普通高等教育“十一五”国家级规划教材

教育部“高等学校教学质量与教学改革工程”立项项目

俞经善 鞠成东 主编

ACM程序设计竞赛基础教程

（第2版）

计算机科学与技术专业实践系列教材

清华大学出版社

普通高等教育"十一五"国家级规划教材

计算机科学与技术系列教材

ACM程序设计竞赛基础教程

（第2版）

俞经善 鞠成东 主编

清华大学出版社
北京

内 容 简 介

本书以循序渐进的方式对 ACM 程序设计竞赛中所涉及的基本题型和知识点进行了综合的介绍。全书共分 10 章，包括基础知识讲解、典型题目分析和算法设计，每道例题均给出了完整的源程序作为参考。内容涵盖了基础算法、数据结构、字符串、搜索、图论、动态规划、组合数学和初等数论等。

本书内容全面，针对性强，言简意赅，讲解透彻，通俗易懂，图例丰富，所有源代码均可进行评测。本书作为 ACM 程序设计竞赛的培训教程，不仅为大学生提供了竞赛入门的指导，而且对参赛学生拓展解题思路和提高训练水平也有很大的帮助。本书也可供喜爱程序设计的学生以及从事算法设计的技术人员学习参考。

图书在版编目(CIP)数据

ACM 程序设计竞赛基础教程/俞经善，鞠成东主编. --2 版. --北京：清华大学出版社，2016
计算机科学与技术专业实践系列教材
ISBN 978-7-302-44607-1

Ⅰ. ①A…　Ⅱ. ①俞… ②鞠…　Ⅲ. ①程序设计－竞赛－高等学校－教材　Ⅳ. ①TP311.1

中国版本图书馆 CIP 数据核字(2016)第 175450 号

责任编辑：张瑞庆
封面设计：傅瑞学
责任校对：焦丽丽
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62795954

印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：18.5　　**字　　数**：448 千字
版　　次：2010 年 10 月第 1 版　2016 年 11 月第 2 版　　**印　　次**：2016 年 11 月第 1 次印刷
印　　数：1～2000
定　　价：39.00 元

产品编号：070652-01

序

从1970年开始，ACM/ICPC赛事就影响着计算机与信息专业的许多大学生，引导着他们应用计算机技术展示自己分析问题解决问题的才能。哈尔滨工程大学于2005年开始积极投身于ACM/ICPC活动中。时至今日，令我欣慰的是，从仅有的几名程序设计爱好者到如今上百人参与集训的规模，我们的校代表队已形成了一套自己的训练方法。

在多年的磨炼中，我看到我们的队员走了一些弯路，也经历了一些波折。经过不断的尝试、努力，以及与其他高校参赛选手积极的交流，他们渐渐成长起来。今天，他们把自己的经验编辑成为一本系统的教材，既是对自己多年来训练学习的总结，也为了使更多的ACM/ICPC爱好者有“据”可依。

哈尔滨工程大学举办过多种形式的区域ACM/ICPC赛事，包括省级区域赛、东北四省区域赛、亚洲区域赛等。2010年，主办了第34届ACM/ICPC全球总决赛。这是我们举办的一次最高等级的ACM/ICPC赛事。赛事获得了圆满成功，得到了ACM/ICPC组委会以及世界各国参赛队员的一致好评。作为东北地区组委会，我们每年都会积极地举办省赛、东北地区赛以及其他各种赛事。所谓的不进沙场，一切皆为纸上谈兵。校代表队要在各种赛事中磨炼才能更加坚毅。

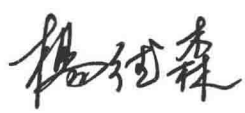
ACM/ICPC赛事活动是大学生自己的活动。大学生通过这些活动全面提高了自己的能力。从组织训练、参加比赛、赛队管理到举办比赛，都有学生参与。我们希望看到在一个十分自由的学术氛围里，学生可以放开手脚，自主地学习、创新。在ACM/ICPC竞赛中，我们能够得到的不仅仅是算法上的知识，还有学生的动手能力、创造能力以及团队合作和组织能力等。这种能够全面培养学生各方面能力的赛事实属不多。我们希望借助这样的赛事，来提高学生的综合素质和专业技能。

“教书育人”四个掷地有声的字是我们永远的教学宗旨。我们时常会思索，应该教什么样的书，育什么样的人。从多年的国际教育研讨会上，不难发现，众多世界顶级的高等院校纷纷将创新型人才培养作为教育的重中之重，这和我们的教学理念是不谋而合的。几年来，我校ACM代表队不仅在各种ACM赛事中收获了奖牌，而且在各种科技创新活动中也非常活跃。他们开发的在线考试系统和Online Judge系统，在实际使用中表现出先进的设计思想、良好的人机环境和可靠的软件质量。

希望本书的出版可以引起更多程序设计爱好者的兴趣，可以成为ACM/ICPC参赛队员道路上的一块垫脚石。

第34届ACM/ICPC全球总决赛执行主席

哈尔滨工程大学副校长



前　言

本书第1版自2010年8月出版以来，承蒙各个高校计算机程序设计爱好者的厚爱，给予了支持和肯定。根据读者反映，本书对学生参与ACM/ICPC竞赛训练、算法思维培养及程序设计实践，起到了有益的指导作用。

近年来，随着ACM/ICPC赛事在中国的不断发展，国内高校的参与度和竞赛成绩不断攀升。究其原因在于ACM/ICPC竞赛训练能够综合、全面、系统地培养学生的算法思维和设计能力。通过赛事训练，能够将计算机语言类课程以及数据结构、算法设计与分析、离散数学、组合数学、具体数学、数论等众多专业课程进行很好的融合，对学生的专业培养和教育大有益处。

自2005年开始，哈尔滨工程大学ACM/ICPC代表队便开始有针对性的训练，并于2007年秋季将ACM/ICPC实践形式引入本科教学，开设了ACM/ICPC竞赛入门课程。2010年，在哈尔滨工程大学成功举办了第34届ACM/ICPC全球总决赛。2014年秋季，哈尔滨工程大学开始探索创新的教学理念、教学模式以及教学考核评价方法，将ACM/ICPC竞赛形式和在线评测平台引入C语言、数据结构与算法设计和具体数学等课程的教学环节中，使理论与实践高效贯通，众多相关课程深度融合。无论是历年赛事训练，还是日常教学实践，我们都深感有必要对本书进行重新修订和完善，以使低年级学生更容易上手学习，使高年级学生能够站在更高起点进行思维训练。为此，根据历年经验积累，我们精心筛选补充了各章节例题。这些例题设计精巧，对学生能力培养和竞赛训练具有很好的指导作用。本次修订始终本着“实用、管用、够用”的原则，在尽量保持原版特色、组织结构和内容体系不变的前提下，对例题做了大量的筛选工作，同时对解题思路等内容进行了精心的设计和编写，既有利于启发，又留有余地，便于推动学生的思维培养。

在本书第2版的修订编写过程中，我们参阅了国内多位专家、学者的相关著作或译著，也参考了国内各高校ACM网站资源，在此对他们表示崇高的敬意和衷心的感谢！为本书的修订做出重要贡献的有宋旭、袁茂洵、岳盈、罗心语、刘晓冬等同学，他们对文字的整理和程序的调试做了大量辛苦的工作，在此一并表示感谢。

限于编者水平，本书在内容取舍、编写方面难免存在欠妥之处，恳请专家、同行和读者批评指正，我们将不胜感激！

作　者

2016年7月

目　录

第1章 基础算法

1.1 分治算法

分治算法的基本思想是将一个规模为 N 的问题分解为 K 个规模较小的子问题，这些子问题相互独立且与原问题性质相同。求出子问题的解，就可得到原问题的解。

例 1-1 计数问题

1. 题目描述

给定两个整数 a 和 b，计算出 1 在 a 和 b 之间出现的次数。例如，如果 $a=1024$，$b=1032$，那么 a 和 b 之间的数就是：

1024 1025 1026 1027 1028 1029 1030 1031 1032

则有 10 个 1 出现在这些数中。

输入：

输入不会超过 500 行。每一行有两个整数 a 和 b，a 和 b 的范围是 $0<a,b<100\ 000\ 000$。输入两个 0 时程序结束，两个 0 不作为输入样例。

输出：

对于每一对输入的 a 和 b，输出一个数，代表 1 出现的个数。

样例输入：

```
1 10
44 497
346 542
1199 1748
1496 1403
1004 503
1714 190
1317 854
1976 494
1001 1960
0 0
```

样例输出：

```
2
185
40
666
113
105
```

```
1133
512
1375
1256
```

2. 解题思路

本题要求出 1 在两个整数 a 和 b 之间出现的次数。可以由分治算法的思想，先求出 1 在 0～a 之间出现的次数，再求出 1 在 0～b 之间出现的次数，然后两者相减即可。现在的问题转换为如何求出 1 在 0～a 之间出现的次数。

将 0～197 的数列出来后可以看出以下规律：

① 可以求出 1 在 190～197 之间出现的次数，然后对于 0～189，1 在个位数上出现了 1 次。

② 个位考虑完后，直接考虑 197/10－1(即 18)中 1 出现的次数，同时考虑到，数字减小了，每一位的权值会增加，也就是说每一个数字出现的次数会增加 10 倍。例如，现在的 1，是原来 10～19 之间的所有的 1，即权值变为原来的 10 倍。

3. 参考程序

```
/*本算法计算出 0~9 在 a 和 b 之间分别出现的次数,取答案时直接取 d[1]即可*/
#include<iostream>
using namespace std;
/*********************************************/
const int N=11;
int d[N];                                   //d[N]中存储数字 0~9 分别出现的次数
int value;                                  //记录相应的权值变化
void deal(int n);
/*********************************************/
void deal(int n)
{
    if(n<=0)return;
    int one, ten;                           //one,ten 分别表示个位和十位
    one=n%10;
    n /=10;
    ten=n;

    for(int i=0; i<=one; i++)               //将个位上出现的数统计下来
        d[i] +=value;
    while(ten)
    {
        d[ten%10] +=(one+1)*value;
        ten /=10;
    }
    for(int i=0; i<10; i++)
        d[i] +=value*n;
    d[0] -=value;                           //将第一位是 0 的情况排除
```

```
    value *=10;                          //权值变化,变为原来的 10 倍
    deal(n-1);
}
/*****************************************/
int main()
{
    int a, b;
    while(cin >>a >>b)
    {
        if(a==0 && b==0)break;
        if(a<b){int tmp=b; b=a; a=tmp;}  //将较大值存入 a 中,较小值存入 b 中
        for(int i=0; i<10; i++)          //初始化操作
            d[i]=0;
        /*处理过程*/
        value=1;
        deal(a);
        value=-1;                  //此处 value=-1 是为了求出最后的答案 deal(a)-deal(b)
        deal(b-1);
        /*输出结果*/
        cout <<d[1] <<endl;
    }

//    system("pause");
      return 0;
}
```

例 1-2　查找等式的解

1. 题目描述

有 3 个数列 A、B 和 C，有一个数 X。计算是否存在 3 个整数 A_i、B_j、C_k，使得公式 $A_i+B_j+C_k=X$ 成立。

输入：

有多组测试样例。每组测试数据描述如下：第一行是 3 个整数 L、M 和 N；第二行有 L 个整数表示数列 A；第三行有 N 个整数表示数列 B；第四行有 M 个整数表示数列 C；第五行是一个整数 S，表示有 S 个需要被计算的整数 X。所有整数都是 32 位整数($1\leqslant L, N, M\leqslant 500, 1\leqslant S\leqslant 1000$)。

输出：

对于每组测试数据，首先应该按如下形式"Case d:"打印情况数；接着对于 S 次查询，需要计算是否能够满足等式，如果满足，就打印"YES"，否则打印"NO"。具体格式见样例。

样例输入：

```
3 3 3
1 2 3
1 2 3
```

```
1 2 3
3
1
4
10
```

样例输出：

```
Case 1:
NO
YES
NO
```

2. 解题思路

本题实际上是一个比较容易求解的题。首先，把 $a+b$ 的情况全部列出来，放在 sum[] 数组中(数组大小为 500×500)。然后，枚举 $s-c[i]$在 sum[]中是否存在，用二分查找。

3. 参考程序

```
//Can you find it?.cpp
#include <ios>
#include <map>
#include <set>
#include <list>
#include <cmath>
#include <ctime>
#include <deque>
#include <queue>
#include <stack>
#include <bitset>
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <cstdio>
#include <cwchar>
#include <iosfwd>
#include <limits>
#include <locale>
#include <memory>
#include <string>
#include <vector>
#include <cassert>
#include <ciso646>
#include <climits>
#include <clocale>
#include <complex>
#include <csetjmp>
#include <csignal>
```

```
#include <cstdarg>
#include <cstddef>
#include <cstdlib>
#include <cstring>
#include <cwctype>
#include <fstream>
#include <iomanip>
#include <istream>
#include <numeric>
#include <ostream>
#include <sstream>
#include <utility>
#include <iostream>
#include <iterator>
#include <valarray>
#include <algorithm>
#include <exception>
#include <stdexcept>
#include <streambuf>
#include <functional>
#define LL long long
#define lson l, m, rt<<1
#define rson m+1, r, rt<<1|1
#define PI 3.1415926535897932626
#define EXIT exit(0);
#define PAUSE system("pause");
#define DEBUG puts("Here is a BUG");
#define SYNC_CLOSE ios::sync_with_stdio(false);
#define CLEAR(name, init)memset(name, init, sizeof(name))
const double eps=1e-8;
const int MAXN=(int)5e2+5;
using namespace std;

int sum[MAXN*MAXN];

int main(int argc, char const *argv[]){
#ifndef ONLINE_JUDGE

    freopen("D:\\Documents\\Disk_Synchronous\\Programs\\Acm\\input.txt", "r",
    stdin);
#endif
    int l, n, m;
    int kase=1;
    while(~scanf("%d%d%d", &l, &n, &m)){
        int a[MAXN], b[MAXN], c[MAXN];
```

```
        for(int i=0; i <l; i++){
            scanf("%d", a+i);
        }
        for(int i=0; i <n; i++){
            scanf("%d", b+i);
        }
        for(int i=0; i <m; i++){
            scanf("%d", c+i);
        }
        int k=0;
        for(int i=0; i <l; i++){
            for(int j=0; j <n; j++){
                sum[k++]=a[i]+b[j];
            }
        }
        sort(sum, sum+k);
        /* for(int i=0; i <k; i++){
            cout <<sum[i] <<" ";
        }
        puts(""); */
        printf("Case %d:\n", kase++);
        int t;
        scanf("%d", &t);
        while(t--){
            int s;
            scanf("%d", &s);
            bool isfind=false;
            for(int i=0; i <m; i++){
                int tmp=s-c[i];
                int l=0, r=k-1;
                while(l<=r){
                    int m=(l+r)>>1;
                    if(tmp <sum[m])r=m-1;
                    else if(tmp >sum[m])l=m+1;
                    else { isfind=true; break; }
                }
            }
            if(isfind)puts("YES");
            else puts("NO");
        }
    }
    return 0;
}
```

例 1-3　解方程

1. 题目描述

给出等式 $8x^4+7x^3+2x^2+3x+6==Y$，能在 x 属于[0,100]之间找到方程的解吗？

输入：

输入的第一行包含一个整数 $T(1\leqslant T\leqslant 100)$，表示测试用例的数目。接着有 T 行，每行有一个实数 $Y(|Y|\leqslant 1e10)$；

输出：

对于每组测试用例，只能输出一个实数，表示方程的解，如果在 0～100 之间没有解，则输出“No solution!”；否则，输出实数解 x（精确到小数点后 4 位）。

样例输入：

```
2
100
-4
```

样例输出：

```
1.6152
No solution!
```

2. 解题思路

根据方程可以看出该函数在 0～100 的定义域内单调递增，即满足二分的性质，故二分枚举 x，即可得到方程的解。

3. 参考程序

```
#include <cstdlib>
#include <iostream>

using namespace std;

const double d=1e-8;

bool solve(double m,double y)
{
    if(y-(8*m*m*m*m+7*m*m*m+2*m*m+3*m+6)>0)
        return true;
    else return false;
}

int main()
{
    int n;
    double l,r,m,y;
    scanf("%d",&n);
```

```
    while(n--)
    {
        l=0;
        r=100;
        scanf("%lf",&y);
        if(y<6||y>807020306)printf("No solution!\n");
        else
        {
            while(r-l>d)
            {
                m=(l+r)/2;
                if(solve(m,y))l=m;
                else r=m;
            }
            printf("%.4lf\n",l);
        }
    }
    return 0;
}
```

1.2 递归算法

程序调用自身的编程技巧称为递归(recursion)。

一个过程或函数在其定义或说明中又直接或间接地调用自身的一种方法,通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解。递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算,大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说,递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时,递归前进;当边界条件满足时,递归返回。

例 1-4 汉诺塔问题

1. 题目描述

汉诺塔(又称河内塔)问题其实是印度的一个古老的传说。

开天辟地的神勃拉玛(和中国的盘古差不多的神)在一个庙里留下了 3 根金刚石的柱子,第一根柱子上面套着 64 个圆的金片,最大的一个金片在底下,其余金片一个比一个小,依次叠上去。庙里的众僧不倦地把第一根柱子上的金片一个一个地搬到第三根柱子上,规定:可利用中间的第二根柱子来帮助,但每次只能搬一个金片,而且大的金片不能放在小的金片上面。移动圆片的次数的计算结果非常恐怖:18 446 744 073 709 551 615,众僧们即便是耗尽毕生精力也不可能完成金片的移动。

输入:

输入一个正整数 n,表示有 n 个盘片在第一根柱子上。

输出:

输出操作序列，格式为 move t from x to y。每个操作一行，表示把 x 柱子上的编号为 t 的盘片挪到 y 柱子上。柱子编号为 a、b、c，要用最少的操作把所有的盘子从 a 柱子上转移到 c 柱子上。

样例输入：

```
3
```

样例输出：

```
move 1 from a to c
move 2 from a to b
move 1 from c to b
move 3 from a to c
move 1 from b to a
move 2 from b to c
move 1 from a to c
```

2. 解题思路

其实算法非常简单，当盘子的个数为 n 时，移动的次数应等于 2^n-1（有兴趣的读者可以自己进行证明）。后来一位美国学者发现一种出人意料的简单方法，只要轮流进行两步操作就可以实现。首先把 3 根柱子按顺序排成“品”字型，把所有的圆盘按从大到小的顺序放在柱子 a 上，根据圆盘的数量确定柱子的排放顺序：若 n 为偶数，按顺时针方向依次摆放 a、b、c；若 n 为奇数，按顺时针方向依次摆放 a、c、b。

① 按顺时针方向把圆盘 1 从现在的柱子移动到下一根柱子，即当 n 为偶数时，若圆盘 1 在柱子 a，则把它移动到 b；若圆盘 1 在柱子 b，则把它移动到 c；若圆盘 1 在柱子 c，则把它移动到 a。

② 接着，把另外两根柱子上可以移动的圆盘移动到新的柱子上。即把非空柱子上的圆盘移动到空柱子上，当两根柱子都非空时，移动较小的圆盘。这一步没有明确规定移动哪个圆盘，你可能以为会有多种可能性，其实不然，可实施的行动是唯一的。

③ 反复进行①和②操作，最后就能按规定完成汉诺塔的移动。

所以，结果非常简单，就是按照移动规则向一个方向移动金片。

例如，3 阶汉诺塔的移动为：$a\rightarrow c$，$a\rightarrow b$，$c\rightarrow b$，$a\rightarrow c$，$b\rightarrow a$，$b\rightarrow c$，$a\rightarrow c$

汉诺塔问题也是程序设计中的经典递归问题，下面将给出递归和非递归的不同实现源代码。

3. 参考代码

```
#include <fstream>
#include <iostream>
using namespace std;
void Move(int n,char x,char y)
{
cout<<"move "<<n<<" from "<<x<<" to "<<y<<endl;
}
void Hannoi(int n,char a,char b,char c)
```