HZ BOOKS
华章教育

本书用于
ACM/ICPC、Google Code Jam、TopCoder
百度之星程序设计大赛
等程序设计竞赛的训练

算法设计编程实验

大学程序设计课程与竞赛训练教材

Algorithm Design Experiment

for Collegiate Programming Contest and Education

吴永辉 王建德 编著

机械工业出版社
China Machine Press

算法设计编程实验

大学程序设计课程与竞赛训练教材

Algorithm Design Experiment

for Collegiate Programming Contest and Education

吴永辉 王建德 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

算法设计编程实验：大学程序设计课程与竞赛训练教材 / 吴永辉，王建德编著. —北京：机械工业出版社，2013.6

ISBN 978-7-111-42383-6

Ⅰ. 算… Ⅱ. ①吴… ②王… Ⅲ. 电子计算机－算法设计－高等学校－教材 Ⅳ. TP301.6

中国版本图书馆 CIP 数据核字（2013）第 096220 号

本书以知识体系结构、思维方式与解题策略为主线，分 8 章分别介绍 Ad Hoc、模拟法、数论、组合分析、贪心法、动态规划方法、高级数据结构、计算几何的编程实验。每个章节由实验范例和题库两个部分组成，试题全部选自 ACM 国际大学生程序设计竞赛以及其他各类程序设计竞赛，共 234 题（3 题为一题多解），并给出了试题来源和在线测试地址。每个实验范例都有详尽的试题解析和标有注释的参考程序，而题库中的所有试题无论难易，都有清晰的提示。另外，华章网站中还给出了本书所有试题的英文原版描述和大部分试题的测试数据。

本书既可以作为大专院校计算机专业算法课程的教材，也可以作为计算机专业学生的研修资料和程序设计竞赛的培训教材。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：李 荣

藁城市京瑞印刷有限公司印刷

2013 年 6 月第 1 版第 1 次印刷

185mm × 260mm · 29.25 印张

标准书号：ISBN 978-7-111-42383-6

定 价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

前　言

2012 年，我们在机械工业出版社出版了《数据结构编程实验：大学程序设计课程与竞赛训练教材》一书。现在，我们又编著了《算法设计编程实验：大学程序设计课程与竞赛训练教材》，旨在为计算机学科的核心课程——算法与数据结构提供配套的实验教材，为全国各大学的 ACM 程序设计竞赛活动提供系列化的培训资料，并为从事计算机软硬件研发或系统自学算法的人员提供丰富、实用的参考手册。

全书以知识单元为基本构件，各单元既保持循序渐进的顺序又相对独立，既可拆卸重组、各取所需，又可在此基础上推广或创新，便于各学校按照不同的层次要求组织教学和培训活动。

全书共分 8 章：

第 1 章求解 Ad Hoc 类问题的编程实验：介绍机理分析法和统计分析法，引导读者在没有相应经典和模式化算法的情况下，学会自创简单的算法。

第 2 章模拟法的编程实验：引导读者如何按照题意设计数学模型的各种参数，观察变更这些参数所引起过程状态的变化，并在此基础上展开算法设计。

第 3 章数论的编程实验和第 4 章组合分析的编程实验：这两章凸显了数论和组合分析知识在算法中的应用。其中第 3 章围绕初等数论中素数运算、求解不定方程和同余方程、应用积性函数等问题展开实验。第 4 章引导读者在编程求解组合类问题时如何计算具有某种特性的对象个数，如何将它们完全列举出来；如何使用抽屉原理解决存在性问题；如何使用容斥原理计算多个集合并的元素数；如何使用波利亚定理对一个问题的各种不同的组合状态计数。

第 5 章贪心法的编程实验和第 6 章动态规划（DP）方法的编程实验：在求解具备最优子结构特征的问题时，这两种方法是最常用、最经典的，但适用场合不同，既有相同点又有不同之处。

第 7 章高级数据结构的编程实验：选择在一般数据结构教材中没有出现但很有用的一些知识，例如后缀数组、线段树、欧拉图、哈密尔顿图、最大独立集、割点、桥和双连通分支等内容，展开编程实验。

第 8 章计算几何的编程实验：计算几何学是算法体系中的一个重要组成部分，也是先前算法教材中最薄弱的环节。该章将展开介绍点线面运算、扫描线算法、计算半平面交、计算凸包和旋转卡壳算法等实验。

全书以上述 8 个知识主线为架构，以问题驱动和启发式引导为主要方式。每个章节都有实验范例，启发性地引出相应的算法设计思想；每个实验范例不仅有知识要点阐述和详尽的试题解析，而且还列出了标有详细注释的参考程序；每个常用的算法都以经典模型为支撑，每个定理、概念都有详尽的说明和推导，并使用大量图表增强直观性和可读性。本书尽力将经典模型推广到具有相同性质的一类问题中，让读者解决一个经典模型就相当于解决一类问题。有时还有意在经典模型外适当加入其他因素，引导读者对被套用的经典模型进行适当修改，加入独特的属性，或者运用已知模型或方法来分析信息原型的属性，在此基础上创造出具有新意的模型

或方法来。

“算法设计编程实验”是一门实践出真知的课程，需要大量引入案例教学，通过模拟或者重现现实生活中的一些场景，让读者把自己置入问题情境之中，通过思考、讨论和上机编程来学习算法。为此，本书别出心裁、独辟蹊径，将 ACM 国际大学生程序设计竞赛和其他程序设计竞赛中的典型试题分门别类，精选了其中 234 道试题（3 题为一题多解），翻译后作为学习算法设计的实验案例。其中 83 道试题作为各章节的实验范例，151 道试题列入各章节的题库。正如上面所说，每个实验范例都有详尽的试题解析和标有注释的参考程序，而题库中的所有试题无论难易，都有清晰的提示。教师既可以将实验范例作为“算法设计编程实验”课程的教学素材，也可以从相关题库中挑选试题作为“算法设计编程实验”课程的作业、考题或 ACM 集训活动的培训资料。每道题都注明了试题来源和在线测试地址，提供了程序测试的明确去处。这些试题既为学生学习算法的相关知识设立了丰富、有趣的问题背景，使得他们能够带着问题学，而且自编程序的正确性和效率也可以通过相关网站上的测试系统得到实时检验，达到学以致用的目的。此外，本书提供了所有试题的英文原版描述、大部分试题的测试数据等资料，有需要者可登录华章网站（http://www.hzbook.com）下载。

需要说明的是，本书是在复旦大学 ACM 集训队长期活动的基础上积累而成的，刘宇达、宋凯强、黄云南、符汉杰、袁梓峰、于竟成、钱雨露、沈晨曦、文琪、李星、金阳华等同学精心编写了所有程序，每道程序都通过了严格的测试验证，其中一些程序经多次修改，精益求精。这些同学为本书的出版付出了辛勤的劳动，做出了不可或缺的贡献。在此，向他们表示由衷的感谢。

由于时间和水平所限，书中肯定夹杂了不少缺点和错误，表述不当和笔误更是在所难免，热忱欢迎学术界同仁或读者赐正。如果你在阅读中发现了问题，请通过书信或电子邮件告诉我们，以便及时整理成勘误表放在本书的专门网站上，供广大读者查询更正。我们更期望读者对本书提出建设性意见，以便再版时改进。我们的联系方式是：

通信地址：上海市邯郸路 220 号复旦大学计算机科学技术学院　吴永辉

邮编：200433

E-mail：yhwu@fudan.edu.cn

王建德　吴永辉

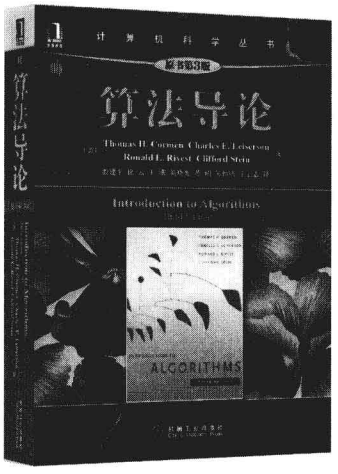
2013 年 2 月 20 日于上海

注：

本书试题的在线测试地址主要如下表所示：

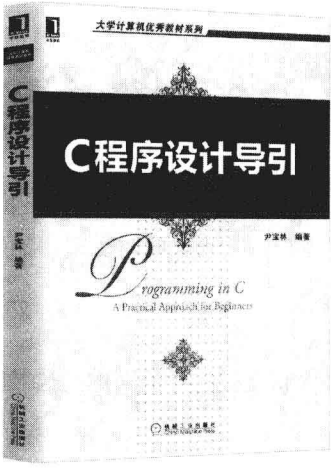
在线评测系统	简称	网址
北京大学在线评测系统	POJ	http://poj.org/
浙江大学在线评测系统	ZOJ	http://acm.zju.edu.cn/onlinejudge/
UVA 在线评测系统	UVA	http://uva.onlinejudge.org/ http://livearchive.onlinejudge.org/
Ural 在线评测系统	Ural	http://acm.timus.ru/
SGU 在线评测系统	SGU	http://acm.sgu.ru/

推荐阅读



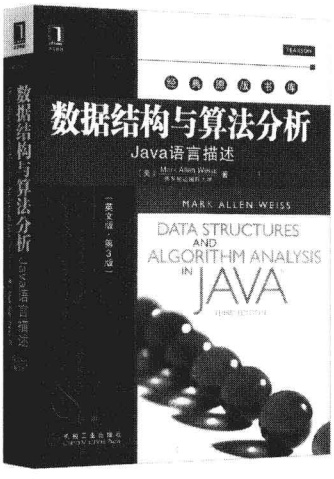
算法导论（原书第3版）

作者：Thomas H.Cormen 等 ISBN：978-7-111-40701-0 定价：128.00元



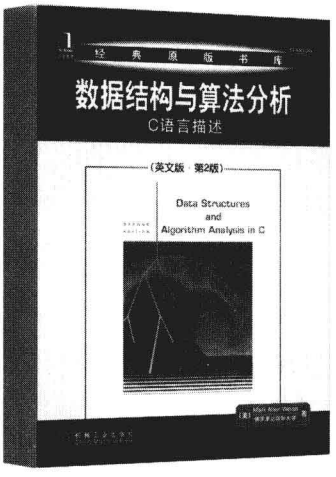
C程序设计导引

作者：尹宝林 ISBN：978-7-111-41891-7 定价：35.00元



数据结构与算法分析
——Java语言描述（英文版·第3版）

作者：Mark Allen Weiss ISBN：978-7-111-41236-6 定价：79.00元



数据结构与算法分析
——C语言描述(英文版·第2版)

作者：Mark Allen Weiss ISBN：978-7-111-31280-2 定价：45.00元

目　录

前言

第 1 章　求解 Ad Hoc 类问题的编程实验 … 1
1.1　机理分析法的实验范例 …… 1
1.2　统计分析法的实验范例 …… 5
1.3　相关题库 …… 10
第 2 章　模拟法的编程实验 …… 35
2.1　直叙式模拟的实验范例 …… 36
2.2　筛选法模拟的实验范例 …… 44
2.3　构造法模拟的实验范例 …… 51
2.4　相关题库 …… 55
第 3 章　数论的编程实验 …… 69
3.1　素数运算的实验范例 …… 69
3.1.1　使用筛法生成素数的实验范例 … 69
3.1.2　测试大素数的实验范例 …… 76
3.2　求解不定方程和同余方程的实验范例 …… 81
3.2.1　计算最大公约数和不定方程 …… 81
3.2.2　计算同余方程和同余方程组 …… 85
3.3　积性函数的实验范例 …… 91
3.3.1　使用欧拉函数 $\varphi(n)$ 计算与 n 互质的正整数个数 …… 92
3.3.2　使用莫比乌斯函数 $\mu(n)$ 计算非平方数 n 的质因子个数 …… 97
3.4　相关题库 …… 102
第 4 章　组合分析的编程实验 …… 118
4.1　生成排列组合的实验范例 …… 118
4.1.1　按字典序思想生成下一排列组合 …… 118
4.1.2　按字典序思想生成所有的排列组合 …… 121
4.2　排列组合计数的实验范例 …… 122
4.2.1　一般的排列组合计数公式 …… 123
4.2.2　两种特殊的排列组合计数公式 … 126
4.3　容斥原理与抽屉原理的实验范例 …… 132
4.3.1　利用抽屉原理求解存在性问题 … 132
4.3.2　利用容斥原理对并集计数 …… 134
4.4　波利亚定理的实验范例 …… 140
4.4.1　波利亚定理的概念基础 …… 141
4.4.2　利用波利亚定理计算集合在置换群作用下产生的等价类个数 …… 148
4.5　相关题库 …… 157
第 5 章　贪心法的编程实验 …… 165
5.1　体验贪心法内涵的实验范例 …… 165
5.2　利用数据有序化进行贪心选择的实验范例 …… 172
5.3　在综合性的 P 类问题中使用贪心法的实验范例 …… 181
5.4　相关题库 …… 187
第 6 章　动态规划（DP）方法的编程实验 …… 197
6.1　线性 DP 的实验范例 …… 198
6.1.1　初步体验线性 DP 问题 …… 198
6.1.2　子集和问题 …… 202
6.1.3　最长公共子序列问题 …… 203
6.1.4　最长递增子序列问题 …… 206
6.2　树形 DP 的实验范例 …… 213
6.3　状态压缩 DP 的实验范例 …… 218
6.4　单调优化 1D/1D DP 的实验范例 …… 224
6.4.1　经典模型 1：利用决策代价函数 w 的单调性优化 …… 224
6.4.2　经典模型 2：利用决策区间下界的单调性优化 …… 228
6.4.3　经典模型 3：利用最优决策点的凸性优化 …… 233
6.5　相关题库 …… 236
第 7 章　高级数据结构的编程实验 …… 273
7.1　后缀数组的实验范例 …… 273
7.1.1　使用倍增算法计算名次数组和后缀数组 …… 273

7.1.2 计算最长公共前缀 …… 276
7.1.3 后缀数组的应用 …… 278
7.2 线段树的实验范例 …… 288
7.2.1 线段树的基本概念和基本操作 … 288
7.2.2 线段树单点更新的维护 …… 290
7.2.3 线段树子区间更新的维护 …… 293
7.3 处理特殊图的实验范例 …… 306
7.3.1 计算欧拉图 …… 306
7.3.2 计算哈密尔顿图 …… 314
7.3.3 计算最大独立集 …… 324
7.3.4 计算割点、桥和双连通分支…… 327
7.4 相关题库 …… 336
第8章 计算几何的编程实验 …… 354
8.1 点线面运算的实验范例 …… 354
8.1.1 计算点积和叉积 …… 354
8.1.2 计算线段交 …… 361
8.1.3 利用欧拉公式计算多面体 …… 371
8.2 利用扫描线算法计算矩形的面积并 … 375
8.2.1 沿垂直方向计算矩形的面积并 … 375
8.2.2 沿水平方向计算矩形的面积并 … 380
8.3 计算半平面交的实验范例 …… 383
8.3.1 计算半平面交的联机算法 …… 384
8.3.2 利用极角计算半平面交的算法 … 390
8.4 计算凸包和旋转卡壳的实验范例…… 398
8.4.1 计算凸包 …… 399
8.4.2 旋转卡壳实验 …… 403
8.5 相关题库 …… 408

第1章

求解Ad Hoc类问题的编程实验

正如现实世界的事物是多姿多彩、千变万化的一样，在解题中经常会出现一些不能套用简单的条条框框和现成模式，需要独立思考、见解独创和有所创新的非标准题。这类试题被称作Ad Hoc类试题（Ad Hoc源自于拉丁语，意思是“为每种目的而”）。其特征是不能简单地对应经典算法，也没有模式化的求解方法，需要编程者自己构建算法来解答试题。由于算法自创，因此能够比较综合地反映编程者的智慧、知识基础和创造性思维的能力。当然，自创的算法只针对问题本身，探索其独有性质，是一种专为解决某个特定的问题或完成某项特定的任务而设计的解决方案，因此一般不具备普适意义和可推广性。

求解Ad Hoc类问题的方法多样，但按照数理分析和思维方式的角度，大致可分两大类：

1）机理分析法，采用顺向思维方式，从分析内部机理出发顺推算法。

2）统计分析法，采用逆向思维方式，从分析部分解出发倒推算法。

这两种方法不是孤立和排斥的，在求解Ad Hoc类问题的过程中，既可以根据需要选择其一，也可以两者兼用。

下面展开机理分析法和统计分析法的编程实验。

1.1 机理分析法的实验范例

根据客观事物的特性，分析其内部的机理，弄清关系，在适当抽象的条件下，得到可以描述事物属性的数学工具。

经过数学分析，如果能够抽象出Ad Hoc类问题的内在规律，则可以采用机理分析法建立数学模型，然后根据模型的原理对应到算法，编程实现，通过执行算法得到问题解，如图1.1-1所示。

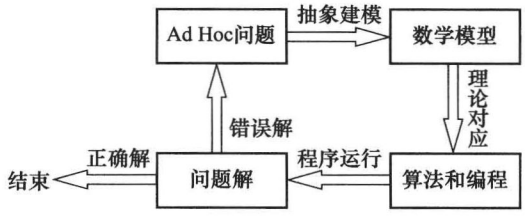


图 1.1-1

机理分析法的核心是数学建模，即使用适当的数学思想建立起模型；或者提取问题中的有效信息，用简明的方式表达其规律。需要注意的是：

1）选择的模型必须尽量多地体现问题的本质特征。但这并不意味着模型越复杂越好，累赘的信息会影响算法效率。

2）模型的建立不是一个一蹴而就的过程，而是要经过反复地检验和修改，在实践中不断完善。

3）数学模型通常有严格的格式，但程序编写形式可不拘一格。

机理分析法是一个复杂的数据抽象过程。我们要善于透视问题的本质，寻找突破口，进而选择适当的模型。模型的构造过程可以帮助我们认识问题，不同的模型从不同的角度反映问题，可以引发不同的思路，起到引导发散思维的作用。但认识问题的最终目的是解决问题，模型的固有性质虽可帮我们建立算法，其优劣也可通过时空复杂度等指标来分析和衡量，但最终

还是以程序的运行结果为标准。所以模型不是一成不变的，同样要通过各种技术不断优化。模型的产生虽然是人脑思维的产物，但它仍然是客观事物在人脑中的反映。所以要培养良好的建模能力，还必须依靠在平时的学习中积累丰富的知识和经验。

下面举两个实验范例。

【1.1.1 Factstone Benchmark】

【问题描述】

Amtel 已经宣布，到2010 年，它将发行128 位计算机芯片；到2020 年，它将发行256 位计算机；等等，Amtel 坚持每持续十年将其字大小翻一番的战略。（Amtel 于2000 年发行了64 位计算机，在1990 年发行了32 位计算机，在1980 年发行了16 位计算机，在1970 年发行了8 位计算机，并首先在1960 年发行了4 位计算机。）

Amtel 将使用新的标准检查等级（Factstone）来宣传其新芯片大大提高的能力。Factstone 等级被定义为这样的最大整数 n，使得 $n!$ 可以表示为一个计算机字中的无符号整数（比如1960 年的4 位计算机可表示 $3!=6$，而不能表示 $4!=24$）。

给出一个年份 $1960 \leqslant y \leqslant 2160$，Amtel 最近发布的芯片的 Factstone 等级是什么？

输入：

给出若干测试用例。每个测试用例一行，给出年份 y。在最后一个测试用例后，给出包含0 的一行。

输出：

对于每个测试用例，输出一行，给出 Factstone 等级。

样例输入	样例输出
1960 1981 0	3 8

试题来源：Waterloo local 2005.09.24

在线测试地址：POJ 2661，UVA 10916

试题解析

对于给定的年份，先求出当年 Amtel 处理器的字大小，然后计算出最大的 n 的值，使得 $n!$ 成为一个符合字的大小的无符号整数。

在1960 年，字的大小是4 位，以后每十年字的大小翻一番，这就意味着，y 年字的位数为 $k=2^{2+\left\lfloor\frac{y-1960}{10}\right\rfloor}$。能放在 k 位中最大的无符号整数是 2^k-1。如果 $n!$ 为不大于 2^k-1 的最大正整数，则 n 为 y 年芯片的 Factstone 等级。计算方法有两种：

方法1：直接求不大于 2^k-1 的最大正整数 $n!$，这种方法极容易溢出且速度慢。

方法2：采用对数计算，即根据

$$\log_2 n! = \log_2 n + \log_2(n-1) + \cdots + \log_2 1 \leqslant \log_2(2^k-1) < k$$

计算 n。显然这种方法的效率比前者要好许多。具体实现如下：

计算 y 年字的位数 k，累加 $\log_2 i$（i 从1 出发，每次加1），直到数字超过 k 为止。此时，$i-1$ 即为 Factstone 等级。

程序清单

```
#include <stdio.h>
#include <math.h>

int y,Y,i,j,m;                          // 年份为 y
double f,w;                             // y 年字的位数为 w,log2 i 的累加值为 f

main(){
   while (1 == scanf("%d",&y) && y){    // 输入年份 y
      w = log(4);                       // 按照每十年字的大小翻一番的规律,计算 y 年字的位数 w
      for (Y=1960; Y<=y; Y+=10){
         w *= 2;
      }
      i = 1;                            // 累加 log2 i(每次 i 加 1),直到数字超过 w
      f = 0;
      while (f < w) {
         f += log((double) ++i);
      }
      printf("%d\n",i-1) ;              // 输出 Factstone 等级
   }
   if (y) printf("fishy ending %d\n",y);
}
```

【1.1.2 Bridge】

【问题描述】

n 个人要在晚上过桥，在任何时候最多两人一组过桥，每组要有一只手电筒。在这 n 个人中只有一只手电筒可以用，因此要安排以某种往返的方式来返还手电筒，使得更多的人可以过桥。

每个人的过桥速度不同，每组的速度由速度较慢的成员所决定。请确定一个策略，让 n 个人用最少的时间过桥。

输入：

输入的第一行给出 n，接下来的 n 行给出每个人的过桥时间，不会超过1000人，且没有人的过桥时间超过100秒。

输出：

输出的第一行给出所有 n 个人过桥的总的秒数，接下来的若干行给出实现策略。每行包含一个或两个整数，表示组成一组过桥的一个人或两个人（每个人用其在输入中给出的过桥所用的时间来标识。虽然许多人的过桥时间相同，但即使有混淆，对结果也没有影响）。这里要注意的是过桥也有方向性，因为要返还手电筒让更多的人通过。如果用时最少的策略有多个，则任意一个都可以。

样例输入	样例输出
4 1 2 5 10	17 1 2 1 5 10 2 1 2

试题来源：Waterloo local 2000.09.30

在线测试地址：POJ 2573，ZOJ 1877，UVA 10037

试题解析

稍动脑筋，便可以得出一个简单的逻辑：要使得 n 个人用最少的时间过桥，慢的成员必须借助快的成员传递手电筒。

由于一次过桥最多两人且手电筒需要往返传递，因此以两个成员过桥为一个分析单位，计算过桥时间。我们按过桥时间递增的顺序将 n 个成员排序。设当前序列中，

- A 是最快的人，B 是次快的人，A 和 B 是序列首部的两个元素。
- a 是最慢的人，b 是次慢的人，a 和 b 是序列尾部的两个元素。

有两种过桥方案：

方案 1：用最快的成员 A 传递手电筒帮助最慢的 a 和 b 过桥。

如果带一个最慢的成员，则所用的时间是 a + A(a 表示最快和最慢的两个成员到对岸所需的时间，而 A 是最快的成员返回所需的时间)。显然带两个最慢的成员所用的时间是 2 * A + a + b。

方案 2：用最快的成员 A 和次快的成员 B 传递手电筒帮助最慢的 a 和 b 过桥。

步骤 1：A 和 B 到对岸，所用时间为 B。

步骤 2：A 返回，将手电筒给最慢的 a 和 b，所用时间为 A。

步骤 3：a 和 b 到对岸，所用时间为 a；到对岸后，他们将手电筒交给 B。

步骤 4：B 需要返回原来的岸边，因为要交还手电筒，所需时间为 B。

所以，需要的总时间为 2 * B + A + a。

显然，最慢的 a 和 b 要用最少的时间过桥，只能借助 A 或者 A 和 B 传递手电筒过桥，其他方法都会增加过桥时间。哪一种过桥方案更有效？比较一下就行了：

如果 (2 * A + a + b < 2 * B + A + a)，则采用第 1 种方案，即用最快的成员 A 传递手电筒；否则采用第 2 种方案，即用最快的成员 A 和次快的成员 B 传递手电筒 (2 * A + a + b < 2 * B + A + a 等价于 b + A < 2 * B)。

我们每次帮助最慢的两个成员过桥 (n -= 2)，累计每个最佳过桥方案的时间。最后产生两种可能情况：

1) 对岸剩下 2 个队员 (n == 2)，全部过桥，即累计时间为 B。

2) 对岸剩下 3 个队员 (n == 3)，用最快的成员传递手电筒，帮助最慢的成员过桥，然后与次慢的成员一起过桥，即累计时间为 a + A + b。

程序清单

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
using  namespace std;
int n,i,j,k,a[111111];                    // 人数为 n,每个人的速度存储于序列 a[]
int ans=0;                                // n 个人过桥的总时间初始化
int main () {
```

```
    scanf("%d",&n);                          // 输入每个人的速度
    for(i=1;i<=n;i++)scanf("%d",a+i);
    if(n==1){                                // 输出1个人的过桥方案
        printf("%d\n%d\n",a[1],a[1]);return 0;
    }
    int nn=n;
    sort(a+1,a+n+1);                         // 按照速度递增顺序排序
    while(n>3){                              // 统计n个人过桥的总时间
      if(a[1]+a[n-1]<2*a[2]){                // 累计用a[1]传递手电筒帮助最慢的2个成员过桥
                                             // 所需的时间
          ans+=a[n]+a[1]*2+a[n-1];
      }else{                                 // 累计用a[1]a[2]传递手电筒帮助最慢的2个成员
                                             // 过桥所需的时间
          ans+=a[2]+a[1]+a[2]+a[n];
        }
        n-=2;                                // 两个最慢的成员过桥
    }
    if(n==2)ans+=a[2];                       // 对岸剩下2个成员,累计其过桥的时间
    else  ans+=a[1]+a[2]+a[3];               // 对岸剩下3个成员,累计其过桥的时间
    printf("%d\n",ans);                      // 输出n个人过桥的总时间
    n=nn;
    while(n>3){                              // 输出每组人过桥所用的时间

        if(a[1]+a[n-1]<2*a[2])               // 输出用a[1]传递手电筒的过桥方案
            printf("%d %d\n%d\n%d %d\n%d\n",a[1],a[n],a[1],a[1],a[n-1],a[1]);
        else                                 // 输出用a[1]和a[2]传递手电筒的过桥方案
            printf("%d %d\n%d\n%d %d\n%d\n",a[1],a[2],a[1],a[n-1],a[n],a[2]);
        n-=2;                                // 两个最慢的成员过桥
    }
    if(n==2)printf("%d %d\n",a[1],a[2]);     // 剩下2个队员过桥,输出过桥方案
    else                                     // 剩下3个队员过桥,输出过桥方案
        printf("%d %d\n%d\n%d %d\n",a[1],a[3],a[1],a[1],a[2]);
    return 0;
}
```

1.2　统计分析法的实验范例

在一时得不到事物的特征机理的情况下，我们可先通过手算或编程等方法测试得到一些数据（即问题的部分解），再利用数理统计知识对数据进行处理，从而得到最终的数学模型。

图1.2-1给出了统计分析法的大致流程：先从Ad Hoc问题的原型出发，通过手工或简单的程序得到问题的部分解（即解集A），然后运用数理统计方法，通过部分解得到问题原型的主要属性（大部分属性是规律性的东西），从而建立数学模型，最后通过算法设计和编程得到问题的全部解（即全集I）。这里需要注意的是：

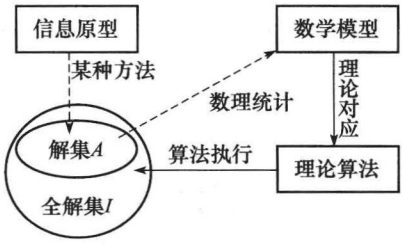


图　1.2-1

1）因为有时候根本无法求出问题的部分解，或者无法用数理统计知识分析部分解，所以求部分解的过程和对部分解进行数理统计的过程画的是虚线，表示不是每个信息原型都能用统计分析法建模。

2）所有模型对应的算法是将盲目搜索排除在外的。因为盲目搜索是从全集I出发求解集A的，这违背了建模的目的。我们所讨论的统计分析法，是在对全解集I的子集A进行数理统计的基础上建立数学模型，所以盲目搜索不属于统计分析法的范畴。

3）一般来说，我们可以先采用机理分析法进行分析；如果机理分析进行不下去，再考虑使用统计分析法。当然，如果问题容易找到部分简单解，我们亦可优先考虑统计分析法。事实上，机理分析所得出的某些结论，往往可有效地运用于统计分析法；而统计分析法得出的某些规律，最终需要通过机理分析验证其准确性。所以这两者彼此并不是孤立的。我们在建模的时候完全可以两者兼用。

【1.2.1 Ants】

【问题描述】

一支蚂蚁军队在长度为 l 厘米的横竿上走，每只蚂蚁的速度恒定，为1厘米/秒。当一只行走的蚂蚁到达横竿终点的时候，它就立即掉了下去；当两只蚂蚁相遇的时候，它们就调转头，并开始往相反的方向走。我们知道蚂蚁在横竿上原来的位置，但不知道蚂蚁行走的方向。请计算所有蚂蚁从横竿上掉下去的最早可能的时间和最晚可能的时间。

输入：

输入的第一行给出一个整数，表示测试用例个数。每个测试用例首先给出两个整数：横竿的长度（以厘米为单位）和在横竿上的蚂蚁的数量 n。接下来给出 n 个整数，表示每只蚂蚁在横竿上从左端测量过来的位置，没有特定的次序。所有输入数据不大于1 000 000，数据间用空格分隔。

输出：

对于输入的每个测试用例，输出用一个空格分隔的两个数，第一个数是所有的蚂蚁掉下横竿的最早可能的时间（如果它们的行走方向选择合适），第二个数是所有的蚂蚁掉下横竿的最晚可能的时间。

样例输入	样例输出
2 10 3 2 6 7 214 7 11 12 7 13 176 23 191	4 8 38 207

试题来源：Waterloo local 2004.09.19

在线测试地址：POJ 1852，ZOJ 2376，UVA 10714

试题解析

蚂蚁数的上限为1 000 000，爬行方向共有 $2^{1\,000\,000}$ 种，这是一个天文数字，因此不可能真的去逐一枚举蚂蚁的方向。

我们先研究蚂蚁少的时候的一些情况（见图1.2-2）。

显然，蚂蚁愈多变化愈多，情况愈复杂。其实解题的瓶颈就是蚂蚁相遇的情况。假如我们拘泥于“对于相遇如何处理”这个细节，将陷入无从着手的窘境。

假如出现这样一种情况：蚂蚁永远不会相遇（即所有向左走的蚂蚁都在向右走的蚂蚁的左边），那么很容易找出 $O(n)$ 的算法。

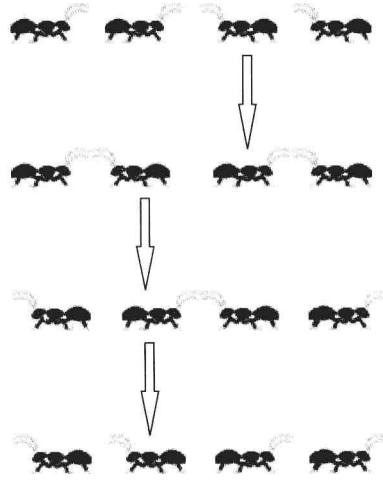


图 1.2-2

我们回过头观察前面给出的那个例子。我们发现相遇

只会发生在出现“  ”时，相遇后就变成了“  ”，这就相当于忽略了“相遇”这一事件。也就是说，我们可以假设这些蚂蚁即使相遇了也不理睬对方而继续走自己的路。对于问题来说，所有的蚂蚁都是一样的，并无相异之处，因此这个假设是合理的。这样，每只蚂蚁掉落所用时间就只有两个取值：一个是向左走用的时间，一个是向右走用的时间。全部掉落的最早时间就是每只蚂蚁尽快掉落用时的最大值，因为这些蚂蚁现在互不干扰。同理，全部掉落的最迟时间就是每只蚂蚁尽量慢掉落用时的最大值。由此得出算法，设

l_i 为蚂蚁 i 在横竿上从左端过来测量的位置（$1 \leqslant i \leqslant n$）；little 为 N 只蚂蚁掉下横竿的最早可能时间；big 为 N 只蚂蚁掉下横竿的最晚可能的时间。

$$\text{little} = \min_{1 \leqslant i \leqslant n} \{l_i, L - l_i\}, \quad \text{big} = \max_{1 \leqslant i \leqslant n} \{l_i, L - l_i\}$$

本题从最简单的情况入手，通过分析发现所有蚂蚁的等价性，将“相遇后转向”转变为“相遇互不干扰”，从而简化了问题，可以轻而易举地计算出答案。

程序清单

```
#include <stdio.h>
int c,big,little,L,i,j,k,n;                          // 测试用例数为 c;big、little 分别为最晚时间
                                                     // 和最早时间;横竿长度为 L;竿上的蚂蚁数为 n
main(){
   scanf("%d",&c);                                   // 输入测试用例数
   while (c-- && (2 == scanf("%d%d",&L,&n))) {       // 输入横竿长度和横竿上的蚂蚁数
      big = little = 0;                              // 最晚时间和最早时间初始化
      for (i=0;i<n;i++) {                            // 输入每只蚂蚁的测量位置
          scanf("%d",&k);
          if (k > big) big = k;                      // 根据 k 的左方长度和右方长度调整最晚时间
          if (L-k > big) big = L-k;
          if (k > L-k) k = L-k;                      // 由 k 左右方长度的最小值调整最早时间
          if (k > little) little = k;
      }
      printf("%d %d\n",little,big);                  // 输出所有蚂蚁掉下横竿的最早时间和最晚时间
   }
   if (c != -1) printf("missing input\n");
}
```

【1.2.2　Matches Game】

【问题描述】

有一个简单的游戏。在这个游戏中，有若干堆火柴和两个玩家。两个玩家一轮一轮地玩。在每一轮中，一个玩家可以选择一个堆，并从该堆取走任意根火柴（当然，取走火柴的数量不可能为0，也不可能大于所选的火柴的数量）。如果一个玩家取了火柴后，没有火柴留下，那么这个玩家就赢了。假设两个玩家非常聪明，请告诉大家是否先玩的玩家会赢。

输入：

输入由若干行组成，每行一个测试用例。每行开始首先给出整数 $M(1 \leqslant M \leqslant 20)$，表示火柴堆的堆数；然后给出 M 个不超过 10 000 000 的正整数，表示每个火柴堆的火柴数量。

输出：

对每个测试用例，如果是先手的玩家赢，则在一行中输出"Yes"；否则输出"No"。

样例输入	样例输出
2 45 45 3 3 6 9	No Yes

试题来源：POJ Monthly，readchild

在线测试地址：POJ 2234

试题解析

先考查最简单的情况：

1）如果游戏开始时只有一堆火柴，则游戏人Ⅰ通过取走所有的火柴而获胜。

2）如果游戏开始时有两堆火柴，且火柴数量分别为 N_1 和 N_2。

游戏人取得胜利并不在于 N_1 和 N_2 的值具体是多少，而是取决于它们是否相等。

- 若 $N_1 \neq N_2$，则游戏人Ⅰ从大堆中取走火柴使得两堆火柴数量相等，于是游戏人Ⅰ以后每次取子的数量与游戏人Ⅱ相等而最终获胜。
- 若 $N_1 = N_2$，则游戏人Ⅱ只要按着游戏人Ⅰ取子的数量在另一堆中取相等数量的火柴，最终获胜者将会是游戏人Ⅱ。

这样，两堆的取子获胜策略就已经找到了。现在的问题是，如何从两堆的取子策略扩展到任意堆数中呢？

3）任意堆数。

首先来回忆一下，每个正整数都有一个对应的二进制数，例如：$57_{(10)} = 111\ 001_{(2)}$，即 $57_{(10)} = 2^5 + 2^4 + 2^3 + 2^0$。于是，我们可以认为每一堆火柴数由 2 的幂数的子堆组成。这样，含有 57 枚火柴的大堆就能看成是分别由数量为 2^5、2^4、2^3、2^0 的 4 个子堆组成。

现在考虑各大堆大小分别为 N_1，N_2，…，N_k 的取子游戏。将每一个数 N_i 表示为其二进制数（数的位数相等，不等时在前面补 0）：

$$N_1 = a_s \cdots a_1 a_0$$

$$N_2 = b_s \cdots b_1 b_0$$

$$\cdots$$

$$N_k = m_s \cdots m_1 m_0$$

如果每一种大小的子堆数都是偶数，我们就称取子游戏是平衡的，而对应位相加是偶数的称为平衡位，否则称为非平衡位。因此取子游戏是平衡的，当且仅当

$$a_s + b_s + \cdots + m_s \text{ 是偶数}$$

$$\cdots$$

$$a_1 + b_1 + \cdots + m_1 \text{ 是偶数}$$

$$a_0 + b_0 + \cdots + m_0 \text{ 是偶数}$$

这里之所以提出取子游戏平衡的概念，是因为当一方处于平衡状态（非平衡状态）时，对方总能够通过下一轮取子使之变成非平衡状态（平衡状态）。而赢方的最后状态是 $s+1$ 位都为平衡位（全 0）。于是，我们猜想出一个获胜策略：

游戏人Ⅰ能够在非平衡取子游戏中取胜，而游戏人Ⅱ能够在平衡的取子游戏中取胜。

下面通过统计分析方法证明上述猜想。先以一个两堆火柴的取子游戏作为试验：

设游戏开始时游戏处于非平衡状态。这样，游戏人Ⅰ就能通过一种取子方式（见下面的归

纳）使得他取子后留给游戏人Ⅱ的是一个平衡状态下的游戏，接着无论游戏人Ⅱ如何取子，再留给游戏人Ⅰ的一定是一个非平衡状态下的游戏（只有二进制情形下能达到这种必然的转变，因为任何取子均会改变一个或多个位置上的比特值，而任何一个比特值的改变要么是0变1要么是1变0，这都会改变奇偶性。但是在多进制情形下，某一位置上2变0或3变1就不会改变平衡性。换句话说，平衡性只在二进制下定义，定义多进制下的平衡性没有意义），如此反复进行，当游戏人Ⅱ在最后一次平衡状态下取子后，游戏人Ⅰ便能一次性取走所有的火柴而获胜。而如果游戏开始时游戏处于平衡状态，那根据上述方式取子，最终游戏人Ⅱ能获胜。

接下来，应用此获胜策略来考虑4堆火柴的取子游戏。其中各堆的大小分别为7，9，12，15枚火柴，用二进制表示各数分别为：0111，1001，1100和1111，于是可得到表1.2-1。

表 1.2-1

堆的规模	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
大小为7的堆	0	1	1	1
大小为9的堆	1	0	0	1
大小为12的堆	1	1	0	0
大小为15的堆	1	1	1	1
平衡性	非平衡位	非平衡位	平衡位	非平衡位

由取子游戏的平衡条件可知，此游戏是一个非平衡状态的取子游戏，因此，游戏人Ⅰ在按获胜策略进行取子游戏下将一定能够取得最终的胜利。具体做法有多种，游戏人Ⅰ可以从大小为12的堆中取走11枚火柴，使得游戏达到平衡（如表1.2-2）。

表 1.2-2

堆的规模	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
大小为7的堆	0	1	1	1
大小为9的堆	1	0	0	1
大小为12的堆	0	0	0	1
大小为15的堆	1	1	1	1

游戏人Ⅰ将非平衡状态转为平衡状态的方法是：将某一行的非平衡位取反并使得取反之后该行的值小于取反之前的值。取子个数为取反前后数值之差。

之后，无论游戏人Ⅱ如何取子，游戏人Ⅰ在取子后仍使得游戏达到平衡。

同样的道理，游戏人Ⅰ也可以选择大小为9的堆并取走5枚火柴而剩下4枚，或者游戏人Ⅰ从大小为15的堆中取走13枚而留下2枚。

归根结底，取子游戏的关键在于游戏开始时游戏处于何种状态（平衡或非平衡）和第一个游戏人是否能够按照取子游戏的获胜策略来进行游戏。由此得出算法：

n堆火柴数量对应的n个二进制数，连续进行$n-1$次异或运算。若结果非0，则说明存在非平衡位，先手的玩家赢；若结果为0，则说明所有位平衡，后手的玩家赢。

程序清单

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <string>
#include <cmath>
```