信息学奥林匹克竞赛指导丛书

吴文虎 主编

信息学奥林匹克竞赛指导

—— 1999—2000竞赛试题解析

吴文虎 王建德 编著

✓ 训练思维能力

✓ 提高解题技巧

清华大学出版社

信息学奥林匹克竞赛指导丛书
吴文虎　主编

信息学奥林匹克竞赛指导

——1999—2000竞赛试题解析

吴文虎　王建德　编著

清华大学出版社
北京

内 容 简 介

本书收集了1994—2000年国际国内信息学(计算机)奥林匹克竞赛试题,重点对1999—2000年试题进行了算法分析,力求得到最佳算法。

这些试题有相当难度,对训练综合思维能力、提高解题技巧大有裨益。本书可供大学、中学编程爱好者学习和参考。

图书在版编目(CIP)数据

信息学奥林匹克竞赛指导.1999—2000竞赛试题解析/吴文虎主编;吴文虎,王建德编著.—北京:清华大学出版社,2003
(信息学奥林匹克竞赛指导丛书)
ISBN 7-302-06226-9

Ⅰ.信… Ⅱ.①吴… ②吴… ③王… Ⅲ.计算机课—中小学—解题 Ⅳ.G634.675

中国版本图书馆CIP数据核字(2003)第064451号

出 版 者:清华大学出版社　　地　　址:北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编:100084
社 总 机:010-62770175　　客户服务:010-62776969

责任编辑:宋　方
印 刷 者:北京鑫丰华彩印有限公司
发 行 者:新华书店总店北京发行所
开　　本:787×1092 1/16　印张:20.25　字数:464千字
版　　次:2003年7月第1版　2003年7月第1次印刷
书　　号:ISBN 7-302-06226-9/TP·3727
印　　数:1～3000
定　　价:29.00元

前　言

国际信息学奥林匹克(International Olympiad in Informatics,IOI)从1989年到2002年,14年赛事的健康发展得益于联合国教科文组织(UNESCO)为这项赛事所做的准确定位:通过竞赛形式对有才华的青少年起到激励作用,促其能力得以发展;让青少年彼此建立联系,推动经验交流,给学校这一类课程增加活力;建立起教育工作者与专家档次上的国际联系,推进学术思想的交流。概括起来说,就是启迪思路,激励英才,发展学科,促进交流。

学科奥林匹克是智力与能力的竞赛,注重考查全面素质与创造能力。从这个意义上讲,信息学奥林匹克活动是素质教育的一个大课堂。在我国,每年国家集训队都要将"怎样做人,怎样做事,怎样求知和怎样健体"的指导思想纳入培训计划。这14年中国队共派出参赛选手55人次,累计获金牌29块、银牌15块、铜牌11块,届届名列前茅,正是因为坚持了全面素质教育的指导思想,把造就高素质有创造精神的人才作为活动的定位目标。

回顾14年的竞赛可以看出,参加高手云集的这种世界大赛是有相当难度的,第一,没有大纲,赛题范围没有界定,谁也无法去猜测每年的主办国会出什么类型的难题;第二,计算机科学与技术发展很快,层出不穷的新思路和新成果会反映到试题中来;第三,所要解决的试题往往涉及图论、组合数学、人工智能等大学开设的课程知识;第四,比较短的给定解题时间与刁难的测试数据让选手必须拿出高超和精巧的解法,无论在时间上还是空间上都是优化的解法才能取得高分。有许多赛题没有固定的现成的解法,选手要在比赛现场凭借实力,理出思路,构建数学模型,写出算法,编出程序,运行并验证整个构思是否正确,出解的时间是否能达到题目的要求,等等。可以看出,在这一过程中最重要的是要有创造能力。为激发创新精神,培养创造能力,就需要树立新的教育观念和教学方法,还要利用现代化的教学手段。引导学生学用电脑,在使用中帮助开发人脑,这可能是信息学奥林匹克活动的最重要的一个特点。我认为在这项活动中应该培养学生的四种能力:自学能力;实践动手能力;创新能力;上网获取信息,并能区分有用信息和无用信息的能力。这样做的结果使许多选手不但有能力在世界赛场上拿金牌,也有能力在学校的学习中名列前茅。

信息学奥林匹克十余年涌现出一大批出类拔萃的计算机后备人才,在他们的带动下,我国的青少年在普及计算机的大潮中阔步前进,取得了可喜的成绩。历史已雄辩地证明:计算机的普及就是要从娃娃做起,这是"科教兴国"、中华崛起的需要。为了提高普及的层次,编写竞赛辅导教材是十分必要的,也是广大青少年电脑爱好者所盼望的。这里我们将历年国际国内大赛的试题集中起来进行剖析,重点讲解题思路与方法。但是必须说明,书

中的解法仅起抛砖引玉的作用。

青少年是国家的希望，不断提高青少年的科学素养是中华民族永远昂首屹立在世界东方的根基所在。“精心育桃李，切望青胜蓝”是我和王建德老师的共同心愿。

国际信息学奥林匹克中国队总教练
清华大学计算机系教授博士生导师
吴文虎

2002 年 10 月

目　录

第 1 章　1999 年国际奥林匹克信息学竞赛中国组队赛试题解析

1.1　01　统　计

试题

近来 IOI 的专家们在进行一项有关二进制数的研究，研究涉及的一个统计问题令他们大伤脑筋。问题是这样的：

对于一个自然数 n，可以把它转换成对应的二进制$\overline{a_k a_{k-1}\cdots a_1 a_0}$，其中：

$n=a_k\times 2^k+a_{k-1}2^{k-1}\cdots+a_1\times 2^1+a_0$，其中，$a_i=0$ 或 $1(0\leqslant i<k)$，$a_k=1$。

如：$10=\overline{1010}$；$5=\overline{101}$。我们统计一下 $a_0\sim a_k$ 这 $k+1$ 个数中 0 的个数和 1 的个数。如果在这 $k+1$ 个数中，0 的个数比 1 的个数多，就称 n 为 A 类数。

现在的任务是，对于一个给定的自然数 m，求 $1\sim m$ 中 A 类数的个数。

输入

输入文件中只有一个自然数 $m(1\leqslant m\leqslant 10^{30})$。

输出

输出文件也只有一个自然数：$1\sim m$ 中 A 类数的个数。

输入输出示例：

输入文件	输出文件
3	0

算法分析

一、整型数组的运算

试题中自然数 m 的有效数位为 31 位$(1\leqslant m\leqslant 10^{30})$，而计算机中整数有效位(包括符号位)充其量也不过 11 位，无论如何也达不到试题的数位要求。无奈，只能采用整型数组 a 来替代自然数 m。设

```
type
    atype=Array[0..31]of integer;      { 整型数组类型 }
var
    a:atype;                           { a[0]—整型数组 a 的长度。m= Σ(i=1..a[0]) a[i]*10^(i-1) }
```

我们以字符串 S 输入自然数 m，并通过下述方法将其转换为整型数组 a：

```
a[0]←length(s);      { 计算自然数串对应的整型数组 }
```

```
for i:=1 to length(s) do a[i]←ord(s⌈length(s)−i+1⌉)−ord('0');
```

问题是，对整型数组进行数学运算要比简单的整型变量复杂。下面介绍几种与解题有关的运算。

1. 规整运算

按照定义，整型数组中 $0\leqslant a[i]\leqslant 9$ $(1\leqslant i\leqslant a[0])$。但是两个整型数组之间或整型数组与整型变量之间的算术运算都可能使 $a[i]$ 超出此范围($a[i]\geqslant 10$)。因此必须加以规整，使得 $a[i]$ 的值为 $a[i]\bmod 10$，并向 $a[i+1]$ 进位$\left(a[i+1]\leftarrow a[i+1]+\left\lfloor\frac{a[i]}{10}\right\rfloor\right)$，计算过程如下：

```
procedure check(Var a:atype);                    { 对整型数组 a 进行规整运算 }
  var p:integer;
  begin
    p←1;                                         { 从第一位开始逐位规整 }
    while(p<a[0])or(a[p]≥10)do
      begin                                      {若还有未规整的数位或者当前位超出范围,则循环}
        if p≥a⌈0⌉ then a⌈p+1⌉ ←0;                { 最高位进位 }
        a[p+1]←a[p+1]+⌊a[p]/10⌋;                 { 进位运算 }
        a[p]←a[p] mod 10;                        { 规整当前位 }
        p←p+1;                                   { 规整下一位 }
    end; {while}
    if p>a[0] then a[0]←p;                       { 设置规整后的位数 }
end; {check}
```

2. 整除运算

设整型数组 a 能整除整数变量 j。我们从 $a[a[0]]$ 开始除起，计算整商对应的整型数组。显然

$$a[i-1]\leftarrow a[i-1]+(a[i]\bmod j)*10;$$

$$a[i]\leftarrow\left\lfloor\frac{a[i]}{j}\right\rfloor(1\leqslant i\leqslant a[0]);$$

若最高位 $a[a[0]]=0$，则舍弃($a[0]\leftarrow a[0]-1$)。如此运算，直至求出整商的最低位 $a[1]$ 为止。此时再对整商数组进行规整，计算过程如下：

```
procedure div_j(var a:atype;j:integer);          { 整型数组 a 整除整数 j,返回整商 a }
  var p:integer;
  begin
    p←a⌈0⌉;                                      { 从最高位开始除起 }
    while p>0 do                                 { 若未除尽,则循环 }
      begin
        a[p−1]←a[p−1]+(a[p] mod j)*10);a[p]←a[p] div j;{ 计算 P 位和 P−1 位的值 }
        if a[a[0]]=0 then a[0]←a[0]−1;           { 舍弃首部无用的 0 }
```

```
        p←p-1;                                    { 除下一位 }
    end; {while}
    check(a);                                     { 规整整商数组 a }
end; {Div_j}
```

3. 相加运算

设整型数组

```
var a,b:atype;
```

计算整型数组 $a=a+b$。显然 $a=(a+b)$数组的长度为 $\max\{a[0],b[0]\}$。若 $a[0]<b[0]$,则被加数 a 的数组头部添足 $b\lfloor 0\rfloor-a\lfloor 0\rfloor$ 个 0,然后 a 与 b 逐位相加,最后对结果 a 进行规整:

```
procedure add(Var a:atype;b:atype);          { 计算和返回 a=a+b }
  var i:integer;
  begin
    while a[0]<b[0] do                       { 若 b 的数位多,则 a 的首部添 0 使其与 b 对齐 }
    begin a[0]←a[0]+1;a[a[0]]←0;
    End; {while}
    for i :=1 to b⌈0⌉ do a⌈i⌉←a⌈i⌉+b⌈i⌉;  { a 与 b 逐位相加 }
    check(a);                                { 对结果 a 进行规整 }
end; {add}
```

4. 计算组合 $c(a,b)$

设整数 a 和 b $(a\geqslant b)$,组合 $c(a,b)=\dfrac{a!}{(a-b)!\ b!}$,那么如何计算 $c(a,b)$对应的整型数组 c 呢?

① 计算 a 与 b 的最大公约数 $\gcd(a,b)$

按照欧几里德递归定理

$$\gcd(a,b)=\begin{cases} a & (b=0)\\ \gcd(b,a \bmod b) & b\neq 0\end{cases}$$

进行如下的计算:

```
function gcd(a,b:integer):integer;     { 计算和返回 a 和 b 的最大公约数 }
  begin
    if b=0 then gcd←a
           else gcd←gcd(b,a mod b);
  end; {gcd}
```

② 计算 $c(a,b)$

设 $u=u_1..u_b$, $d=d_1..d_b$。其中

$$u_1=a,\ u_2=a-1,..,u_b=a-b+1$$

$$d_1=1,\ d_2=2,..,d_b=b$$

显然 $\prod_{i=1}^{b} u_i = (a-b+1)\times\cdots\times(a-1)\times a = \frac{a!}{(a-b)!}$，$\prod_{i=1}^{b} d_i = 1\times 2\times\cdots\times b = b!$，$C(a,b) = \frac{a!}{(a-b)!b!} = \frac{\prod_{i=1}^{b} u_i}{\prod_{i=1}^{b} d_i}$，计算过程如下：

```
procedure combine(a,b:integer;var c:atype);        { 计算 c=c(a,b) }
  var
    i,j,x:integer;
    u,d:atype;
  begin
    u[0]←0;                                        { 设置 u={a,a+1,…,a-b+1} }
    for i:=a downto a-b+1 do begin u⌈0⌉←u⌈0⌉+1;u⌈u⌈0⌉⌉←i;End; {for}
    for i:=1 to b do d[i]←i;                       { 设置 d={1,2,..,b } }
    for i:=1 to u[0] do if u[i]<>1 then
                                   {计算 u1 * u2 * … * ub与 d1 * d2 * … * db的约分数组 u}
    for j:=1 to b do

      if d[j]<>1 then Begin x←gcd[u[i],d[j]];u[i]←⌊u[i]/x⌋;d[j]←⌊d[j]/x⌋;
      End;{for}
    c[0]←1;c[1]←1;                                 { 计算 c=u1 * u2 * … * ub }
    for i:=1 to u[0] do
    begin
      if u[i]<>1 then for j:=1 to c⌈0⌉ do c⌈j⌉←c⌈j⌉ * u⌈i⌉;
    check(c);                                      { 对组合 c 进行规整 }
    end; {for}
  end; {combine}
```

5. 十进制数——二进制数转换

我们采用除 2 取余法转换。即用 2 不断去除待转换的十进制数，直至商为 0 止。将所得的各次余数以最后余数为最高位依次排列，即得到转换后的二进制数。设

```
Var
a,b:Atype;
```

a 为十进制整型数组（$0\leqslant a⌈i⌉\leqslant 9, 1\leqslant i\leqslant a⌈0⌉$），$b$ 为对应的二进制整型数组 b（$0\leqslant b⌈j⌉\leqslant 1, 1\leqslant j\leqslant b⌈0⌉$），计算过程如下：

```
procedure dec2bin(var a:atype);    { 将十进制整型数组 a 转换为对应的二进制整型数组 a }
  var b:atype;
      p:integer;
  begin
```

```
    b[0]←0;                          { 二进制数组初始化 }
    while a⌈0⌉>0 do                  { 若未除尽,则循环 }
      begin
        b[0]←b[0]+1;                 { 二进制数增加一位 }
        if odd(a[1])                 { 根据余数的奇偶性确定当前二进制位的值 }
        then b[b[0]] ←1
        else b[b[0]] ←0;
        p←a[0];                      { 用 2 去除待转换的十进制数 }
        while p>0 do
         begin
          if odd(a[p]) and (p>1) then a[p-1]←a[p-1]+10);
          a[p]←⌊a[p]/2⌋;
          if a[a[0]]=0 then a[0]←a[0]-1;
          p←p-1;
         end; {while}
      end; {while}
    a←b;                             { 返回转换后的二进制数 }
end; {dec2bin}
```

按照题意要求,我们将自然数 m 转换成对应的二进制数 b,然后按照下列顺序累计 $1\sim m$ 中 A 类数的个数:

- 累计 1 位到 $b[0]-1$ 位二进制数中 A 类数的个数(由右向左扫描);
- 累计 $b[0]$位二进制数中小于等于 m 的 A 类数的个数(由左向右扫描)。

由于 $1\leqslant m\leqslant 10^{30}$,使得 $1\sim m$ 中 A 类数的个数上限超过任一种整型变量的范围,因此只能采用整型数组 ans 存储答案:

```
var ans:atype;
```

二、由右向左累计 A 类数的个数

设 m 对应的二进制数 a 有 $a[0]$位。我们按照位数递增的顺序累计 1 位、2 位…$a[0]-1$ 位二进制数中 A 类数的个数,如图 1.1 所示。

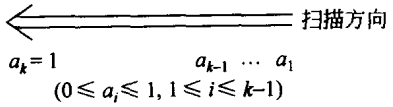

图 1.1

对于有 i $(1\leqslant i\leqslant a[0]-1)$位二进制数中的 A 类数,0 的个数多于 1 的个数。由于 i 位的值已确定,且 $a[i]=1$,因此要使得 $a\lfloor i\rfloor\sim a\lfloor 1\rfloor$ 成为 A 类数,则必须使 $a\lfloor i-1\rfloor\sim a\lfloor 1\rfloor$ 中至少含 $\left\lfloor\frac{i}{2}\right\rfloor+1$ 个 0,至多含 $i-1$ 个 0。由此得出,i 位二进制数中共有

$\sum_{j=\lfloor \frac{i}{2} \rfloor+1}^{i-1} c(i-1,j)$ 个 A 类数。

注意：位数 $i=1$、2 时，都不可能产生 A 类数(1,10,11 都不是 A 类数)。因此 $3 \leqslant i \leqslant k-1$。

计算过程如下：

```
dec2bin(a);                          { 将十进制数组转换成二进制数组 }
k←a[0];                              { 从右向左累计 A 类数的个数 }
ans⌈0⌉←1; ans⌈1⌉←0;                  { A 类数的个数初始化}
for i:=3 to k-1 do begin             {ans = Σ(i=3..k-1) Σ(j=⌊i/2⌋+1..i-1) c(i-1,j)}
  combine(i-1,i div 2+1,c);
  add(ans,c);
  for j:=i div 2+2 to i-1 do
    begin
      for k:=1 to c[0] do c[k]←c[k]*(i-j);
      div_j(c,j);add(ans,c);
    end; {for}
end; {for}
```

至此，我们得出了 1 位到 $a[0]-1$ 位二进制数中 A 类数的个数 ans，但这只是问题的一部分。问题解还应包括 $a[0]$位二进制数中小于等于 m 的 A 类数的个数。

三、自左向右累计 A 类数的个数

设 a 为某 A 类数对应的 k 位二进制数($a[0]=k$)；b 为 m 对应的二进制数。由于 a 对应的十进制数小于等于 m，因此设 a 的前 $k-i$ 位 $a[k]\sim a[i+1]$与 b 的 $b[k]\sim b[i+1]$相同，其中有 c_0个 0、c_1个 1，如图 1.2 所示。显然，初始时 $c_0=0, c_1=1$。当最终扫描至$a⌈1⌉$后，若 $c_0>c_1$，则说明 m 本身也是一个 A 类数。我们从 $a[k-1]$开始逐位向右扫描。设 A 类数的长度为 $i-1$ 的后缀中最少有 l 个 0，即 $a[i-1]\sim a[1]$中至少有 l 个 0 才能使当前 k 位二进制数 a 成为 A 类数；

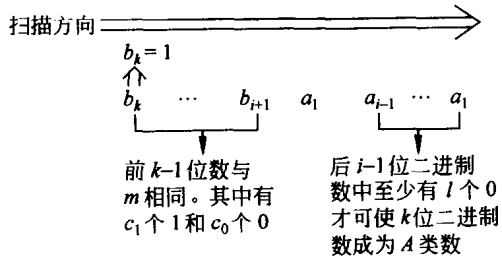

图 1.2

(1) $b[i]=0$ 时，由于 $a \leqslant b$ 且 $a[k]..a[i+1]=b[k]..b[i+1]$，因此 $a[i]$只能为 0，$c_0 \leftarrow c_0+1$，继续向右扫描；

(2) $b[i]=1$ 时，$a[i]$的值有两种可能：

① 设 $a[i]=0$。显然 A 类数中 0 的个数为 c_0+1+l'（l'为后 $i-1$ 位二进制数中 0 的个数），1 的个数为 $c_1+i-1-l'$。

当 0 的个数与 1 的个数相同时，得出 $l'=\left\lfloor\frac{c_1-c_0+i-2}{2}\right\rfloor$。显然，为了使得当前二进制数 a 成为 A 类数，必须使得 0 的个数至少为 $l'+1$，即 $l=\left\lfloor\frac{c_1-c_0+i-2}{2}\right\rfloor+1$。在这种情况下共有 $c(i-1,l)$种可能性。但这里有两个例外：

• $l<0$，则说明只有 $a[i-1]\sim a[1]$全 1 才能使当前 k 位二进制数 a 成为 A 类数。因此设 $l\leftarrow 0$；

• $l>i-1$，则说明即使 $a[i-1]\sim a[1]$全 0 也不可能使当前 k 位二进制数 a 成为 A 类数。因此 $c_1\leftarrow c_1+1$，继续向右扫描。

② 设 $a[i]=1$。在这种情况下 $a[i-1]\sim a[1]$中至少需有 $l+1=\left\lfloor\frac{c_1-c_0+i-2}{2}\right\rfloor+2$ 个 0 才可使当前 k 位二进制数 a 中 0 的个数多于 1 的个数，使之成为 A 类数。显然共有 $c(i-1,l+1)$个可能性。

在计算出前 $k-i$ 位数与 m 相同情况下的 A 类数的个数后，$c_1\leftarrow c_1+1$，继续向右扫描。由此得出算法：

```
计算 3 位..k-1 位二进制数中 A 类数的个数 ans;
c0←0; c1←1;                          { 0 的个数和 1 的个数初始化 }
for i:=a⌈0⌉-1 downto 1 do            { 由左向右累计 A 类数的个数 }
  if a[i]=1                          { 若当前位为 1 }
    then Begin
           k←⌊(c1-c0+i-2)/2⌋+1       { 保证当前二进制数成为 A 类数,至少需要 k 个 0 }
           if k<0 then k←0;          {只有 a⌈i-1⌉..a⌈1⌉全 1 才能使当前二进制数成为 A 类数 }
           if k>i-1                  {即使 a[i-1]..a[i]全 0 也不可能使当前二进制数成为 A 类数}
             then begin c1←c1+1;continue;End;{then}  { 累计 1 的个数并继续向右扫描 }
           combine(i-1,k,c);         { 计算 ans←ans+c(i-1,k)+c(i-1,k+1) }
           add(Ans,c);
           for j:=k+1 to i-1 do
             begin
               for k:=1 to c[0] do c[k]←c[k]*(i-j);
               div_j(c,j);
               add(ans,c);
             end; {for}
           c1←c1+1;
         end {then}
       else c0←c0+1;                 { 若当前位为 0,则累计 0 的个数并继续向右扫描 }
  if c0>c1 then                      { 若 m 为 A 类数,则 A 类数个数增 1, 对 A 类数个数进行规整 }
    begin ans[1]←ans[1]+1); check(ans);end; {then}
for i:=ans⌈0⌉ downto 1 do write(ans⌈i⌉);      { 输出 1~m 中 A 类数的个数 }
```

1.2 补丁 VS 错误

试题

错误就是人们所说的 Bug。用户在使用软件时总是希望其错误越少越好，最好是没有错误的。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疯狂地发放补丁(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，随后发放了一批补丁。最近 T 公司发现其发放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，往往会加入另外一些错误。

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 公司目前共发放了 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另外一些错误。此外，使用每个补丁都要耗一定的时间(即补丁程序的运行时间)。

更准确地说明如下：

设此字处理软件中可能出现的 n 个错误为集合 $B=\{b_1,b_2,\cdots,b_n\}$ 中的元素，T 公司目前共发放了 m 个补丁：$p_1,p_2,\cdots,p_m$。对于每一个补丁 p_i，都有特定的适用环境，某个补丁只有在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用，为了说明清楚，设错误集合：B_i+、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+、B_i- 交集为空。补丁 p_i 将修复某些错误而同时加入另外的某些错误，设错误集合 F_i-、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-、F_i+ 交集为空。另外，使用每个补丁都要耗一定的时间(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集合 B 中的全部 n 个错误，有没有可能通过使用这些补丁(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

输入文件第一行有两个正整数 n 和 m，n 表示错误总数，m 表示补丁总数，$1\leqslant n\leqslant 20$，$1\leqslant m\leqslant 100$。接下来 m 行给出了 m 个补丁的信息。每行包括一个正整数(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+；若为“-”，则表示 b_k 属于 B_i-；若为“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+；若为“-”，则表示 b_k 属于 F_i-；若为“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

输出一个整数，如果问题有解，输出总耗时，否则输出 0。

输入输出示例

输入文件			输出文件
3	3		8
1	000	00−	
1	00−	0−+	
2	0−−	−++	

算法分析

一、处理补丁信息

输入文件的第 $i+1$ 行($1\leqslant i\leqslant m$)给出了补丁 i 的信息，包括：

(1) 补丁 i 的运行耗时 time[i](由该行第一个正整数给出)；

(2) 补丁 i 的适用环境 B_i+ 和 B_i-。按题意，当软件包含了错误集合 B_i+ 中的所有错误而不含 B_i- 中的任何错误($B_i+\cap B_i-=\{\}$)时，补丁 i 方可使用。输入文件第 $i+1$ 行中的第 1 个字符串 $S=S_1..S_n$ 蕴含了 B_i+ 和 B_i- 信息：

若 $S_j=0$，说明第 j 个错误 $\notin(B_i+\cup B_i-)$；

若 $S_j='+'$，说明第 j 个错误 $\in B_i+$；

若 $S_j='-'$，说明第 j 个错误 $\in B_i-$；

其中，$1\leqslant j\leqslant n$。

(3) 补丁 i 衍生的错误。按题意，补丁 i 使用后修复了 F_i- 中的所有错误，并衍生出 F_i+ 中的所有错误($F_i+\cap F_i-=\{\}$)。输入文件第 $i+1$ 行中的第 2 个字符串 $S'=S'_1..S'_n$ 蕴含了 F_i+ 和 F_i- 信息：

若 $S'_j=0$，说明第 j 个错误 $\notin(F_i+\cup F_i-)$；

若 $S'_j='+'$，说明第 j 个错误 $\in F_i+$；

若 $S'_j='-'$，说明第 j 个错误 $\in F_i-$；

其中，$1\leqslant j\leqslant n$。

显然，当 $F_1+\notin\{\}$、$F_2+\notin\{\}\cdots F_n+\notin\{\}$ 时，说明所有补丁使用后软件仍含错误，应无解退出。

设数组 $b0$、$b1$、$f0$、$f1$ 存储所有补丁信息，数组元素为整数集合。

```
const
    maxn=20;                    { 错误总数的上限 }
    maxm=100;                   { 补丁总数的上限 }

type
    bugstates=set of 1..maxn;   { 错误集合类型 }
var
    b0,b1,f0,f1:array [1..maxm] of bugstates;
```

其中：

$b0[i]$——B_i＋中错误序号的集合；$b1[i]$——B_i－中错误序号的集合；

$f0[i]$——F_i＋中错误序号的集合；$f0[i]$——F_i－中错误序号的集合($1 \leqslant i \leqslant m$)。

我们可通过下述方式建立 $b0$、$b1$、$f0$、$f1$ 数组：

```
procedure initialize;                    { 输入数据 }
var
  i,j:integer;
  ch:char;                               { 输入字符 }
  ff:boolean;                            { 软件经某补丁修补后不衍生错误的标志 }
begin
  读错误总数 n 和补丁数 m;
  ff←false;                              { 假设所有补丁都衍生错误 }
  for i:=1 to m do                       { 依次输入每个补丁的信息 }
    begin
      读补丁 i 的运行耗时 time[i]和空格 ch;
      b0[i]←{};                          { 建立 Bi＋集合和 Bi－集合 }
      b1[i]←{};
      for j:=1 to n do
        begin
          读字符 ch;
          case ch of
            '+':b0[i]←b0[i]+{j};
            '-':b1[i]←b1[i]+{j};
          end; {case}
        end; {for}
      读空格 ch;
      f0[i]←{};                          { 建立 Fi＋集合和 Fi－集合 }
      f1[i]←{};
      for j:=1 to n do
        begin
          读字符 ch;
          case ch of
            '+':f0[i] ←f0[i]+{j};
            '-':f1[i] ←f1[i]+{j};
          end; {case}
          end; {for}
        if f0[i]={} then ff:=true;       { 若补丁 i 使用后不衍生错误,则标志 }
        readln;
      end; {for}
    if not ff then                       { 若所有补丁都衍生错误,则输出无解信息,并退出程序 }
      begin 输出无解信息 0;halt; end; {then}
  end; {initialize}
```