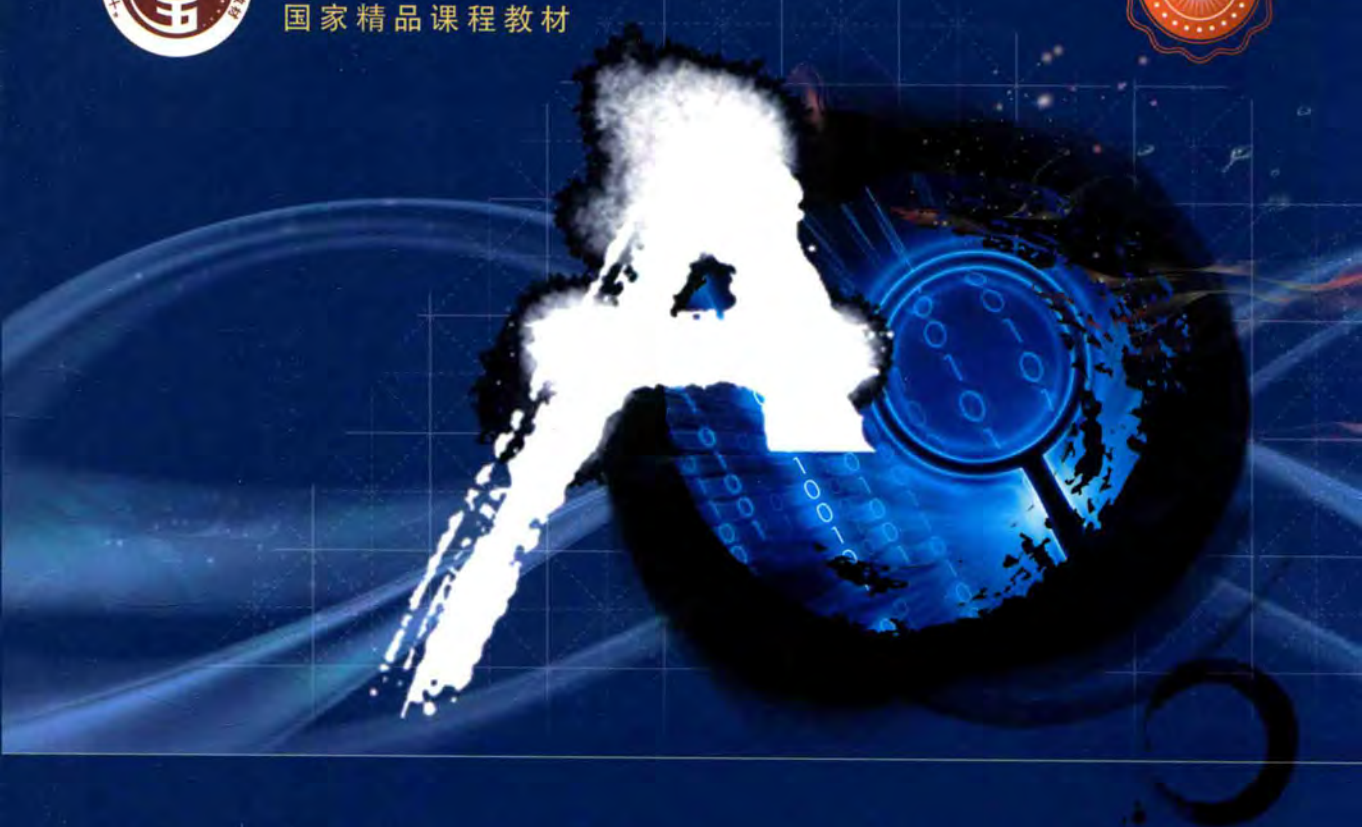




“十二五”普通高等教育本科国家级规划教材
国家精品课程教材



计算机算法设计与分析习题解答

(第5版)

◎ 王晓东 编著

非外借



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

“十二五”普通高等教育本科国家级规划教材
国家精品课程教材

计算机算法设计与分析 习题解答 (第5版)

王晓东 编著



电子工业出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书是与“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析（第5版）》配套的辅助教材和国家精品课程教材，分别对主教材中的算法分析题和算法实现题给出了解答或解题思路提示。为了提高学生灵活运用算法设计策略解决实际问题的能力，本书还将主教材中的许多习题改造成算法实现题，要求学生设计出求解算法并上机实现。本书教学资料包含各章算法实现题、测试数据和答案，可在华信教育资源网免费注册下载。

本书内容丰富，理论联系实际，可作为高等学校计算机科学与技术、软件工程、信息安全、信息与计算科学等专业本科生和研究生学习计算机算法设计的辅助教材，也是工程技术人员和自学者的参考书。

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

计算机算法设计与分析习题解答/王晓东编著. —5版. —北京：电子工业出版社，2018.10

ISBN 978-7-121-34438-1

I. ① 计… II. ① 王… III. ① 电子计算机—算法设计—高等学校—题解 ② 电子计算机—算法分析—高等学校—题解 IV. ① TP301.6-44

中国版本图书馆 CIP 数据核字（2018）第 120711 号

策划编辑：章海涛

责任编辑：章海涛

印 刷：三河市良远印务有限公司

装 订：三河市良远印务有限公司

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：22.75 字数：580 千字

版 次：2005 年 8 月第 1 版

2018 年 10 月第 5 版

印 次：2018 年 10 月第 1 次印刷

定 价：56.00 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）88254888，88258888。

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

本书咨询联系方式：192910558（QQ 群）。

前 言

一些著名的计算机科学家在有关计算机科学教育的论述中认为,计算机科学是一种创造性思维活动,其教育必须面向设计。“计算机算法设计与分析”正是一门面向设计,且处于计算机学科核心地位的教育课程。通过对计算机算法系统的学习与研究,理解掌握算法设计的主要方法,培养对算法的计算复杂性正确分析的能力,为独立设计算法和对算法进行复杂性分析奠定坚实的理论基础,对每一位从事计算机系统结构、系统软件和应用软件研究与开发的科技工作者都是非常重要和必不可少的。课程结合我国高等学校教育工作的现状,追踪国际计算机科学技术的发展水平,更新了教学内容和教学方法,以算法设计策略为知识单元,在内容选材、深度把握、系统性和可用性方面进行了精心设计,力图适合高校本科生教学对学时数和知识结构的要求。

本书是“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析(第5版)》(ISBN 978-7-121-34439-8)配套的辅助教材,对《计算机算法设计与分析(第5版)》一书中的全部习题做了详尽的解答,旨在让使用该书的教师更容易教,学生更容易学。为了便于对照阅读,本书的章序与《计算机算法设计与分析(第5版)》一书的章序保持一致,且一一对应。

本书的内容是对《计算机算法设计与分析(第5版)》的较深入的扩展,许多教材中无法讲述的较深入的主题通过习题的形式展现出来。为了加强学生灵活运用算法设计策略解决实际问题的能力,本书将主教材中的许多习题改造成算法实现题,要求学生不仅设计出解决具体问题的算法,而且能上机实现。作者的教学实践反映出这类算法实现题的教学效果非常好。作者还结合国家精品课程建设,建立了“算法设计与分析”教学网站。国家精品资源共享课地址:http://www.icourses.cn/sCourse/course_2535.html。欢迎广大读者访问作者的教学网站并提出宝贵意见。

在本书编写过程中,福州大学“211工程”计算机与信息工程重点学科实验室为本书的写作提供了优良的设备与工作环境。电子工业出版社负责本书编辑出版工作的全体同仁为本书的出版付出了大量辛勤劳动,他们认真细致、一丝不苟的工作精神保证了本书的出版质量。在此,谨向每位曾经关心和支持本书编写工作的各方面人士表示衷心的感谢!

作 者

目 录

第 1 章 算法概述	1
算法分析题 1	1
1-1 函数的渐近表达式	1
1-2 $O(1)$ 和 $O(2)$ 的区别	1
1-3 按渐近阶排列表式	1
1-4 算法效率	1
1-5 硬件效率	1
1-6 函数渐近阶	2
1-7 $n!$ 的阶	2
1-8 $3n+1$ 问题	2
1-9 平均情况下的计算时间复杂性	2
算法实现题 1	3
1-1 统计数字问题	3
1-2 字典序问题	4
1-3 最多约数问题	4
1-4 金币阵列问题	6
1-5 最大间隙问题	8
第 2 章 递归与分治策略	11
算法分析题 2	11
2-1 证明 Hanoi 塔问题的递归算法与非递归算法实际上是一回事	11
2-2 判断这 7 个算法的正确性	12
2-3 改写二分搜索算法	15
2-4 大整数乘法的 $O(nm^{\log(3/2)})$ 算法	16
2-5 5 次 $n/3$ 位整数的乘法	16
2-6 矩阵乘法	18
2-7 多项式乘积	18
2-8 $O(1)$ 空间子数组换位算法	19
2-9 $O(1)$ 空间合并算法	21
2-10 \sqrt{n} 段合并排序算法	27
2-11 自然合并排序算法	28
2-12 第 k 小元素问题的计算时间下界	29
2-13 非增序快速排序算法	31

2-14 构造 Gray 码的分治算法.....	31
2-15 网球循环赛日程表.....	32
2-16 二叉树 T 的前序、中序和后序序列	35
算法实现题 2.....	36
2-1 众数问题.....	36
2-2 马的 Hamilton 周游路线问题	37
2-3 半数集问题.....	44
2-4 半数单集问题.....	46
2-5 有重复元素的排列问题.....	46
2-6 排列的字典序问题.....	47
2-7 集合划分问题.....	49
2-8 集合划分问题.....	50
2-9 双色 Hanoi 塔问题.....	51
2-10 标准二维表问题.....	52
2-11 整数因子分解问题.....	53
第 3 章 动态规划	54
算法分析题 3.....	54
3-1 最长单调递增子序列.....	54
3-2 最长单调递增子序列的 $O(n\log n)$ 算法	54
3-3 整数线性规划问题.....	55
3-4 二维 0-1 背包问题	56
3-5 Ackermann 函数.....	57
算法实现题 3.....	59
3-1 独立任务最优调度问题.....	59
3-2 最优批处理问题.....	61
3-3 石子合并问题.....	67
3-4 数字三角形问题.....	68
3-5 乘法表问题.....	69
3-6 租用游艇问题.....	70
3-7 汽车加油行驶问题.....	70
3-8 最小 m 段和问题.....	71
3-9 圈乘运算问题.....	72
3-10 最大长方体问题.....	78
3-11 正则表达式匹配问题.....	79
3-12 双调旅行售货员问题.....	83
3-13 最大 k 乘积问题.....	84
3-14 最少费用购物问题.....	86
3-15 收集样本问题.....	87

3-16	最优时间表问题	89
3-17	字符串比较问题	89
3-18	有向树 k 中值问题	90
3-19	有向树独立 k 中值问题	94
3-20	有向直线 m 中值问题	98
3-21	有向直线 2 中值问题	101
3-22	树的最大连通分支问题	103
3-23	直线 k 中值问题	105
3-24	直线 k 覆盖问题	109
3-25	m 处理器问题	113
第 4 章 贪心算法		116
算法分析题 4		116
4-1	程序最优存储问题	116
4-2	最优装载问题的贪心算法	116
4-3	Fibonacci 序列的哈夫曼编码	116
4-4	最优前缀码的编码序列	117
算法实现题 4		117
4-1	会场安排问题	117
4-2	最优合并问题	118
4-3	磁带最优存储问题	118
4-4	磁盘文件最优存储问题	119
4-5	程序存储问题	120
4-6	最优服务次序问题	120
4-7	多处最优服务次序问题	121
4-8	d 森林问题	122
4-9	虚拟汽车加油问题	123
4-10	区间覆盖问题	124
4-11	删数问题	124
4-12	磁带最大利用率问题	125
4-13	非单位时间任务安排问题	126
4-14	多元 Huffman 编码问题	127
4-15	最优分解问题	128
第 5 章 回溯法		130
算法分析题 5		130
5-1	装载问题改进回溯法 1	130
5-2	装载问题改进回溯法 2	131
5-3	0-1 背包问题的最优解	132
5-4	最大团问题的迭代回溯法	134

5-5 旅行售货员问题的费用上界.....	135
5-6 旅行售货员问题的上界函数.....	136
算法实现题 5.....	137
5-1 子集和问题.....	137
5-2 最小长度电路板排列问题.....	138
5-3 最小重量机器设计问题.....	140
5-4 运动员最佳配对问题.....	141
5-5 无分隔符字典问题.....	142
5-6 无和集问题.....	144
5-7 n 色方柱问题.....	145
5-8 整数变换问题.....	150
5-9 拉丁矩阵问题.....	151
5-10 排列宝石问题.....	152
5-11 重复拉丁矩阵问题.....	154
5-12 罗密欧与朱丽叶的迷宫问题.....	156
5-13 工作分配问题.....	158
5-14 布线问题.....	159
5-15 最佳调度问题.....	160
5-16 无优先级运算问题.....	161
5-17 世界名画陈列馆问题.....	163
5-18 世界名画陈列馆问题 (不重复监视)	166
5-19 算 m 点问题.....	169
5-20 部落卫队问题.....	171
5-21 子集树问题.....	173
5-22 0-1 背包问题.....	174
5-23 排列树问题.....	176
5-24 一般解空间搜索问题.....	177
5-25 最短加法链问题.....	179
第 6 章 分支限界法.....	185
算法分析题 6.....	185
6-1 0-1 背包问题的栈式分支限界法	185
6-2 释放结点空间的队列式分支限界法.....	187
6-3 及时删除不用的结点.....	188
6-4 用最大堆存储活结点的优先队列式分支限界法.....	189
6-5 释放结点空间的优先队列式分支限界法.....	192
6-6 团顶点数的上界.....	194
6-7 团顶点数改进的上界.....	194
6-8 修改解旅行售货员问题的分支限界法.....	195
6-9 试修改解旅行售货员问题的分支限界法, 使得算法保存已产生的排列树.....	197

6-10 电路板排列问题的队列式分支限界法	199
算法实现题 6	201
6-1 最小长度电路板排列问题	201
6-2 最小权顶点覆盖问题	203
6-3 无向图的最大割问题	206
6-4 最小重量机器设计问题	209
6-5 运动员最佳配对问题	212
6-6 n 后问题	214
6-7 布线问题	216
6-8 最佳调度问题	218
6-9 无优先级运算问题	220
6-10 世界名画陈列馆问题	223
6-11 子集空间树问题	226
6-12 排列空间树问题	229
6-13 一般解空间的队列式分支限界法	232
6-14 子集空间树问题	236
6-15 排列空间树问题	241
6-16 一般解空间的优先队列式分支限界法	246
6-17 推箱子问题	250
第 7 章 概率算法	256
算法分析题 7	256
7-1 模拟正态分布随机变量	256
7-2 随机抽样算法	256
7-3 随机产生 m 个整数	257
7-4 集合大小的概率算法	258
7-5 生日问题	258
7-6 易验证问题的拉斯维加斯算法	259
7-7 用数组模拟有序链表	260
7-8 $O(n^{3/2})$ 舍伍德型排序算法	260
7-9 n 后问题解的存在性	260
7-10 整数因子分解算法	262
7-11 非蒙特卡罗算法的例子	262
7-12 重复 3 次的蒙特卡罗算法	263
7-13 集合随机元素算法	263
7-14 由蒙特卡罗算法构造拉斯维加斯算法	265
7-15 产生素数算法	265
7-16 矩阵方程问题	265
算法实现题 7	266

7-1 模平方根问题.....	266
7-2 素数测试问题.....	268
7-3 集合相等问题.....	269
7-4 逆矩阵问题.....	269
7-5 多项式乘积问题.....	270
7-6 皇后控制问题.....	270
7-7 3-SAT 问题.....	274
7-8 战车问题.....	275
第 8 章 线性规划与网络流.....	278
算法分析题 8.....	278
8-1 线性规划可行区域无界的例子.....	278
8-2 单源最短路与线性规划.....	278
8-3 网络最大流与线性规划.....	279
8-4 最小费用流与线性规划.....	279
8-5 运输计划问题.....	279
8-6 单纯形算法.....	280
8-7 边连通度问题.....	281
8-8 有向无环网络的最大流.....	281
8-9 无向网络的最大流.....	281
8-10 最大流更新算法.....	282
8-11 混合图欧拉回路问题.....	282
8-12 单源最短路与最小费用流.....	282
8-13 中国邮路问题.....	282
算法实现题 8.....	283
8-1 飞行员配对方案问题.....	283
8-2 太空飞行计划问题.....	284
8-3 最小路径覆盖问题.....	285
8-4 魔术球问题.....	286
8-5 圆桌问题.....	287
8-6 最长递增子序列问题.....	287
8-7 试题库问题.....	290
8-8 机器人路径规划问题.....	291
8-9 方格取数问题.....	294
8-10 餐巾计划问题.....	298
8-11 航空路线问题.....	299
8-12 软件补丁问题.....	300
8-13 星际转移问题.....	301
8-14 孤岛营救问题.....	302
8-15 汽车加油行驶问题.....	304

8-16	数字梯形问题	307
8-17	运输问题	311
8-18	分配工作问题	314
8-19	负载平衡问题	315
8-20	最长 k 可重区间集问题	317
8-21	最长 k 可重线段集问题	319
第 9 章 串与序列的算法		323
算法分析题 9		323
9-1	简单子串搜索算法最坏情况复杂性	323
9-2	后缀重叠问题	323
9-3	改进前缀函数	323
9-4	确定所有匹配位置的 KMP 算法	324
9-5	特殊情况下简单子串搜索算法的改进	325
9-6	简单子串搜索算法的平均性能	325
9-7	带间隙字符的模式串搜索	326
9-8	串接的前缀函数	326
9-9	串的循环旋转	327
9-10	失败函数性质	327
9-11	输出函数性质	328
9-12	后缀数组类	328
9-13	最长公共扩展查询	329
9-14	最长公共扩展性质	332
9-15	后缀数组性质	333
9-16	后缀数组搜索	334
9-17	后缀数组快速搜索	335
算法实现题 9		338
9-1	安全基因序列问题	338
9-2	最长重复子串问题	342
9-3	最长回文子串问题	343
9-4	相似基因序列性问题	344
9-5	计算机病毒问题	345
9-6	带有子串包含约束的最长公共子序列问题	347
9-7	多子串排斥约束的最长公共子序列问题	349
参考文献		351

第 1 章 算法概述

算法分析题 1

1-1 函数的渐近表达式。

求下列函数的渐近表达式：

$3n^2+10n$; $n^2/10+2^n$; $21+1/n$; $\log n^3$; $10\log 3^n$ 。

分析与解答：

$3n^2+10n=O(n^2)$; $\log n^3=O(\log n)$;

$n^2/10+2^n=O(2^n)$; $10\log 3^n=O(n)$ 。

$21+1/n=O(1)$;

1-2 $O(1)$ 和 $O(2)$ 的区别。

分析与解答：根据符号 O 的定义易知 $O(1)=O(2)$ 。用 $O(1)$ 或 $O(2)$ 表示同一个函数时，差别仅在于其中的常数因子。

1-3 按渐近阶排列表达式。

按照渐近阶从低到高的顺序排列以下表达式： $4n^2$, $\log n$, 3^n , $20n$, 2 , $n^{2/3}$ 。又 $n!$ 应该排在哪一位？

分析与解答：按渐近阶从低到高，函数排列顺序如下： 2 , $\log n$, $n^{2/3}$, $20n$, $4n^2$, 3^n , $n!$ 。

1-4 算法效率。

(1) 假设某算法在输入规模为 n 时的计算时间为 $T(n)=3\times 2^n$ 。在某台计算机上实现并完成该算法的时间为 t 秒。现有另一台计算机，其运行速度为第一台的 64 倍，那么在这台新机器上用同一算法在 t 秒内能解输入规模为多大的问题？

(2) 若上述算法的计算时间改进为 $T(n)=n^2$ ，其余条件不变，则在新机器上用 t 秒时间能解输入规模为多大的问题？

(3) 若上述算法的计算时间进一步改进为 $T(n)=8$ ，其余条件不变，那么在新机器上用 t 秒时间能解输入规模为多大的问题？

分析与解答：

(1) 设新机器用同一算法在 t 秒内能解输入规模为 n_1 的问题。因此有： $t=3\times 2^n=3\times 2^{n_1}/64$ ，解得 $n_1=n+6$ 。

(2) $n_1^2=64n^2\Rightarrow n_1=8n$ 。

(3) 由于 $T(n)=\text{常数}$ ，因此算法可解任意规模的问题。

1-5 硬件效率。

硬件厂商 XYZ 公司宣称他们最新研制的微处理器运行速度为其竞争对手 ABC 公司同类产品的 100 倍。对于计算复杂性分别为 n 、 n^2 、 n^3 和 $n!$ 的各算法，若用 ABC 公司的计算机在 1 小时内能解输入规模为 n 的问题，那么用 XYZ 公司的计算机在 1 小时内分别能解输入

规模为多大的问题?

分析与解答:

$$n' = 100n$$

$$n'^3 = 100n^3 \Rightarrow n' = \sqrt[3]{100n} = 4.64n$$

$$n'^2 = 100n^2 \Rightarrow n' = 10n$$

$$n'! = 100n! \Rightarrow n' < n + \log 100 = n + 6.64$$

1-6 函数渐近阶。

对于下列各组函数 $f(n)$ 和 $g(n)$, 确定 $f(n) = O(g(n))$ 或 $f(n) = \Omega(g(n))$ 或 $f(n) = \theta(g(n))$, 并简述理由。

$$(1) f(n) = \log n^2;$$

$$g(n) = \log n + 5$$

$$(5) f(n) = 10;$$

$$g(n) = \log 10$$

$$(2) f(n) = \log n^2;$$

$$g(n) = \sqrt{n}$$

$$(6) f(n) = \log^2 n;$$

$$g(n) = \log n$$

$$(3) f(n) = n;$$

$$g(n) = \log^2 n$$

$$(7) f(n) = 2^n;$$

$$g(n) = 100n^2$$

$$(4) f(n) = n \log n + n;$$

$$g(n) = \log n$$

$$(8) f(n) = 2^n;$$

$$g(n) = 3^n$$

分析与解答:

$$(1) \log n^2 = \theta(\log n + 5)$$

$$(5) 10 = \theta(\log 10)$$

$$(2) \log n^2 = O(\sqrt{n})$$

$$(6) \log^2 n = \Omega(\log n)$$

$$(3) n = \Omega(\log^2 n)$$

$$(7) 2^n = \Omega(100n^2)$$

$$(4) n \log n + n = \Omega(\log n)$$

$$(8) 2^n = O(3^n)$$

1-7 $n!$ 的阶。

证明: $n! = o(n^n)$ 。

分析与解答: Stirling's approximation:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left[1 + \theta\left(\frac{1}{n}\right)\right]$$
$$\lim_{n \rightarrow \infty} n! / n^n = \frac{\sqrt{2\pi n} \left[1 + \theta\left(\frac{1}{n}\right)\right]}{e^n} = 0$$

1-8 $3n+1$ 问题。

下面的算法片段用于确定 n 的初始值。试分析该算法片段所需计算时间的上界和下界。

```
while(n>1)
    if(odd(n))
        n = 3*n+1;
    else
        n = n/2;
```

分析与解答: 该算法表述的是著名的 $3n+1$ 问题。在最坏情况下, 该算法的计算时间下界显然为 $\Omega(\log n)$ 。

算法的计算时间上界至今未知。算法是否在有限时间内结束, 至今还是一个悬而未决的问题。日本学者米田信夫曾对 10^{13} 内的所有自然数验证上述算法均在有限步结束。人们猜测, 对所有自然数, 上述算法均在有限步结束, 但无法给出理论证明, 因此也无法分析上述算法的计算时间上界。这个猜测就成为著名的 $3n+1$ 猜想, 也称为 Collatz 猜想。

1-9 平均情况下的计算时间复杂性。

证明：如果一个算法在平均情况下的计算时间复杂性为 $\theta(f(n))$ ，则该算法在最坏情况下所需的计算时间为 $\Omega(f(n))$ 。

分析与解答：

$$\begin{aligned} T_{\text{avg}}(N) &= \sum_{I \in D_N} P(I)T(N, I) \leq \sum_{I \in D_N} P(I) \max_{I' \in D_N} T(N, I') \\ &= T(N, I^*) \sum_{I \in D_N} P(I) = T(N, I^*) = T_{\text{max}}(N) \end{aligned}$$

因此， $T_{\text{max}}(N) = \Omega(T_{\text{avg}}(N)) = \Omega(\theta(f(n))) = \Omega(f(n))$ 。

算法实现题 1

1-1 统计数字问题。

问题描述：一本书的页码从自然数 1 开始顺序编码直到自然数 n 。书的页码按照通常的习惯编排，每个页码不含多余的前导数字 0。例如，第 6 页用数字 6 表示而不是 06 或 006 等。数字计数问题要求对给定书的总页码 n ，计算书的全部页码分别用到多少次数目 0、1、2、…、9。

算法设计：给定表示书的总页码的十进制整数 n ($1 \leq n \leq 10^9$)，计算书的全部页码中分别用到多少次数目 0、1、2、…、9。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。每个文件只有 1 行，给出表示书的总页码的整数 n 。

结果输出：将计算结果输出到文件 output.txt。输出文件共 10 行，在第 k ($k=1, 2, \dots, 10$) 行输出页码中用到数字 $k-1$ 的次数。

输入文件示例

input.txt

11

输出文件示例

output.txt

1

4

1

1

1

1

1

1

1

1

分析与解答：考察由 0, 1, 2, …, 9 组成的所有 n 位数。从 n 个 0 到 n 个 9 共有 10^n 个 n 位数。在这 10^n 个 n 位数中，0, 1, 2, …, 9 每个数字使用次数相同，设为 $f(n)$ 。 $f(n)$ 满足如下递归式：

$$f(n) = \begin{cases} 10f(n-1) + 10^{n-1} & n > 1 \\ 1 & n = 1 \end{cases}$$

由此可知， $f(n) = n10^{n-1}$ 。

据此，可从高位向低位进行统计，再减去多余的 0 的个数即可。

1-2 字典序问题。

问题描述：在数据加密和数据压缩中常需要对特殊的字符串进行编码。给定的字母表 A 由 26 个小写英文字母组成，即 $A=\{a, b, \cdots, z\}$ 。该字母表产生的升序字符串是指字符串中字母从左到右出现的次序与字母在字母表中出现的次序相同，且每个字符最多出现 1 次。例如，a、b、ab、bc、xyz 等字符串都是升序字符串。现在对字母表 A 产生的所有长度不超过 6 的升序字符串按照字典序排列并编码如下。

1	2	...	26	27	28	...
a	b	...	z	ab	ac	...

对于任意长度不超过 6 的升序字符串，迅速计算出它在上述字典中的编码。

算法设计：对于给定的长度不超过 6 的升序字符串，计算它在上述字典中的编码。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行是一个正整数 k ，表示接下来有 k 行。在接下来的 k 行中，每行给出一个字符串。

结果输出：将计算结果输出到文件 output.txt。文件有 k 行，每行对应一个字符串的编码。

输入文件示例

input.txt

2

a

b

输出文件示例

output.txt

1

2

分析与解答：考察一般情况下长度不超过 k 的升序字符串。

设以第 i 个字符打头的长度不超过 k 的升序字符串个数为 $f(i, k)$ ，长度不超过 k 的升序字符串总个数为 $g(k)$ ，则 $g(k) = \sum_{i=1}^{26} f(i, k)$ 。易知

$$f(i, 1) = 1$$

$$g(1) = \sum_{i=1}^{26} f(i, 1) = 26$$

$$f(i, 2) = \sum_{j=i+1}^{26} f(j, 1) = 26 - i$$

$$g(2) = \sum_{i=1}^{26} f(i, 2) = \sum_{i=1}^{26} (26 - i) = 325$$

一般情况下有

$$f(i, k) = \sum_{j=i+1}^{26} f(j, k-1)$$

$$g(k) = \sum_{i=1}^{26} f(i, k) = \sum_{i=1}^{26} \sum_{j=i+1}^{26} f(j, k-1)$$

据此可计算出每个升序字符串的编码。

1-3 最多约数问题。

问题描述：正整数 x 的约数是能整除 x 的正整数。正整数 x 的约数个数记为 $\text{div}(x)$ 。例如，1、2、5、10 都是正整数 10 的约数，且 $\text{div}(10)=4$ 。设 a 和 b 是 2 个正整数， $a \leq b$ ，找出 a 和 b 之间约数个数最多的数 x 。

算法设计：对于给定的 2 个正整数 $a \leq b$ ，计算 a 和 b 之间约数个数最多的数。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行有 2 个正整数 a 和 b 。

结果输出：若找到的 a 和 b 之间约数个数最多的数是 x ，则将 $\text{div}(x)$ 输出到文件 output.txt。

输入文件示例

input.txt

输出文件示例

output.txt

分析与解答：设正整数 x 的质因子分解为

$$x = p_1^{N_1} p_2^{N_2} \cdots p_k^{N_k}$$

则

$$\text{div}(x) = (N_1+1)(N_2+1)\cdots(N_k+1)$$

搜索区间 $[a, b]$ 中数的质因子分解。

primes()函数用于产生质数：

```
void primes(){
    bool get[MAXP+1];
    for(int i=2; i <= MAXP; i++)
        get[i]=true;
    for(i=2; i <= MAXP; i++)
        if(get[i]) {
            int j=i+i;
            while(j <= MAXP) {
                get[j]=false;
                j += i;
            }
        }
    for(int ii=2, j=0; ii <= MAXP; ii++)
        if(get[ii])
            prim[++j]=ii;
}
```

search()函数用于搜索最多约数：

```
void search(int from,int tot,int num,int low,int up) {
    if(num >= 1)
        if((tot > max) || ((tot == max) && (num < numb))) {
            max=tot;
            numb=num;
        }
    if((low == up) && (low > num))
        search(from, tot*2, num*low, 1, 1);
    for(int i=from; i <= PCOUNT; i++) {
        if(prim[i] > up)
            return;
        else {
            int j = prim[i], x = low1, y = up, n = num, t = tot, m = 1;
            while(true) {
                m++;
                t += tot;
                x /= j;
                y /= j;
                if(x == y)
                    break;
                n *= j;
                search(i+1, t, n, x+1, y);
            }
        }
    }
}
```

```

    }
}
m = 1 << m;
if(tot < max/m)
    return;
}
}

```

实现算法的主函数如下。

```

int main() {
    primes();
    cin >> l >> u;
    if((l == 1) && (u == 1)) {
        max = 1;
        numb = 1;
    }
    else {
        max = 2;
        numb = 1;
        search(1, 1, 1, 1, u);
    }
    cout << max << endl;
    return 0;
}

```

1-4 金币阵列问题。

问题描述：有 $m \times n$ ($m \leq 100$, $n \leq 100$) 枚金币在桌面上排成一个 m 行 n 列的金币阵列。每枚金币或正面朝上或背面朝上。用数字表示金币状态，0 表示金币正面朝上，1 表示金币背面朝上。

金币阵列游戏的规则是：① 每次可将任一行金币翻过来放在原来的位置上；② 每次可任选 2 列，交换这 2 列金币的位置。

算法设计：给定金币阵列的初始状态和目标状态，计算按金币游戏规则，将金币阵列从初始状态变换到目标状态所需的最少变换次数。

数据输入：由文件 input.txt 给出输入数据。文件中有多组数据。文件的第 1 行有 1 个正整数 k ，表示有 k 组数据。每组数据的第 1 行有 2 个正整数 m 和 n 。以下 m 行是金币阵列的初始状态，每行有 n 个数字表示该行金币的状态，0 表示正面朝上，1 表示背面朝上。接着的 m 行是金币阵列的目标状态。

结果输出：将计算出的最少变换次数按照输入数据的次序输出到文件 output.txt。相应数据无解时，输出 -1。

输入文件示例

input.txt

2

4 3

1 0 1

0 0 0

1 1 0

输出文件示例

output.txt

2

-1

```

1 0 1
1 0 1
1 1 1
0 1 1
1 0 1
4 3
1 0 1
0 0 0
1 0 0
1 1 1
1 1 0
1 1 1
0 1 1
1 0 1

```

分析与解答：枚举初始状态每列变换为目标状态第 1 列的情况。算法描述如下。

```

int k, n, m, count, best;
int b0[Size+1][Size+1], b1[Size+1][Size+1], b[Size+1][Size+1];
bool found;
int main() {
    cin >> k;
    for(int i=1; i <= k; i++) {
        cin >> n >> m;
        for(int x=1; x <= n; x++) {
            for(int y=1; y <= m; y++)
                cin >> b0[x][y];
            for(x=1; x <= n; x++)
                for(int y=1; y <= m; y++)
                    cin >> b1[x][y];
            acpy(b, b1);
            best = m+n+1;
            for(int j=1; j <= m; j++) {
                acpy(b1, b);
                count = 0;
                trans2(1, j);
                for(int p=1; p <= n; p++)
                    if(b0[p][1] != b1[p][1])
                        trans1(p);
                for(p=1; p <= m; p++) {
                    found = false;
                    for(int q=p; q <= m; q++) {
                        if(same(p, q)) {
                            trans2(p, q);
                            found = true;
                            break;
                        }
                    }
                    if(!found)
                        break;
                }
            }
        }
    }
}

```



```

    }
    if(found && count < best)
        best = count;
    }
    if(best < m+n+1)
        cout << best << endl;
    else
        cout << 1 << endl;
    }
}
return 0;
}

```

其中，trans1()函数模拟行翻转变换，trans2()函数模拟列交换变换。

```

void trans1(int x) {
    for(int i=1; i <= m; i++)
        b1[x][i] = b1[x][i]^1;
    count++;
}

void trans2(int x, int y) {
    for(int i=1; i <= n; i++)
        Swap(b1[i][x], b1[i][y]);
    if(x != y)
        count++;
}

bool same(int x, int y) {
    for(int i=1; i <= n; i++)
        if(b0[i][x] != b1[i][y])
            return false;
    return true;
}

void acpy(int a[Size+1][Size+1], int b[Size+1][Size+1]) {
    for(int i=1; i <= n; i++)
        for(int j=1; j <= m; j++)
            a[i][j] = b[i][j];
}

```

1-5 最大间隙问题。

问题描述：最大间隙问题：给定 n 个实数 x_1, x_2, \dots, x_n ，求这 n 个数在实轴上相邻两个数之间的最大差值。假设对任何实数的下取整函数耗时 $O(1)$ ，设计解最大间隙问题的线性时间算法。

算法设计：对于给定的 n 个实数 x_1, x_2, \dots, x_n ，计算它们的最大间隙。

数据输入：输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行有 1 个正整数 n 。接下来的 1 行中有 n 个实数 x_1, x_2, \dots, x_n 。

结果输出：将找到的最大间隙输出到文件 output.txt。

输入文件示例

input.txt

5

输出文件示例

output.txt

3.2

分析与解答：用鸽舍原理设计最大间隙问题的线性时间算法如下。

```
double maxgap(int n, double *x) {
    double minx = x[mini(n, x)], maxx = x[maxi(n, x)];
    // 用 n-2 个等间距点分割区间[minx, maxx], 产生 n-1 个桶
    // 每个桶的 i 中用 high[i] 和 low[i] 分别存储分配给桶 i 的数中的最大数和最小数
    int *count = new int[n+1];
    double *low = new double[n+1];
    double *high = new double[n+1];
    // 桶初始化
    for(int i=1; i <= n1; i++) {
        count[i] = 0;
        low[i] = maxx;
        high[i] = minx;
    }
    // 将 n 个数置于 n-1 个桶中
    for(i=1; i <= n; i++) {
        int bucket = int ((n-1)*(x[i]-minx) / (maxx-minx))+1;
        count[bucket]++;
        if(x[i] < low[bucket])
            low[bucket] = x[i];
        if(x[i] > high[bucket])
            high[bucket] = x[i];
    }
    // 此时, 除了 maxx 和 minx 的 n-2 个数被置于 n-1 个桶中。由鸽舍原理即知, 至少有一个桶是空的
    // 这意味着最大间隙不会出现在同一桶中的两个数之间对每个桶做一次线性扫描即可找出最大间隙
    double tmp = 0, left = high[1];
    for(i=2; i <= n1; i++) {
        if(count[i]) {
            double thisgap = low[i]-left;
            if(thisgap > tmp)
                tmp = thisgap;
            left = high[i];
        }
    }
    return tmp;
}
```

其中, mini()函数和 maxi()函数分别计算数组中最小元素和最大元素的下标。

```
template<class T>
int mini(int n, T* x) {
    T tmp = x[1];
    for(int i=1, k=1; i <= n; i++) {
        if(x[i] < tmp) {
            tmp = x[i];
            k = i;
        }
    }
}
```

```

    return k;
}
template<class T>
int maxi(int n, T* x) {
    T tmp = x[1];
    for(int i=1, k=1; i <= n; i++) {
        if(x[i] > tmp) {
            tmp = x[i];
            k = i;
        }
    }
    return k;
}

```

由于下取整函数耗时 $O(1)$ ，故循环体内的运算耗时 $O(1)$ 。因此，整个算法耗时 $O(n)$ 。即算法 `maxgap` 是求最大间隙问题的线性时间算法。注意到在代数判定树计算模型下， $\Omega(n \log n)$ 是最大间隙问题的一个计算时间下界。这意味着在代数判定树的计算模型下，最大间隙问题是不可能有线性时间算法的。在此题中假设下取整函数耗时 $O(1)$ ，实际上这可以看作是在代数判定树模型中，将下取整运算作为基本运算增加到原有的基本运算集中，从而使代数判定树计算模型的计算能力得到增强。因而可以在线性时间内解最大间隙问题。



计算机算法设计与分析习题解答（第5版）

本书是与“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析（第5版）》配套的辅助教材和国家精品课程教材，分别对主教材中的算法分析题和算法实现题给出了解答或解题思路提示。为了提高学生灵活运用算法设计策略解决实际问题的能力，本书还将主教材中的许多习题改造成算法实现题，要求学生设计出求解算法并上机实现。本书教学资料包含各章算法实现题、测试数据和答案，可在华信教育资源网免费注册下载。

本书内容丰富，理论联系实际，可作为高等学校计算机科学与技术、软件工程、信息安全、信息与计算科学等专业本科生和研究生学习计算机算法设计的辅助教材，也是工程技术人员和自学者的参考书。

提升学生“知识—能力—素质”	体现“基础—技术—应用”内容
把握教学“难度—深度—强度”	提供“教材—教辅—课件”支持

相关图书：《计算机算法设计与分析（第5版）》 ISBN 978-7-121-34439-8



策划编辑：章海涛
责任编辑：章海涛
封面设计：张 昱

ISBN 978-7-121-34438-1



9 787121 344381 >

定价：56.00 元