数学模型及其在信息学竞赛中的应用

上海市复旦附中高三（8）班　郭一

【关键字】

数学模型，可靠性，可解性

【引言】

数学模型是人们解决现实问题的有力武器。人们把现实问题经过科学地抽象、提炼得到数学模型，再用数学方法去解决。数学模型可分为离散和连续两种。连续数学模型需要大量的高等数学知识，中学生很少接触。在信息学竞赛经常出现的则是离散数学模型。本文主要介绍的就是离散数学模型的一般概念及建立方法。

【正文】

所谓数学模型，就是现实世界中某一类特殊的运动变化过程、关系或结构的一种模拟性的数学结构，其实也就是对现实模型进行科学抽象后得到的模型。在信息学竞赛中，试题给出的问题通常是一个现实问题，这也就需要选手在审清题意后首先把问题的关键因素总结、提炼出来，形成一个抽象的数学模型，这样有利于问题的分析与解决。

一般来说，我们在解一道有关现实问题的试题时，需要分以下几个步骤：

1. **审清题意**，了解题目的来龙去脉，弄清哪些量是已知的（输入），需要求什么（输出），数据规模如何等等。这是解决问题的前提。
2. **建立模型**，使之能够简洁高效地表达出题目给出的现实模型。
3. **解决模型**，得出算法。建模之后就是要解决模型。这步顺利与否很大部分上取决于所建模型的可解性如何。
4. **编程实现**。

其中，２、３两步是关键。模型建立与解决得好与坏对能否成功解决该题起着决定性的作用。

（1） 对于同一个现实问题，可能可以建立不同的数学模型

这种现象十分普遍，也就是平时所说的一题多解。既然如此，这里有必要讨论一下数学模型的选择问题，其实也就是评判一个模型好坏标准的问题。一个好的数学模型不仅要能够准确地反映出现实模型（可靠性），所建立的模型还必须能够被有效地解决（可解性）。这里“有效”指的是解决它的算法所需的空间与时间都在可以承受的范围之内。通常在解一些要求最优解或要求准确计数的一类具有唯一正确解的试题时，我们一般在保证可靠性的前提下，尽

量提高模型的可解性。若几个模型都具有可靠性，则当然可解性越强的模型越好。

例： 最佳旅行路线问题 [IOI'93]

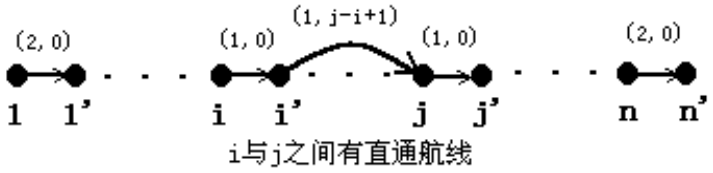
【问题描述】你在加拿大航空公司组织的一次竞赛中获奖，奖品是一张免费机票，可在加拿大旅行，从最西的一个城市出发，单方向从西向东经若干城市到达最东的一个城市（必须到达最东的城市）；然后再单方向从东向西飞回起点（可途径若干城市）。除起点城市外，任何城市只能访问 1 次，起点城市被访问 2 次：出发一次；返回一次。除指定的航线外，不允许乘其他航线也不允许使用其他交通工具。求解的问题是：给定城市表列及两城市之间的直通航线表列，请找出一条旅行路线，在满足上述限制条件下，使访问的城市数尽可能多。

这是一个现实问题，我们首先可以做如下的抽象：把每个城市抽象成一个顶点。不妨设由西到东的城市对应编号分别是 1 至 n。若两个城市之间有直通航线，则在相应的两点之间连一条边。这样，所有城市与直通航线就被抽象成了一个无向图。

【盲目搜索】这是最原始的想法。我们一开始假想有两架飞机都从最西边的城市飞向最东边的城市，并且在搜索的过程中保证两条路线中的城市除起点与终点外都不相同。记下所有找到的路线中所经城市最多的方案，把其中的一条作为向东旅行的路线，一条作为向西旅行的路线，合并起来即得最佳旅行路线。

因为搜索的时间复杂度是指数级的，所以这样做的话，时间效率不够理想。究其主要原因就是所有模型的抽象程度不够，没有把试题中的限制充分融入数学模型中，盲目性太大。

【网络流模型】为了把更多的试题中的限制融入模型中，我们在原有的模型的基础上建立了如下的最大费用最大流的模型：



图一：网络流模型

1）为了保证每个城市最多只能被访问一次，我们把每个城市 i 拆成两个顶点 i 和 i'，并在两个顶点之间连接一条 i 至 i'的有向弧，单位费用设为 0。

2）将原图中的无向边改为有向弧：若城市 i 到城市 j 有直通航线（i<j），则在顶点 i'与顶点 j 之间连接一条弧，方向由顶点 i'至顶点 j，容量为 1，这样可以保证这条航线最多只能通过一次；费用为(j-i+1)，表示这条

航线从西至东飞过的城市数（包括起点和终点）。这样使得我们能够简单地从流量费用算出旅行路线上经过的城市数。

3）顶点 1 与 1’，n 与 n’之间的弧的容量为 2。

至此，本题的数学模型已经建立，试题给出的限制条件已体现在数学模型中，因此由此模型得出的解是可行的。我们求从 1 到 n’的最大费用最大流。若得到的最大流量不是 2，则无解（即不可能从西飞到东再飞回来），否则我们设得到的最大费用为 C。因为有两条路线，所以每个未被访问的城市在费用中的贡献为 2，被访问的城市的贡献为 3。考虑到最西最东两个城市的贡献是 2 而不是 3，旅行路线中访问城市数=C+2-2*N。因为 C 最大，所以访问城市数也一定最多，即方案最佳，模型具有可靠性。因为这是一个最大费用最大流的问题，我们可以使用 Ford-Fulkerson 算法去解。

这个模型与搜索相比，可解性大大提高，时间复杂度从指数级降低到了多项式级（大约为 O（N^3））。但我们还是觉得这个模型不够简洁，抽象程度还是不够。

【动态规则模型】设 f[i,j]为从顶点 1 出发的两条不相交的路线分别到达顶点 i 与顶点 j 时，两路线的所需乘航线之和的最大值。有两种情况，若 i>j，或 i=j=n 时，我们考虑 i 的前趋顶点，不妨设为 k。此时，到达顶点 k 与 j 的两条路线的所需乘航线之和也一定最大，否则与 f[i,j]最大矛盾。若 i<j 时，结论同理可得。由此，我们有如下的动态规则方程：

$$f[1,1]=0$$

$$f[i,j]=\begin{cases}\underset{k<i\text{且}(k,i)\in E}{Min}\{f(k,j)\}+1 & (i>j\text{或}i=j=n)\\ \underset{k<j\text{且}(k,j)\in E}{Min}\{f(i,k)\}+1 & (i<j)\end{cases}$$

$f[i,i]$ 无意义 $(1<i<n)$

实际最多可能的访问城市数为 $f[n,n]$-2。时间复杂度降为 O（N^2）。

对于最佳旅行路线这一个问题，我们建立了三种不同的数学模型。这三种模型都具有可靠性（可以得出最优解），但可解性不同。一般来说，建立的模型越简洁，抽象程度越高，算法实现时不必要的操作也越少，运行效率也就越高。

值得注意的是，最近的一些竞赛中有时会出现根据解的近似程度给分的题目。对于这类题目，我们更多考虑的则是所建模型的可解性。当然，可靠性的降低也是有限度的，这个限度就是方案的可行性及误差允许范围。此外，许多数学模型有近似解法，这些都是通过适当牺牲模型自身可靠性来提高模型的可解性。

（二）一个模型可能同时适用于多个现实问题。

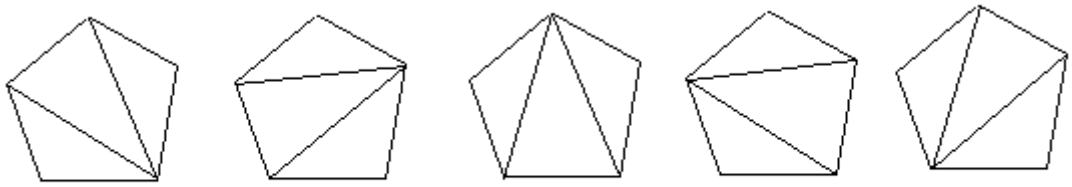
这也就是我们要研究数学模型的主要原因之一。我们解决一个数学模型就相当于解决了一类问题。比如说，最短路径问题，可谓在现实生活中无处不在，上文中提到的网络流的模型也有着很高的实用价值。这些数学模型的解决使得许多实际问题迎刃而解。然而，数学模型的解决只是解决一个现实问题的一半，另一半就是如何将现实问题转化为一个已经解决的数学模型，即如何建模。建模在现实与抽象之间起着桥梁的作用。尤其在竞赛中，后者有时显得更为重要。

（3） 如何建立数学模型？

要能够建立数学模型，首先必须*熟悉一些经典的数学模型及其算法*。这是建模的基础，没有这个基础则根本谈不上建立什么数学模型。竞赛中许多问题最终都可以转化为经典的数学模型，因此必须掌握这些常见的模型。

其次建立数学模型需要选手*有把实际问题相互联系，类比的能力*。上文已经指出许多实际模型都有着相同或相似的数学模型。既然这样，它们之间必然存在着一些相同或相似。相互联系、类比是发掘这些信息的有效手段。这里先给出一个大家都很熟悉的模型：

【凸 n 边形分割】一个凸 n 边形，可以通过不相交的(n-3)根对角线，把 n 边形拆分成(n-2)个三角形。求方案数 h_n。当 n=5 时，方案数 $h_5=5$。



图二：分割多边形 $h_5=5$

【配对括号序列】求由 n 个左括号‘(’，n 个右括号‘)’，能组成多少种不同的配对括号序列，记作 Q_n。一个序列的配对与否定义如下：

1．（）是配对的。

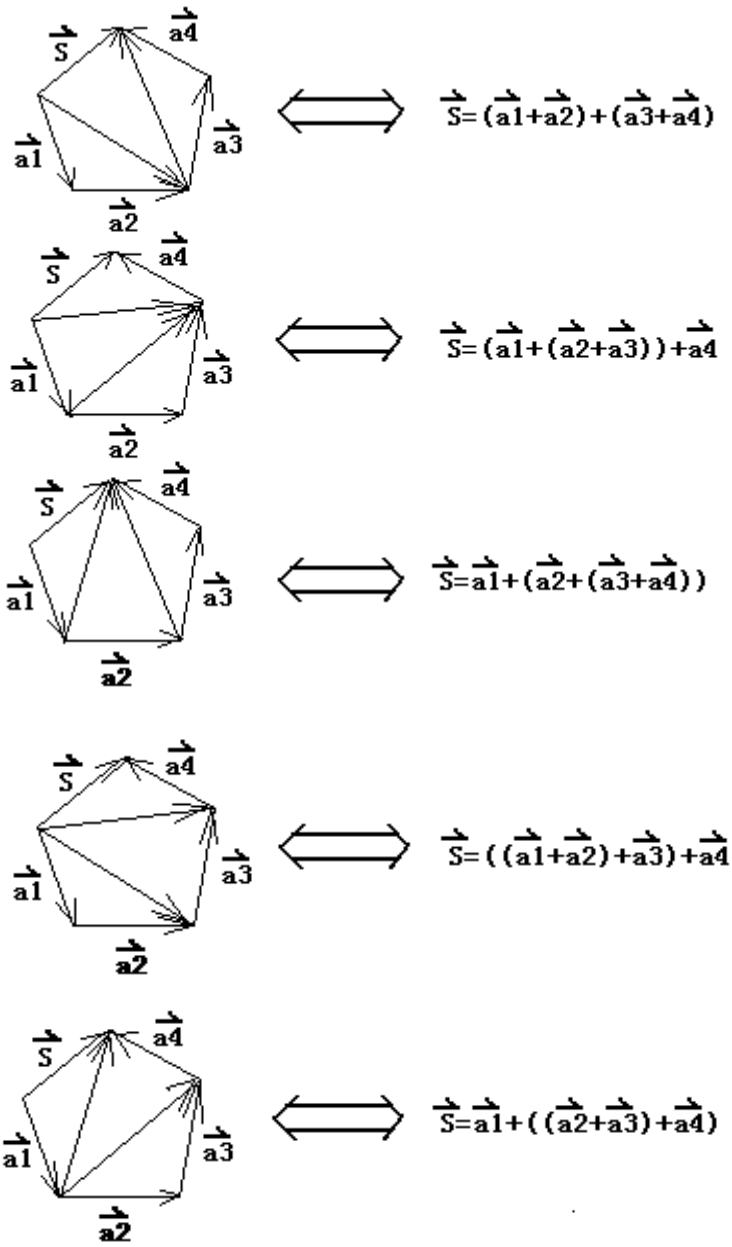
2．若 A 是配对的，则(A)也是配对的。

3．若 A、B 都是配对的，则 AB 也是配对的。

这两题的模型都是大家十分熟悉的 Catalan 数 $C_n=1/(n+1)*C(2n,n)$。通过数学计算可知，$h_n=C_{n-2}$，$Q_n=C_n$（证明略）。

【结和律】有 n 个数 $a_1,a_2,a_3,\ldots,a_n$ 依据加法结合律，不改变其顺序，只用括号表示成对的和，问有几种求和方案 P_n？

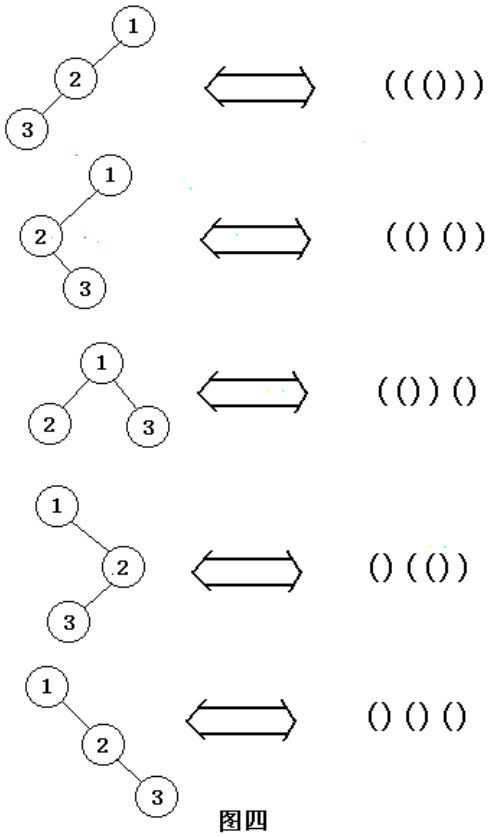
因为题目只要求求配对方案数，与 $a_1,a_2,a_3,\ldots,a_n$ 的值无关。我们不妨令 $S=a_1+a_2+a_3+\ldots+a_n$，并把 a_i（0<=i<=n）当成矢量（矢量加法也有结合律）。如下图。这样本题的方案与上题的方案就建立了一一对应的关系，本题的结果也就可以很容易地用 Catalan 数表示出来。$P_n=h_{n+1}=C_{n-1}$。



图三

【二叉树计数】问有 n 个结点的二叉树共有几种？

本题其实可以与【配对括号序列】联系。考虑从根结点开始中序遍历一个二叉树的过程。每当第一次到达一个结点时，将该结点压栈，记作‘(’，并开始递归访问其左子树；返回时访问该结点，并将该结点出栈，记作‘)’，再递归访问其右子树。这样一个二叉树就对应与一个配对的括号序列。反过来，根据任何一个配对的括号序列，我们都可以画出与其惟一对应的二叉树（证明略）。这样一来，每个结点都要出入栈一次，所以序列中一共有 n 对括号。即 n 个结点的二叉树共有 $Q_n=C_n$ 种。



图四

类似的问题还有许多，比如说【碗摞问题】，【排队找零问题】等等，这里就不再赘述。它们都是一些表面上看上去不同，本质却相同的问题。我们可以通过在它们之间相互联系、类比，得到一个经典的 Catalan 数的模型。

信息学竞赛不只是编程的竞赛，更是智力的竞赛。创造力是智力很重要的一方面。数学建模没有固定模式，能够很大程度上体现出选手的*创造力*，因此深受信息学竞赛的青睐。随着信息学竞赛的不断发展，题目中数学模型越来越隐蔽，对选手的创造力要求也越来越高。

例：01 串 [NOI99]

【题目描述】寻找一个全有 0 或 1 组成且长度为 N 字符串并使得其中

- 任意连续 L_0 个字符中的 0 的个数不小于 A_0 不大于 B_0，
- 任意连续 L_1 个字符中的 1 的个数不小于 A_1 不大于 B_1。

若不存在这样的 01 串，则输出－1，否则输出其中一个串。

数据范围：N≤1000

这道题目的数学模型非常隐蔽。许多选手当时使用了盲目搜索。由于字符串长度最大可达 1000，每一位可以是 ’ 0’ 或 ’ 1’ 两种情况，盲目搜索的最坏时间趋近于 2 ^ 1000，十分恐怖，运行时间也就可想而知了。即使有限的约束条件也改变不了搜索效率低下的本质。

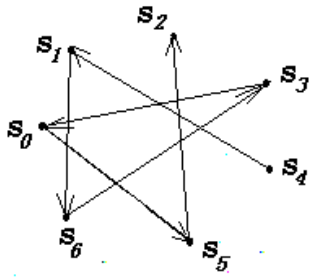
其实，本题是用图论模型来解竞赛题的成功之作。在解本题之前，我们先来看另一道题：

例：Black and White [CEOI'94]

【题目描述】寻找一个由 n 个整数组成的数列，其中任意连续 p 个整数之和为正，任意连续 q 个整数之和为负。若不存在这样的整数数列，则输出 NO，否则输出其中一个数列。

本题数学模型建立的关键在于连续数之和的表示。如果我们把紧跟在第 i 个整数之后的 k 个整数写作 $a_{i+1}+a_{i+2}+\ldots+a_{i+k}$ 的话，数学模型就很难建立了，因为这里涉及 k 个整数之和。但是，我们可以利用连续这一特点，将其表示为 $S_{i+k}-S_i$,这样的表示就十分简单，使得数学模型的建立相对容易些。我们记 S_i 为数列中前 i 个整数之和，$S_0=0$。根据题意，可以列出如下两组不等式：

$S_i > S_{i-p}$ $(p \leqslant i \leqslant n)$
$S_i < S_{i-q}$ $(q \leqslant i \leqslant n)$



图五：Black and White. n=6, p=5, q=3

我们再建一个有向图，共有 n＋1 个顶点，分别是 S_0 至 S_n，若 $S_i<S_j$，那么就从 S_i 往 S_j 引一条边。如图（n=6,p=5,q=3）。这样对于 $S_0,S_1,\ldots,S_n$ 来说，他们必须是拓扑有序的（$S_0=0$），反过来，任何一组 $S_0\ldots S_n$ 都惟一地对应一个整数数列。现在，我们已经把这道题目轮换为一个图论的拓扑排序的问题，而这个问题又是我们非常熟悉的。程序见附录。

回到 NOI'99 的那道题，将此题与 CEOI'94 的那题相互联系，我们发觉这两题都涉及到连续这个概念。我们也同样可以建一个图，顶点分别表示 $S_0\ldots S_n$，这里 S_n 表示所求串第 i 位以前（包括第 i 位）' 1'的出现次数。略有不同的是，这次不但 S_i 与 S_j 之间有大小关系，还需要表示出到底大多少。我们把这个量作为 S_i 与 S_j 之间的边上的权。具体地说，若 $S_j \geq S_i+k$，则我们就从 S_i 向 S_j 引一条权为 k 的单向边。至此，题目中的两个条件都已经在图中体现出来了。还有一点需要注意的是，本题与上题不同，它要求每个字符非 ' 0' 即 ' 1'，所以我们也需要把这点体现在图中，即再加 2n 个不等式：$S_{i+1} \geq S_i$，$S_{i+1} \leqslant S_i+1$。接下来的问题就是求其它各点到 S_0 的最长路。其实，本题与上题都可以转化为同一个模型，即图论最长路问题，因为我们可以把上题图中边的权就看作是 1。

初看上去，以上两题都似乎与图论无甚关系。题目中并没有出现图论中常见的“城市”，“终端”等顶点，也没有“铁路”，“线路”等边，还没有“长度”，“传输时间”等权，但都确确实实用图论模型漂亮地解决了，不由地让人发出感叹－－真没想到呀！其实，想到与没想到虽就一念之差，却不是偶

然的：这里面既有经验的因素，也与你所掌握数学模型的多少有关，更重要的则是你的创造力。在上两题中把S_i作为顶点，大小关系作为边，以及权的确定都不可以不说是一种创造。而正是它们在上两题的解决中起了关键性的作用。

纵观人类的进步史， 创造力有着举足轻重的作用。计算机发展至今，无论是性能的提高，软件的发展，还是网络的诞生，处处体现了人类非凡的创造力。创造力是研究信息学的基本素质之一，也是信息学竞赛考察的重点。

【参考书目】

青少年国际和全国信息学（计算机）奥林匹克竞赛指导丛书
——组合数学的算法与程序设计　吴文虎、王建德　　编著
——图论的算法与程序设计　　吴文虎、王建德　　编著
清华大学出版社

数学模型基础　　王树禾　　编著
中国科学技术大学出版社

【附程序】
Black and White，CEOI94:

```
{$A+，B-，D-，E-，F-，G+，I-，l-，N-，O-，P-，Q-，R-，S-，T-，V-，X-}
{$M 65520，0，655360}
program Black_and_White(input, output);
const
  maxn = 1000;
var
  i, n, p, q, count: integer;              {count: 拓朴排序的序号}
  s, d: array[0..maxn] of integer;         {d[i]为顶点 i 的入度}
  next: boolean;                           {拓朴排序结束标记}
begin
  write('n,p,q = ');
  readln(n, p, q);
{计算入度}
  fillchar(d, sizeof(d), 0);
  for i := 0 to n do
  begin
    if i + p <= n then inc(d[i + p]);      {S(i) < S(i+p)}
    if i - q >= 0 then inc(d[i - q]);      {S(i) < S(i-q)}
  end;
{拓朴排序}
  count := 0;
  repeat
    next := false;
    for i := 0 to n do
      if d[i] = 0 then
      begin
        s[i] := count;
        inc(count);
        if i + p <= n then dec(d[i + p]);          {相邻顶点入度减 1}
        if i - q >= 0 then dec(d[i - q]);
        d[i] := -1;
        next := true;
      end;
  until not next or (count = n + 1);   {直到所有顶点已被赋上序号或无 0 度顶点为止}
{输出}
  if count <> n + 1 then               {存在环}
    writeln('NO')
  else
    for i := 1 to n do writeln(s[i] - s[i - 1]);
end.
```

01 串，NOI99：

```
{$A+，B-，D-，E-，F-，G+，I-，l-，N-，O-，P-，Q-，R-，S-，T-，V-，X-}
```

```
{$M 65520，0，655360}
program sequence (input，output)；
const
  inputfile='input.txt'；
{ 输入文件名串 }
  outputfile='output.txt'；
{ 输出文件名串 }
var
  head:array[0..1000，1..6]of record { 有向图。head[i，k]—顶点 k 引出的第 k 条有向边，其中}
                    no，v:integer；{head[i，k].no 为该边的终点序号；head[i，k].v 为该边的权}
                  end；
  num:array[0..1000]of shortint；                    { num[i]—顶点 i 引出的边数 }
  s:array[0..1000]of integer；                       { s[i]—0 位‥i 位中 1 的个数 }
  n，a0，b0，l0，a1，b1，l1:integer；{n—串长；l0，a0，b0—连续长度为 l0 的子串中，0 的个数的下限和上限为 a0、b0；l1，a1，b1—连续长度为 l1 的子串中，1 的个数的下限和上限为 a1、b1}

procedure addedge(a，b，c:integer)；          { 从顶点 a 出发，构造一条权为 c 的有向边<a，b> }
  begin
    inc(num[a])；                                    { 累计顶点 a 引出的边数 }
    head[a，num[a]].no:=b；                          { 设置该边的终点和权 }
    head[a，num[a]].v:=c；
  end；  {addedge}

procedure init；                                     { 输入数据，构造有向图 head }
  var i:integer；
  begin
    readln(n，a0，b0，l0，a1，b1，l1)；
{ 输入数据 }
    fillchar(head，sizeof(head)，0)；
{有向图初始化 }
    for i:=1 to n do begin                  { 逐个顶点地构造有向图 }
      if i>=l0 then begin                   { 根据 a0≤l0-(si- s i→i0 )≤b0 构造有向边 }
        addedge(i，i-l0，a0-l0)；
        addedge(i-l0，i，l0-b0)；
      end；  {then}
```

```
        if i>=l1 then begin                    { 根据 a1≤si-s[i-l1]≤b1 构造有向边 }
            addedge(i-l1，i，a1)；
            addedge(i，i-l1，-b1)；
        end；  {then}
        addedge(i-1，i，0)；                                    { 根据
si-1≤si 构造有向边 }
        addedge(i，i-1，-1)；                                   { 根据
si≤si-1+1 构造有向边}
      end；  {for}
  end；  {init}

procedure main；                                                { 计算顶点 0 至其
余顶点的最长路径 }
  var i，j，k:integer；
  begin
    fillchar(s，sizeof(s)，0)；
{ s 数组清零 }
    for i:=1 to n do                                  { 逐条边地延长路径 }
      for j:=0 to n do                               { 枚举第 i 条边的所有可能情况 }
        for k:=1 to num[j] do
          if s[j]+head[j，k].v>s[head[j，k].no] { 若顶点 0 至顶点 head[j，k].no
的所有路径中，经过顶点 j 的第 k 条有向边的一条路径为目前最长，则记下 }
            then begin
              s[head[j，k].no]:=s[j]+head[j，k].v；
              if s[head[j，k].no]>head[j，k].no then begin
                                { 若 0 位‥head[j，k].no 位全填 1 也不能满足条
件，则无解退出 }
                writeln(-1)；
                exit；
              end；  {then}
            end；  {then}
        for j:=0 to n do                    { 检查有向图。若出现有向环，则无解
退出 }
          for k:=1 to num[j] do
            if s[j]+head[j，k].v>s[head[j，k].no] then begin
              writeln(-1)；
              exit；
            end；  {then}
      for i:=1 to n do                      { 根据 s 数组的值构造满足条件的 01 串
}
      if s[i]=s[i-1] then write(0)
                     else write(1)；
    writeln；
  end；  {main}
```

```
begin
   assign(input，inputfile)；                                    { 输入文件名
串与文件变量连接 }
   reset(input)；                                                {
输入文件读准备 }
   assign(output，outputfile)；                                  { 输出文件名串
与文件变量连接 }
   rewrite(output)；                                             { 输
出文件写准备 }
   init；
{ 输入数据，构造有向图 }
   main；                                                        { 计算和输出一个满
足所有条件的 01 串 }
   close(input)；                                  { 关闭输入文件和输出文件 }
   close(output)；
end.                                               { 程序结束 }
```