补丁甚至是收费的）。T 公司就是其中之一。上个月，T 公司推出了一个新的字处理软件，随后发布了一批补丁。最近 T 公司发现其发布的补丁有致命的问题，那就是一个补丁在排除某些错误的同时，往往会加入另一些错误。此字处理软件中只可能出现 n 个特定的错误，这 n 个错误是由软件本身决定的。T 公司目前共发布了 m 个补丁，对于每一个补丁，都有特定的适用环境，某个补丁只有在当前软件中包含某些错误而同时又不包含另一些错误时才可以使用，如果它被使用，它将修复某些错误而同时加入其他错误。另外，使用每个补丁都要耗费一定的时间（即补丁程序的运行时间）。

现在 T 公司的问题很简单，其字处理软件的初始版本不幸地包含了全部 n 个错误，有没有可能通过使用这 m 个补丁（任意顺序地使用，一个补丁可使用多次），使此字处理软件成为一个没有错误的软件。如果可能，希望找到总耗时最少的方案。

显然，这是一个比较复杂的实际问题，在求解之前，需要我们对该问题进行适当的简化和较为形式化的定义，得到如下问题原型。

问题描述：将 T 公司的字处理软件中可能出现的 n 个错误记为集合 $B = \{b_1, b_2, \cdots, b_n\}$，发布的 m 个补丁记为集合 $P = \{p_1, p_2, \cdots, p_m\}$。对于每一个补丁 p_i，假设存在错误集合 B_{i+}

和 B_{i-}。当软件包含了 B_{i+} 中的所有错误，而没有包含 B_{i-} 中的任何错误时，补丁 p_i 才可以被使用，否则不能使用，显然 B_{i+} 和 B_{i-} 的交集为空。补丁 p_i 将修复某些错误而同时加入某些错误，设错误集合为 F_{i+} 和 F_{i-}，使用过补丁 p_i 之后，F_{i-} 中的任何错误都不会在软件中出现，而软件将包含 F_{i+} 中的所有错误，同样 F_{i+} 和 F_{i-} 的交集为空。另外，使用每个补丁都要耗费一定的时间 T_i。

现在 T 公司的字处理软件的初始版本不幸地包含了全部 n 个错误，试编写程序判断有没有可能通过使用这 m 个补丁(任意顺序地使用，一个补丁可使用多次)，使此字处理软件成为一个没有错误的软件。如果可能，希望找到总耗时最少的方案。

输入格式： 第一行有两个正整数 n 和 m，n 表示 Bug 总数，m 表示补丁总数，$1 \leqslant n \leqslant 15$，$1 \leqslant m \leqslant 100$。接下来 m 行给出了 m 个补丁的信息。每行包括一个正整数 T_i（表示此补丁程序 p_i 的运行耗时）和两个长度为 n 的字符串，中间用一个空格符隔开。

第一个字符串，如果第 k 个字符为“+”，则表示 b_k 属于 B_{i+}；若字符为“-”，则表示 b_k 属于 B_{i-}；若字符为“0”，则 b_k 既不属于 B_{i+} 也不属于 B_{i-}，即软件中是否包含 b_k 不影响补丁 p_i 是否可用。

第二个字符串，如果第 k 个字符为“+”，则表示 b_k 属于 F_{i+}；若字符为“-”，则表示 b_k 属于 F_{i-}；若字符为“0”，则 b_k 既不属于 F_{i+} 也不属于 F_{i-}，即软件中是否包含 b_k 不会因使用补丁 p_i 而改变。

输出格式： 输出一个整数，如果问题有解，输出总耗时，否则输出 0。

经过上述的问题简化得到问题原型后，下一步就是对该问题进行分析，构建相应的模型。客观世界是纷繁复杂的，当人们面对一个新问题时，通常的想法是**通过分析、变形和转换，得到本质相同且熟悉的问题**，这就是**归化思想**。如果把初始的问题或对象称为**原型**，则把归化后的相对定型的模拟化或理想化的对象称为**模型**。模型化的方向主要有图论模型、数学模型和规划(动态规划)模型。

因为补丁与 Bug 问题涉及耗时“最少”的要求，自然可以联想到图论中的最短路径问题。如果把 n 个 Bug 的状态(存在和不存在)的组合用一个 0−1 字符串表示(称为模式串)，显然，所有 n 个 Bug 构成的模式串的数目最多是 2^n 个，同时，运行一个补丁就会导致从一个模式串转换到另外一个模式串。如果把模式串组成一个图模型的顶点，那么特定的补丁就构成该图的有向边，该补丁运行的时间则是该边的权重，如图 1-2 所示。当构建完补丁与 Bug 问题的图模型后，原问题的解就对应了从起点模式串(11111)到终点模式串(00000)的一条最短路径。

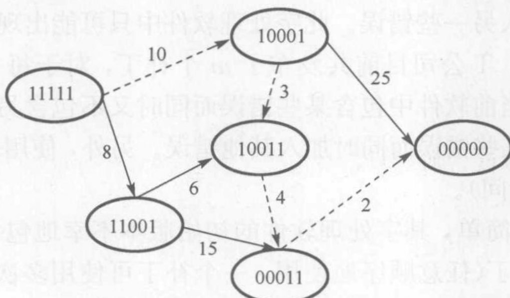

图 1-2 补丁与 Bug 问题的图模型示例

其中每个顶点都表示软件的状态，0−1 字符串中的 1 表示对应 Bug 存在，0 表示对应 Bug 不存在，每一条边表示一个补丁，边的权重表示执行补丁所需要的时间，虚线所示的路径则是该示例的最优解。

接下来就是针对补丁与Bug的图模型设计求解算法。对于学过图论的读者应该很容易想到，此问题可以直接应用Dijkstra最短路径算法求解。鉴于Dijkstra最短路径算法的典型性，Dijkstra算法的程序设计和调试的过程请读者自己完成。

另外，补丁和Bug问题也可以应用有限状态自动机模型，但是基于该模型的求解过程会复杂些，学有余力的读者可以自己设计和实现。

1.3 算法与程序

算法(Algorithm)是计算机问题求解的核心和关键。虽然人们对算法一词非常熟悉，但到目前为止，对于算法尚没有统一而精确的定义。有人说：算法就是一组有穷的规则，它们规定了解决某一特定问题的一系列运算。而Thomas H. Cormen等人在《Introduction to Algorithms》一书中将算法描述为：**算法是任何定义好了的计算程式，它取某些值或值的集合作为输入，并产生某些值或值的集合作为输出**。因此，算法是将输入转化为输出的一系列计算步骤。概括起来，算法有以下5个特性：

(1) **确定性**。算法的每一种运算(包括判断)必须有确切的定义，即每一种运算应该执行何种动作必须是相当清楚的、无二义性的。

(2) **可实现性**。算法中有待实现的运算都是相当基本的，每种运算至少在原理上能由人用纸和笔在有限的时间内完成。

(3) **具有数据输入**。一个算法有零个或多个数据输入，它们是在算法开始之前对算法最初赋予的量，这些输入取自特定的对象集合。

(4) **具有数据输出**。一个算法产生一个或多个输出，它们是同输入有某种特定关系的量。

(5) **有穷性**。一个算法必须在执行了有穷步之后终止。

程序(Program)是算法用某种程序设计语言的实现结果。程序可以不满足算法的第5个特性。例如操作系统，它是一个无限循环中执行的程序，因而不是一个算法。当然，如果把操作系统按照任务分解成一些独立的问题，每一个问题则由操作系统中的一个子程序通过特定的算法来实现。

1.4 算法复杂性分析

算法的复杂性是算法效率的度量，是评价算法优劣的重要依据。一个算法复杂性的高低体现了运行该算法所需要的计算机资源的多少。算法执行所需的资源越多，则它的复杂性越高；反之，算法所需的资源越低，则其复杂性越低。**时间和空间**(即内存)是计算机最重要的两种资源，因而算法的复杂性分为**时间复杂性**和**空间复杂性**。

不言而喻，对于任意给定的问题，复杂性尽可能低的算法是问题求解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，是选用算法时应遵循的一个重要准则。总之，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。

1.4.1 空间复杂性

算法的空间复杂性是指计算机执行一个算法所需要占用的存储空间，它是算法优劣的重

要度量指标。一般地，空间复杂性越小，算法越好。算法执行需要的空间包括指令空间，数据空间和系统栈空间三大类。

指令空间用来存储经过编译之后的程序指令。程序所需的指令空间的大小取决于如下因素：① 把程序编译成机器代码的编译器；② 编译时实际采用的编译器选项；③ 目标计算机。程序编译时使用的编译器不同，产生的机器代码的长度就会有差异。另外，有些编译器带有选项，如优化模式、覆盖模式等，如果设置的编译选项不同，产生机器代码也会不同。当然，目标计算机的配置也会影响代码的规模。例如，如果计算机具有浮点处理的硬件部件，那么每个浮点操作可以转化为一条机器指令；否则，必须生成仿真的浮点计算代码，使整个机器代码加长。一般情况下，**指令空间对于所解决的特定问题不够敏感，以至于可以忽略不计**。

数据空间用来存储所有常量和变量的值，分成两部分：① 存储常量和简单变量；② 存储复合变量。前者所需的空间取决于所使用的计算机和编译器，以及变量与常量的数目。因为人们往往是计算所需内存的字节数，而每字节所占的数位依赖于具体的机器环境(16 位字长计算机中 C/C++各数据类型所需内存参阅表 2-1)。后者包括数据结构所需的空间及动态分配的空间。结构变量所占空间等于各个成员所占空间的累加；数组变量所占空间等于数组大小乘以单个数组元素所占的空间。

系统栈空间保存函数调用返回时恢复运行所需要的信息。当一个函数被调用时，下面数据将被保存在系统栈中：① 返回地址；② 所有局部变量的值、递归函数的传值形式参数的值；③ 所有引用参数以及常量引用参数的定义。**对于递归程序调用，系统栈空间大小还依赖于递归调用的深度**，而且，往往这是决定系统栈空间大小的主要因素。

程序 1-1　数组求和程序

```
int sum1(int iArray[], int iLen){
    int i=0;
    int iSum=0;
    for(i=0; i<iLen; i++)
        iSum += iArray[i];
    return iSum;
}
int sum2(int iArray[], int iLen){
    int iSum=0;
    if(iLen>0)
        iSum=sum2(iArray, iLen-1) + iArray[iLen-1];
    return iSum;
}
```

在程序 1-1 中，程序 sum1 采用循环累加的办法，而程序 sum2 采用递归的办法。在 sum1 中，int 型变量 i，iSum，iLen 各自需要分配 4 字节(在 16 位字长计算机为 2 字节)内存空间，指针型参数 iArray 需要分配 4 字节内存空间。在 sum2 中，递归栈空间包括参数 iArray，iLen 所需的 8 字节，以及保存返回地址所需的 2 字节(假定是 near 指针)。每次调用 sum2 就需要 10 字节空间，假定某一个实例的递归深度是 n，则总共需要 $10\times(n+1)$ 的内存空间。

随着半导体技术的飞跃式发展，存储器资源的成本越来越低，人们对于算法空间复杂性的关注度也越来越低。因此，本书后续章节分析算法的复杂性时，也忽略了空间复杂性分析。

1.4.2　时间复杂性

1.4.2.1　时间复杂性的表示

算法的时间复杂性是指算法运行所需要的时间资源的量，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算法要解的问题的规模（N）、算法的输入（I）和算法本身（A）的函数。比如，在例1-2的迷宫问题中，棋盘的大小 N 反映了迷宫问题的求解规模，某一个特定的棋盘布局则表示了迷宫问题的输入。显然，用某一个特定算法 A 求解该问题时，求解图1-1(a)所示问题实例和图1-1(b)所示问题实例所需的时间不一样。一般地，问题规模 N 越大，所需要的时间就越多。即使在问题规模 N 相同的情况下，求解不同输入所对应的问题实例所需要的时间也不一样，比如，应用普通广度优先搜索算法求解图1-1(b)和图1-1(c)的问题实例所需要的时间也不尽相同。

不失一般性，如果分别用 N，I 和 A 来表示算法要解问题的规模、算法的输入和算法本身，用 T 表示算法的时间复杂性，那么应有

$$T = T(N, I, A) \tag{1-1}$$

式中，$T(N, I, A)$ 是 N，I 和 A 的一个确定的三元函数。通常，人们让 A 隐含在复杂性函数名当中，于是公式(1-1)可以简写为

$$T = T(N, I) \tag{1-2}$$

运行于特定计算机的算法程序由该计算机系统的若干操作和运算指令组成，显然，运行算法所需要的时间等价于执行这些指令所需要的时间之和。因为不同的计算机其指令以及执行指令所需要的时间不一定相同。不失一般性，假定算法运行在一台抽象的计算机上，此抽象的计算机提供 k 种元运算，分别记为 O_1，O_2，…，O_k；再设这些元运算每执行一次所需要的时间分别为 t_1，t_2，…，t_k。给定算法 A，设经过统计，用到元运算 O_i 的次数为 e_i，$i=1, 2, \cdots, k$，显然对于每一个 i，$1 \leqslant i \leqslant k$，$e_i$ 是 N 和 I 的函数，即 $e_i = e_i(N, I)$。那么得到算法执行时间的表达式：

$$T(N, I) = \sum_{i=1}^{k} t_i e_i(N, I) \tag{1-3}$$

式中，t_1，t_2，…，t_k 是跟 N 和 I 无关的常量。

对于任何一个问题，对应于特定规模 N 的问题实例通常都比较多，甚至大得无法计量。比如，迷宫问题的规模 $N=1000$ 时，其对应的问题实例则有 $2^{1000\times1000}$ 个。显然，不可能对规模 N 的每一种合法输入 I 都统计 $e_i(N, I)$，$i=1, 2, \cdots, k$。因此 $T(N, I)$ 的表达式还得进一步简化，或者说，只在规模为 N 的某些或某类有代表性的合法输入中统计相应的 $e_i(N, I)$，$i=1, 2, \cdots, k$，评价其时间复杂性。

下面只考虑三种情况的时间复杂性，即最坏情况、最好情况和平均情况下的时间复杂性，并分别记为 $T_{\max}(N)$，$T_{\min}(N)$ 和 $T_{\text{avg}}(N)$。在数学上有

$$T_{\max}(N) = \max_{I\in D_N} T(N, I) = \max_{I\in D_N} \sum_{i=1}^{k} t_i e_i(N, I) = \sum_{i=1}^{k} t_i e_i(N, I^*) = T(N, I^*) \tag{1-4}$$

试读结束：需要全本请在线购买： www.ertongbook.com