国际大学生程序设计竞赛试题解析

王建德 柴晓路 编著 施伯乐 主审

Keys and Notes to ACM International Collegiate Programming Contest

复旦大学出版社

国际大学生程序设计竞赛
试题解析

王建德　柴晓路　编著
施伯乐　　主审

复旦大学出版社

目　录

第一章　'98 国际大学生计算机程序设计竞赛试题解析

§1.1　计算材料的有效容量

试　题

【英文原稿】

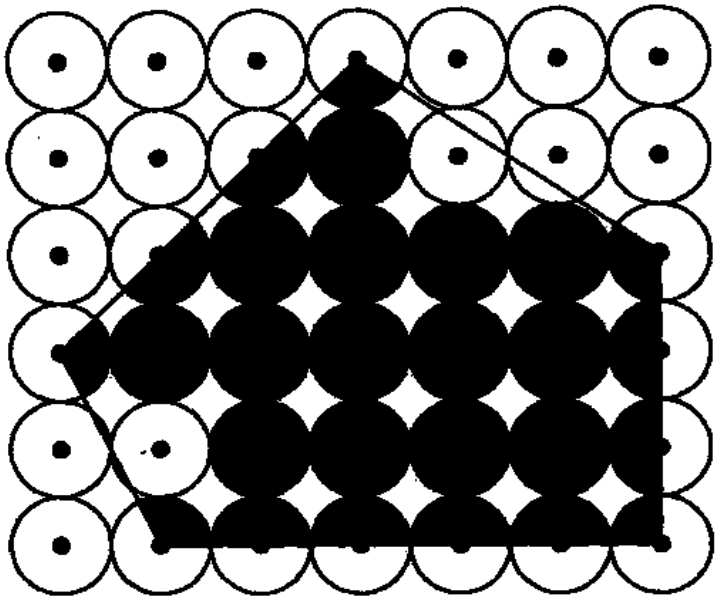
Input file： crystal.in

A new high technology company has developed a material that it hopes to market as an insulator. The material consists of crystals and the square lattice on which the crystals are grown. The points on the lattice are at 1 centimeter intervals. The crystals are formed from seeds that are planted at the lattice points. Each crystal grown into a circle of diameter 1 centimeter.

Using this material in applications will require cutting the lattice into pieces. One of the problems in cutting the lattice is that some crystals will be sliced in the process. Slicing a crystal other than through the center completely destroys that crystal's insulation property. (A cut touching a crystal tangentially does not destroy that crystal's insulation property.)

Retain insulation　　lose insulation

The insulation capacity of a piece is directly proportional to the total area of the insulating crystals (or parts of crystals) that are on the piece. The following figure shows a polygonal piece with its insulating crystals shaded.



Your job is to determine the insulating capacity of such polygonal pieces by computing the total area of the insulating crystals in it.

Input

The input consists of a sequence of polygon descriptions. Each description consists of a positive integer n ($3 \leqslant n \leqslant 25$) representing the number of vertices, followed by n pairs of integers. Each pair is the x and y coordinates of one vertex of the polygon. (The coordinate system is aligned with lattice such that integer coordinates are precisely the lattice points.)

Vertices of each polygon are given in clockwise order. No polygon will be degenerate. No coordinate will be larger than 250 in absolute value.

The input is terminated by zero for the value of n.

Output

For each polygon, first print its number ("Shape 1", "Shape 2", etc.) and then the area of the insulating crystals in cm^2, exact to three digits to the right of the decimal point.

The following sample corresponds to the previous illustration.

Input sample

```
5
0 2
3 5
6 3
6 0
1 0
0
```

Output sample

Shape 1

insulating area = 15.315 cm^2

【中文译稿】

输入文件名：crystal.in

一个高新技术公司研制了一种绝缘的新材料，这种材料由晶体和晶体赖以生长的网格矩形组成，网格上生长点的间隔距离为 1cm。晶体就是由这些生长点为萌芽向外生长，直到生长出直径为 1cm 的一个圆。

应用这种新材料需要将网格切割成块。在切割中存在一个问题，就是在切割过程中一些晶体可能会被破坏。当晶体圆片被切割，并且切割不过圆片的中心时，晶体的绝缘性能被破坏(切割线与晶体圆片相切时，仍不破坏晶体的绝缘性能)。

未被破坏的晶体　　被破坏的晶体

图 1.1-1

于是，一个材料块的有效容量就是其包含的未被破坏的晶体（或晶体的一部分）的总面积。图 1.1-2 给出了一个实例，阴影部分是未被破坏的晶体。

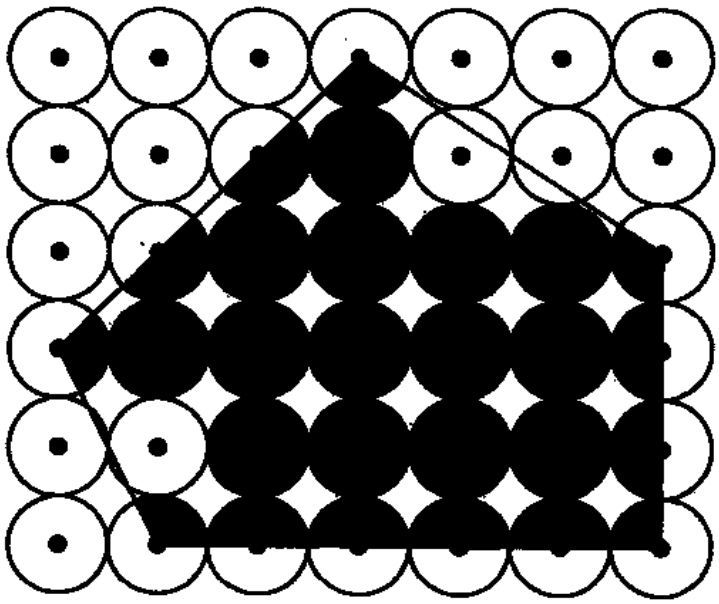


图 1.1-2

你的任务就是测量统计一个给定材料块的有效容量。

输入

输入包括一组多边形的顶点序列。每一个描述多边形的数据包括一个正整数 $n(3\leqslant n\leqslant 25)$，表示顶点总数。以下有 n 行，每行两个整数 x 和 y，表示一个顶点。所有顶点按照顺时针排列，并且坐标的绝对值不超过 250。当 $n=0$ 时表示输入结束。

输出

对于每一个有效的多边形，其输出包括一个编号(“Shape 1”，“Shape 2”，…)，之后一行输出其有效容量，单位 cm^2，精确到小数点后 3 位。

输入范例

```
5
0 2
3 5
6 3
6 0
1 0
0
```

输出范例

```
Shape 1
insulating area = 15.315 cm^2
```

算 法 分 析

一、计算包含多边形的最小网格矩形

设多边形的顶点 $P_i=(x_i, y_i)(0\leqslant i\leqslant n+1)$，其中 $P_0=P_n$，$P_{n+1}=P_n$。从 P_0 开始，$\overline{P_{i-1}P_i}$ 为多边形顺时针排列的第 i 条边。

首先求一个包含多边形的最小网格矩形，该矩形的左上角为(minx，miny)，右下角为(maxx，maxy)。

$$\text{minx}=\min\{x_i \mid 1\leqslant i\leqslant n\}\quad,\quad \text{miny}=\min\{y_i \mid 1\leqslant i\leqslant n\},$$
$$\text{maxx}=\max\{x_i \mid 1\leqslant i\leqslant n\}\quad,\quad \text{maxy}=\max\{y_i \mid 1\leqslant i\leqslant n\}$$

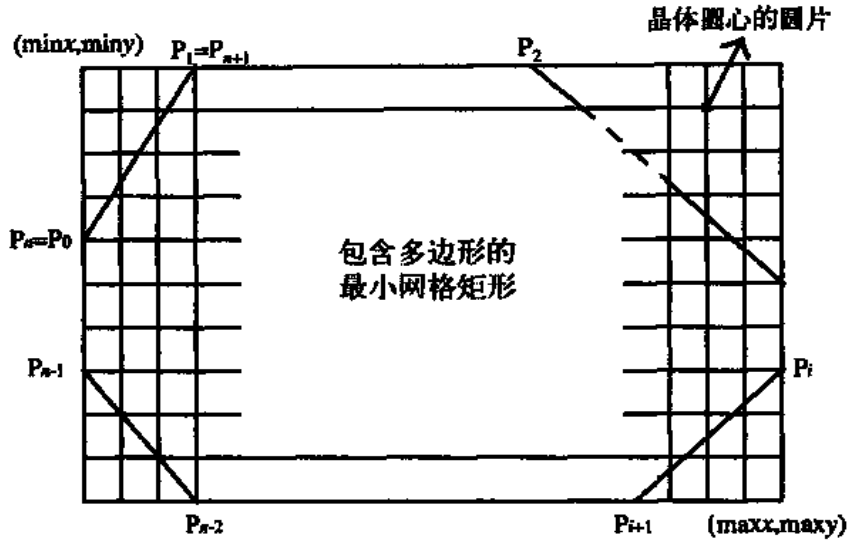


图 1.1-3

从minx行至maxx行划出(maxx−minx+1)条横线，从miny列至maxy列划出(maxy−miny+1)条竖线，形成(maxx−minx+1)×(maxy−miny+1)个网格点，每一个网格点假设为一个晶体圆片的圆心 $P=(x, y)$，它与多边形间的位置关系有五种

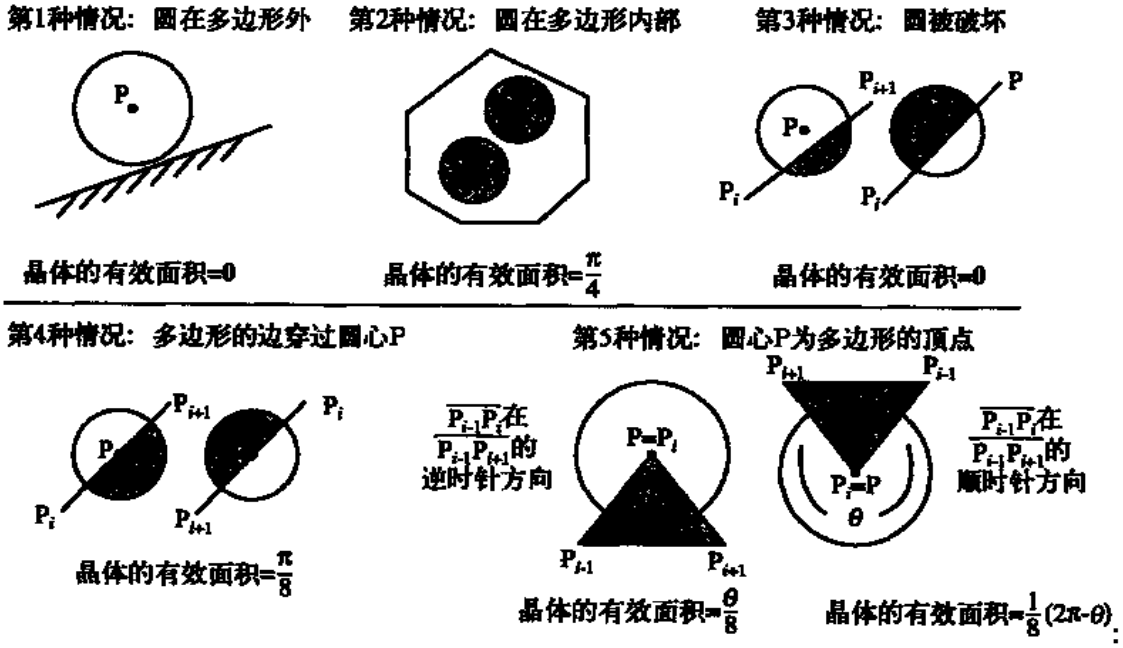


图 1.1-4

二、 几个几何问题的计算方法

1. 由一个公共点引出的两条线段间的方向问题

图 1.1-5

由点 P_1 引出两条线段 $\overline{P_1P_2}$ 和 $\overline{P_1P_3}$。通过计算这两条线段的叉积判断它们之间的方向关系。

$$\overline{P_1P_2}\times\overline{P_1P_3}=(x_2-x_1)\times(y_3-y_1)-(y_2-y_1)\times(x_3-x_1)=\begin{cases}>0 & \overline{P_1P_2}\text{ 在 }\overline{P_1P_3}\text{ 的顺时针方向}\\ =0 & \overline{P_1P_2}\text{ 与 }\overline{P_1P_3}\text{ 共线}\\ <0 & \overline{P_1P_2}\text{ 在 }\overline{P_1P_3}\text{ 的逆时针方向}\end{cases}$$

2. 两条线段的相交问题

判断线段 $\overline{P_1P_2}$ 是否与 $\overline{P_3P_4}$ 相交，必须通过跨立实验：

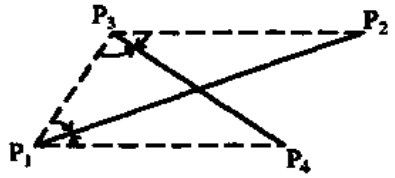


图 1.1-6

从 P_1 出发向 P_3 和 P_4 引两条辅助线 $\overline{P_1P_3}$、$\overline{P_1P_4}$，从 P_3 出发引一条辅助线 $\overline{P_3P_2}$。

若 $\overline{P_1P_2}$ 在 $\overline{P_1P_3}$ 的方向与 $\overline{P_1P_2}$ 在 $\overline{P_1P_4}$ 的方向相反，

$$((\overline{P_1P_2}\times\overline{P_1P_3})\times(\overline{P_1P_2}\times\overline{P_1P_4})<0),$$

则说明 P_3 和 P_4 分别位于 $\overline{P_1P_2}$ 的两旁；若 $\overline{P_3P_4}$ 在 $\overline{P_3P_1}$ 的方向与 $\overline{P_3P_4}$ 在 $\overline{P_3P_2}$ 的方向相反，

$$((\overline{P_3P_4}\times\overline{P_3P_1})\times(\overline{P_3P_4}\times\overline{P_3P_2})<0),$$

则说明 P_1 和 P_2 分别位于 $\overline{P_3P_4}$ 的两旁；

如果上述两个条件同时成立，则判定 $\overline{P_1P_2}$ 与 $\overline{P_3P_4}$ 相交。

3. 点与直线的距离

设多边形边 $\overline{P_iP_{i+1}}$ 所在的直线方程为

$$ax+by+c=0$$

晶体圆片的圆心 $P=(x',y')$ 为 $\overline{P_iP_{i+1}}$ 外的一点，P 点至 $\overline{P_iP_{i+1}}$ 的垂足为 $P'=(x'',y'')$。求 P 至 $\overline{P_iP_{i+1}}$ 的距离 $d=|PP'|$。

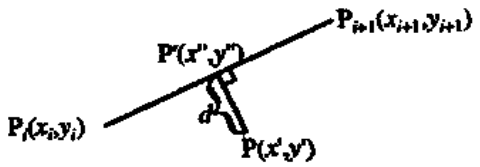


图 1.1-7

由于$(x_i,\ y_i)$和$(x_{i+1},\ y_{i+1})$在直线上，因此

$$ax_i+by_i+c=0 \quad , \quad ax_{i+1}+by_{i+1}+c=0$$

由此得出

$$a=y_i-y_{i+1} \quad , \quad b=x_{i+1}-x_i \quad , \quad c=x_i\times y_{i+1}-y_i\times x_{i+1}$$

即$\overline{P_iP_{i+1}}$所在的直线方程为

$$(y_i-y_{i+1})x+(x_{i+1}-x_i)y+x_i\times y_{i+1}-y_i\times x_{i+1}=0$$

由于$\overline{PP'}\perp\overline{P_iP_{i+1}}$所以它们的斜率互成负倒数，即

$$\frac{x''-x'}{y''-y'}=\frac{a}{b} \quad \text{或者} \quad \frac{x''-x'}{a}=\frac{y''-y'}{b}=k$$

$$x''=x'+a\times k \qquad y''=y'+b\times k$$

由于 P'在$\overline{P_iP_{i+1}}$上，所以$ax''+by''+c=0$，即$a(x'+a\times k)+b(y'+b\times k)+c=0$，从而得出

$$k=-\frac{ax'+by'+c}{a^2+b^2}$$

即$x''=x'-a\dfrac{ax'+by'+c}{a^2+b^2}$，$\qquad y''=y'-b\dfrac{ax'+by'+c}{a^2+b^2}$.

将上式代入$d=\sqrt{(x''-x')^2+(y''-y')^2}$，得出

$$d=\frac{|ax'+by'+c|}{\sqrt{a^2+b^2}}$$

4.计算两条线段的夹角弧度

设多边形边为$\overline{P_iP_{i+1}}$，晶体圆片的圆心为 P，计算$\overline{P_iP_{i+1}}$和$\overline{P_iP}$的夹角弧度θ

$$a=|\ \overline{P_{i+1}P}\ |=\sqrt{(x_{i+1}-x)^2+(y_{i+1}-y)^2}$$

$$b=|\ \overline{P_iP}\ |=\sqrt{(x_i-x)^2+(y_i-y)^2}$$

$$c=|\ \overline{P_iP_{i+1}}\ |=\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}$$

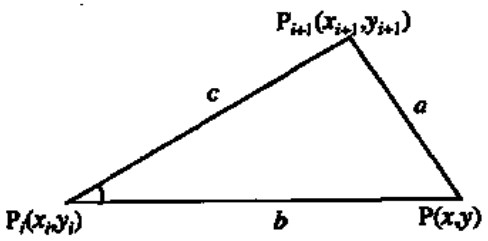


图 1.1-8

(1)若 b=0 或者 c=0，则$\theta=\frac{\pi}{2}$。

(2)根据余弦法定理计算$\cos\theta=\frac{b^2+c^2-a^2}{2bc}$

若$\cos\theta=-1$ 并且 $a=b+c$，　则$\theta=\pi$；

若$\cos\theta=0$　则$\theta=\frac{\pi}{2}$。

(3)计算

$$\theta=\begin{cases}\text{Arctan}(\sqrt{1-\cos^2\theta}/\cos\theta)+\pi & \arctan(\sqrt{1-\cos^2\theta}/\cos\theta)<0\\ \text{Arctan}(\sqrt{1-\cos^2\theta}/\cos\theta) & \arctan(\sqrt{1-\cos^2\theta}/\cos\theta)\geqslant 0\end{cases}$$

5. 几点重要的结论

设 P 为一个晶体圆片的圆心，$\overline{P_iP_{i+1}}$ 为多边形的一条边。如果 P 与 $\overline{P_iP_{i+1}}$ 构成两个锐角（$\overline{P_iP_{i+1}}$ 与 $\overline{P_iP}$ 的夹角$\leqslant\frac{\pi}{2}$，并且 $\overline{P_{i+1}P}$ 与 $\overline{P_{i+1}P_i}$ 的夹角$\leqslant\frac{\pi}{2}$），则可通过分析 P 点至$\overline{P_iP_{i+1}}$ 的距离 d 得出许多有趣的结论:

(1)若 d=0，则说明 P 在多边形边 $\overline{P_iP_{i+1}}$ 上，该晶体圆片的一半($\frac{\pi}{8}$)为有效面积;

(2)若 d<0.5，则说明该晶体圆片被 $\overline{P_iP_{i+1}}$ 破坏;

(3)若 $d\geqslant$0.5，则说明该晶体圆片或在多边形内部或在多边形外部，其中 d=0.5 表明圆与 $\overline{P_iP_{i+1}}$ 相切。在这种情况下，由 P 点出发引一条辅助线（即 P 点的 (x,y) 至 $(-1000,y+1)$ 连一条线段，该线段与多边形顶点不可能重合）。若该辅助线与多边形边相交的次数为奇数，则说明晶体圆片在多边形内部；若相交偶数次，则说明圆在多边形外。

另外还有一种特例：

(4) 若 P 与多边形的顶点 P_i 重合($P=P_i$)，则必须计算出 $\overline{P\,P_{i-1}}$ 和 $\overline{P\,P_{i+1}}$ 间位于多边形内 P 的夹角θ，该晶体圆片的有效面积为$\frac{\theta}{8}\left(\frac{\pi}{4}\times\frac{\theta}{2\pi}\right)$。

问题是 P_{i-1}，P_i 和 P_{i+1} 是顺时针排列的。若$\overline{P_{i-1}P_i}$ 在 $\overline{P_{i-1}P_{i+1}}$ 的顺时针方向，按上述方

法计算出的θ是$\overline{P_{i-1}P_i}$和$\overline{P_{i+1}P_i}$间位于多边形外端的夹角弧度，因此$2\pi-\theta$才真正是这两条边在多边形内部的有效夹角弧度。在这种情况下，晶体的有效面积应为$\frac{2\pi-\theta}{8}(\frac{\pi}{4}\times\frac{2\pi-\theta}{2\pi})$。

有了上述结论，便不难得出计算材料块有效容积的方法如下。

三、计算材料块有效容积的方法

将(maxx-minx+1)×(maxy-miny+1)个晶体中未被破坏的晶体(或者晶体的一部份)面积累加起来，便可构成材料块的有效容积s。计算方法如下:

```
s←0;
for x:=minx to maxx do
  for y:=miny to maxy do
    begin
     for i:=1 to n do
      if (x，y)至P_iP_{i+1}的距离<0.5) and (x，y）与P_iP_{i+1}构成两个锐角)
       then begin 设置(x，y)为圆心的晶体被破坏标志; break; end;{then}
     if (x，y)为圆心的晶体未被破坏 then
   begin
    for i:=1 to n do
          if (x，y)重合 P_i then
           begin
            θ ←P_iP_{i-1}与P_iP_{i+1}间的夹角弧度;
           设置(x，y)是多边形顶点的标志;
           if P_{i-1}P_i在P_{i-1}P_{i+1}的顺时针方向
             then s←s+(2π－θ）/8
             else  s←s+θ/8;
            break;
           end; {then}
        if  (x，y)非多边形顶点 then
         for i:=1 to n do
     if ((x，y)与P_iP_{i+1}的距离=0) and (（x，y）与P_iP_{i+1}构成两个锐角)
          then begin
           s←s+π/8; 设置多边形边通过(x，y)的标志 break;
               end {then}
       if ((x，y)非多边形顶点) and (未有多边形边通过(x，y))
          then begin;
           从(x，y)出发向（-1000,y+1）引一条辅助线L;
```

```
                    t←0;
                    for i:=1 to n do if P_iP_{i+1} 与 L 相交 then t←t+1;
                    if t 为奇数 then s←s+π/4;
                        end; {then}
                end; {then}
            end;  {for}
```

程 序 题 解

```
{$N+}
program crystal;
const
    oarea          = 0.7853981634;              { π/4}
type
    ptype          = array[1..2] of integer;{ 坐标类型 }
var
    f1，f2         : text;                      { 输入文件变量和输出文件变量}
    n，i，j，k，no    : integer;
    s              : array[0..100] of ptype;{ 多边形的顶点序列 }
    m1，m2         : ptype;                 { 包含多边形的最小矩形的左上角和右下角 }
    p，p2          : ptype;                 {P为圆心，由圆心出发向左延伸的辅助线为 PP_2 }
    flag       : byte;                  { 晶体圆片的类型 }
    t，  area      : real;                  { 圆心至多边形边的距离，有效容量 }
function distance(p1，p2:ptype):real; { 计算和返回 P_1P_2 的长度 }
begin
    distance := sqrt(sqr(p1[1]-p2[1])+sqr(p1[2]-p2[2]));
end;  {distance}

function getl(p，p1，p2:ptype):real;  { 计算和返回圆心P至多边形边 P_1P_2 的距离 }
var
    a，b，c        : real;
begin
    a := p1[2]-p2[2];
    b := p2[1]-p1[1];
    c := p1[1]*p2[2]-p1[2]*p2[1];
    getl := abs(a*p[1]+b*p[2]+c)/sqrt(a*a+b*b);
end;  {getl}
```

```
function angle(p1，p2，p3:ptype):real;  { 计算和返回P1P2和P1P3夹角的弧度 }
var
    a，b，c，d，e     : real;
begin
    a := distance(p2，p3);
    b := distance(p1，p3);
    c := distance(p1，p2);
    if (b<1e-5) or (c<1e-5) then d := 0
    { 若|P1P3|=0或者|P1P2|=0，则夹角弧度为 π/2 }
    else d := (b*b+c*c-a*a)/2/b/c;      { 否则计算夹角弧度的余弦值 }
    if (abs(d+1)<1e-5) and (abs(a-b-c)<1e-5) then
    { 若余弦值=-1且|P2P3|=|P1P3|+|P1P2|则夹角弧度为 π }
    begin
        angle := pi;
        exit;
    end;   {then}
        if abs(d) < 1e-5 then e := pi/2{ 若余弦值=0，则夹角弧度为 π /2 }

    else e := arctan(sqrt(1-d*d)/d);   { 否则计算和返回夹角弧度 }
    if e < 0 then e := pi+e;
    angle := e;
end;   {angle}

function gets(p1，p2，p3:ptype):real;  { 计算和返回P1P2和P1P3的叉积 }
begin
    gets := (p2[1]-p1[1])*(p3[2]-p1[2])-(p2[2]-p1[2])*(p3[1]-p1[1]);
end;   {gets}

function intersect(p1，p2，p3，p4:ptype):boolean;
{ 若P1P2与P3P4相交，则返回true;否则返回false }
begin
intersect:=(gets(p1，p2，p3)*gets(p1，p2，p4)<0)and(gets(p3，p4，p1)*gets(p3，p4，p2)<0)end;
{intersect}

begin
    assign(f1，  'crystal.in');  { 输入文件名串与文件变量连接，输入文件读准备 }
    reset(f1);
```

```
assign(f2,  'crystal.out'); {输出文件名串与文件变量连接，输出文件写准备 }
rewrite(f2);
no := 0;                          { 编号初始化 }
readln(f1,  n);                   { 读多边形的顶点数 }
while n <> 0 do                   { 若文件未读完，则累计编号 }
begin
    inc(no);
    writeln(f2,  'Shape ',  no);
    { 读入多边形的顶点序列，并计算包含多边形的最小矩形的左上角和右下角 }
    m1[1] := maxint; m1[2] := maxint;
    m2[1] := -maxint; m2[2] := -maxint;
    for i := 1 to n do
    begin
    readln(f1,  s[i,  1],  s[i,  2]);
        if s[i,  1] < m1[1] then m1[1] := s[i,  1];
        if s[i,  1] > m2[1] then m2[1] := s[i,  1];
        if s[i,  2] < m1[2] then m1[2] := s[i,  2];
        if s[i,  2] > m2[2] then m2[2] := s[i,  2];
    end; {for}
    area := 0;                    { 有效容量初始化 }
    s[n+1] := s[1];               { 形成封闭多边形 }
    s[0] := s[n];
    for p[1] := m1[1] to m2[1] do   { 枚举每一个可能的圆心P }
    for p[2] := m1[2] to m2[2] do
    begin
    flag := 0;                { 圆心类型初始化 }
    for i := 1 to n do{ 搜索多边形的每一条边。若圆心p与某条边的距离小于0.5
        且p与该边构成两个锐角，则设置以P为圆心的晶体圆片被破坏标志 }
        begin
        t := getl(p,  s[i],  s[i+1]);
        if(t>1e-5)and(t<0.5)and(angle(s[i],  p,  s[i+1])<=pi/2)
            and(angle(s[i+1],  p,  s[i])<=pi/2)
          then begin
            flag := 2;
            break;
          end   {then}
        end;   {for}
      if flag = 0 then{若以P为圆心的晶体圆片未被破坏，则搜索多边形每一个顶点}
        for i := 1 to n do
```

```
        if (p[1]=s[i，1]) and (p[2]=s[i，2]) then
        { 若p重合于多边形的某顶点，则设置p为多边形顶点标志 }
        begin
          flag := 1;
          break;
        end;  {then}
      if flag=0 then{若p非多边形顶点且所在圆未被破坏，则搜索多边形的每一条边}
      for i := 1 to n do
        begin
{若P与多边形某条边的距离=0且P与该边构成两个锐角，则设置多边形边穿过P的标志}
        t := getl(p，  s[i]，  s[i+1]);
        if(abs(t)<1e-5)and(angle(s[i]，p，s[i+1])<=pi/2)and
          (angle(s[i+1]，p，s[i])<=pi/2)
          then begin
                 flag := 1;
                 break;
               end  {then}
        end;  {for}
      if flag = 1 then              { 若P为多边形的顶点或者多边形边穿过P }
        begin
          for i := 1 to n do  { 判断P是否为多边形顶点 }
          if (p[1] = s[i，1]) and (p[2] = s[i，2]) then
          begin
          flag := 0;
               break;
          end;  {then}
          if flag = 0 then  { 若P为多边形的第i个顶点，则根据P与多边形第i-1
          个顶点和第i+1个顶点的夹角方向计算有效容量 }
          begin
          if gets(s[i-1]，s[i]，s[i+1]) > 0
           then area:=area+oarea*abs(pi*2-angle(s[i]，s[i-1]，s[i+1]))/pi/2
           else area := area+oarea*abs(angle(s[i]，s[i-1]，s[i+1]))/pi/2;
          end  {then}
          else area := area+oarea/2;
                                        {否则多边形边穿过P，π/8计入总有效容量 }
      end{then}
        else  {若P即非多边形顶点、所在的圆也未被多边形的边穿过 }
        if flag = 0 then
        begin
```

```
            p2[1] := -1000;  { 则从P出发，向左延伸一条水平线 P P2 }
                p2[2] := p[2]+1;
                j := 0;              { 计算 P P2 与多边形边相交的次数 }
                for i := 1 to n do
                if intersect(p, p2, s[i], s[i+1]) then inc(j);
                if odd(j) then area := area+oarea;
            {若相交奇数次，则以P为圆心的晶体圆片在多边形内， π/4 计入总有效容量}
             end;  {then}
        end;  { then}
        writeln(area/oarea);  { 输入总有效容量 }
        writeln(f2,  'insulating area = ',  area:0:3,  ' cm^2');
        writeln(f2);
        readln(f1,  n);          { 读下一个多边形的顶点数 }
    end;  {while}
    close(f1);                   { 关闭输入文件和输出文件 }
    close(f2);
end.  {main}
```

§1.2 飞 行 计 划

试　题

【英文原稿】

Input file：flight.in

Your job is to write a program that plans airplane flights. Each flight consists of a series of one or more legs. Your program must choose the best altitude for each leg to minimize the amount of fuel consumption during the trip.

The airplane has a fixed airspeed, given by the constant VCRUISE, and a most－efficient cruising altitude, AOPT. When flying at altitude AOPT, fuel consumption in gallons per hour is given by GPHOPT. When flying at an altitude that is different from AOPT, fuel consumption increases by GPHEXTRA for each 1000 feet above or below AOPT. The flight starts and finishes at an altitude of 0. Each 1000-feet-climb burns an extra amount of fuel given by CLIMBCOST (there is no reduction in fuel consumption when you descend). Make the approximation that all climbing and descending is done in zero time at the beginning of each flight leg. Thus each leg is flown at a constant airspeed and altitude. For this problem, the airplane characteristics are given

by the following constant:

VCRUISE	400 knots (a knot is one nautical mile per hour)
AOPT	30000 feet
GPHOPT	2000 gallons per hour
GPHEXTRA	10 gallons per hour for each 1000 feet
CIIMBCOST	50 gallons per 1000 feet of climb

Before each flight, you are given the length of each leg and the tailwind expected for each leg. A positive tailwind increases the airplane's speed over the ground, and a negative tailwind decreases its speed over the ground. For example, if airspeed is 400 knots and the tailwind is −50 knots, the speed over the ground is 350 knots.

By policy, altitude for each leg must be some integer multiple of 1000 feet, between 20000 and 40000 feet, inclusive. Your program must compute the best altitude for each leg to minimize overall fuel consumption for the trip, and must compute the fuel required for the trip.

Input

The first line contains an integer N, representing the number of flights you are required to plan. Each flight consists of the following input lines:

The first input line in a flight contains an integer K ($0<K<10$), representing the number of legs in the flight.

The next K input lines each contain the following three integer:

(1) The length of the leg, in nautical miles

(2) The expected tailwind at 20000 feet, in knots

(3) The expected tailwind at 40000 feet, in knots

The expected tailwind at altitudes between 20000 and 40000 feet is computed by linear interpolation. For example, the expected tailwind at 30000 feet is halfway between the expected tailwind at 20000 feet and the expected tailwind at 40000 feet.

Output

Your program must produce one output line for each flight. The output line must contain the flight number (Counting from the beginning of the problem), the chosen altitude for each leg (in thousand of feet), and the fuel required for the trip (in gallons, round to the nearest gallon).

Input sample

```
2
2
1500 -50 50
1000 0 0
3
```

1000 50 0
2000 0 20
1800 -50 100

Output sample

Flight 1: 35 30 13985
Flight 2: 20 30 40 23983

【中文译稿】

输入文件名：flight.in

你的任务是设计一个程序来规划航线。每一条航线包括一系列的地域。你的程序必须对每一个地域选择一个最好的海拔高度飞行，以使总的耗油量最少。

一架飞机具有一个正常飞行速度，用一个常数 VCRUISE 表示；对于这个正常的飞行速度飞机有一个固定的飞行海拔高度，用一个常数 AOPT 表示。当飞机在这个海拔高度 AOPT 飞行时，其每小时耗油 GPHOPT(gallon)，当飞机在不同于 AOPT 高度飞行时，每小时耗油量将随较 AOPT 的垂直距离每高或低 1000(feet)而增加 GPHEXTRA(gallon)。飞机的起点和终点的海拔高度都为 0。另外飞机每上升 1000(feet)需要增加 CllMBCOST(gallon)的耗油量，规定飞机升降的时间不计。这样，飞机在每一个地域都应有一个飞行高度和飞行速度，那么，你的任务就是确定这个飞行计划。下面我们给出前述定义的一些常数：

VCRUISE ＝ 400 节（1 节指一海里每小时)　表示飞机的速度；

AOPT ＝ 30000 (feet)　　表示飞机正常飞行的高度；

GPHOPT ＝ 2000 (gal/h)　　表示飞机正常飞行时每小时耗油量；

GPHEXTRA＝10(gal/h • 1000 ft) 表示飞机高度离开 AOPT 每 1000 feet 需增加每小时的耗油量；

CLIMBCOST＝50(gal/1000 ft) 表示飞机每上升 1000 feet 需要的耗油量；

(注：gallon，加仑，1 gal(gallon)=3.785412 L　feet, 英尺，1 ft=0.3048 m)

对于每个地域，我们已知它的长度，以及当地在 20000 feet 和 40000 feet 处的风速，并且风速是随高度线性变化的。于是，飞机飞行的实际速度为 VCRUISE 与当地等高的风速的矢量和（注意，风速与 VCRUISE 方向均水平）。

飞行高度在[20000feet，40000feet]范围内，并且是 1000 feet 的整数倍。你的程序需要确定飞机在各个地域的实际飞行高度，使总耗油量最小，并算出每个旅程的耗油量。

输入

N（待规划的航线数）。

(对于每个航线有：

K（地域数）(0<K<10)。

(接下去的 K 行描述这 K 个地域，每行包括三个整数：

(1)地域长度（单位为海里）；

(2)低空风速（20000 feet 上。单位为节)；

(3)高空风速（40000 feet 上。单位为节)。

注意，当地风速在垂直方向上呈反向线性变化，例如在 30000 feet 处的风速为在 20000 feet 处的风速与在 40000 feet 处的风速的平均数。

输出

Flight 编号 ：各个地域的飞行高度(1000 feet 为单位)，最小总耗油量（单位为 gallon）

输入范例

```
2
2
1500 -50 50
1000 0 0
3
1000 50 0
2000 0 20
1800 -50 100
```

输出范例

```
Flight 1 : 35 30 13985
Flight 2 : 20 30 40 23983
```

算 法 分 析

一、计算飞机在地域 *i* 的耗油量

假设飞机在地域 *i*-1（2≤*i*≤*k*+1）的海拔高度为 *l*，飞至地域 *i* 后的海拔高度变为 *m*，计算飞机在地域 *i* 的耗油量 g[*i*，*l*，*m*]必须考虑如下几个因素：

注: 为了计算方便，飞行高度和风速以 1000 feet 为一个单位。

1. 上升过程增加的耗油量 *a*

由于飞机每上升一个单位，需要增加 *climbcost* 的耗油量，因此

$$a=\begin{cases}0 & m\le l\\ climbcost\ (m-l) & m>l\end{cases}$$

2. 每小时的实际耗油量 *b*

飞机在海拔高度 *aopt* 飞行时每小时耗油 *gphopt*。当飞机在 *m* 高度飞行时，与 *aopt* 的垂直距离为 | *m*-*aopt* | 个单位。每垂直偏离 *aopt* 高度一个单位，需要增加耗油量 *gphextra*。因此

$$b=gphopt+|m-aopt|\times gphextra$$

3. 在地域 i 的实际飞行时间 c

c= 地域 i 的长度/地域 i 的实际飞行速度

由于地域 i 的风速垂直反向线性变化，m 单位处的风速为在 20 单位处的风速与 40 单位处的风速的平均数，因此地域 i 的等高风速为

$$\frac{(m-20)\times\text{地域}i\text{的高空风速}+(40-m)\times\text{地域}i\text{的低空风速}}{40-20}$$

而飞机的实际速度为 VCRUISE 与地域 i 等高风速的矢量和，因此

c=地域 i 的长度/（地域 i 的等高风速+VCRUISE）

显而易见，g[i，l，m]=a+b×c。

二、规划飞行方案

设 data[i，j]为在确定地域 i 的飞行高度为 j 的情况下，飞机由地域 1 飞至地域 i 所耗费的最小总耗油量；data2[i，j]为记忆表。飞机由地域 i-1 的 data2[i，j]高度到达地域 i 的 j 高度，可使耗油总量最小。即 data[i，j]=data[i-1，data2[i，j]]+g[i，data2[i，j]，j]。

问题是怎样才能计算 data[i，j]呢？首先我们必须明确，一旦按最优性要求确定了飞机在地域 i-1 的飞行高度为 data2[i，j]，由于 g[i，data2[i，j]，j]为一个定值，因此 data[i-1，data2[i，j]]的值必须最小。不然的话，将它替换到 data[i，j]表达式中，则可使表达式值变得更小，出现矛盾。这就说明问题的最优解包含了子问题的最优解，即该问题具备了最优子结构。

下面我们来分析一下最优表达式的构造：

飞机由地域 1 起飞，因此 data[1，j]=g[1，0，j] (20≤j≤40)。然后顺序计算：data[2，20]‥data[2，40]，data[3，20]‥data[3，40]，……，data[k，20]‥data[k，40]。

在计算 data[i，j]时，我们无法预知飞机在地域 i-1 时应以怎样的高度飞行方可使表达式值最小，因此只能一一假设飞机在地域 i-1 的高度为 20，21，…，40，即分别求出

data[i-1，l]+g[i，l，j] (20≤l≤40)

从上述 21 个表达式中选出值最小的一个。即

data[i，j]=min{data[i-1，l]+g[i，l，j] | 20≤l≤40}

将满足上式的飞行高度 l 存入记忆表 data2[i，j]。

由此可见，在求 data[i，j]的过程中不断查阅子问题的解 data[i-1，20]‥data[i-1，40]。显然这个最优化问题包含了重叠子问题，采用动态程序设计方法是最合适不过的了。按照上述分析，我们可以直接写出动态规划方程：

$$\text{data}[i,j]=\begin{cases}\text{g}[1,0,j] & i=1\\ \min\{\text{data}[i\text{-}1,l]+\text{g}[i,l,j]\} & i>1\end{cases}$$

$1\leqslant i\leqslant k$　　$20\leqslant j\leqslant 40$　　$20\leqslant l\leqslant 40$

求解 data 表和 data2 表的算法十分简单：

```
for j:=20 to 40 do data[1，j]←g[1，0，j];
for i:=2 to k do
```

```
for j:=20 to 40 do
  for l:=20 to 40 do
    begin
     计算 data[i，j]=min{data[i-1，l]+g[i，l，j]};
     将满足上式的 l 存入记忆表 data2[i，j];
    end; {for}
```

三、输出飞行计划

有了记忆表 data2 我们便可以从地域 k 出发倒推飞机在各地域的飞行高度 data3：

```
for j:=20 to 40 do data3[k]←计算满足 min{data[k，j]}的 j;
for i:=k downto 2 do data3[i-1]←data2[i，data3[i]];
```

最后输出飞行计划：

```
for i:=1 to k do 输出飞机在地域 i 的飞行高度 data3[i];
输出最小总耗油量 data[k，data3[k]];
```

每输入一条航线信息，我们便使用上述方法规划该航线的飞行方案，直至 n 条航线规划完毕。

程 序 题 解

```
program flight;
const
    VCRUiSE        = 400;      { 正常飞行速度 }
    AOPT           = 30;    { 正常飞行的高度 }
    GPHOPT         = 2000;     { 正常飞行时每小时耗油量 }
    GPHEXTRA       = 10;  { 离开aopt 每1000feet增加的每小时的耗油量 }
    CliMBCOST      = 50;       { 上升1000feet需要的耗油量 }
var
    f1，f2         : text;     { 输入文件和输出文件 }
    n，k，i，j，l，n2: integer; { n为待规划的航线数；n2为已规划的航线数 }
    t              : real;     { 辅助变量 }
    data       : array[1..10，  20..40] of real;{data[i，j]——在确定地域i的飞行高
                  度为j的情况下，飞机由地域i-1飞至地域i所耗费的最小耗油量 }
    data2          : array[1..10，  20..40] of integer;{记忆表。Data2[i，j]为耗油量达
                  到data[i，j]时，飞机在地域i-1的飞行高度}
    data3          : array[1..10] of integer;
                  {data3[i]为最佳方案中飞机在地域i的飞行高度}
function gets(r:real):real;  { 若r为负数，则返回0，否则返回r }
begin
    if r < 0 then gets := 0
```

```
    else gets := r;
end;  {gets}
function getgallon(n, m, l:integer):real;
begin      {飞机在地域n-1的高度为l，在地域n达到m高度。计算飞机在地域n的耗油量 }
    getgallon := CliMBCOST*gets(m-l)+(abs(m-AOPT)*GPHEXTRA+GPHOPT)*
        (idata[n, 1]/( ((m-20)*idata[n, 3]+(40-m)*idata[n, 2])/20+VCRUISE));
end;  { getgallon}
begin
    assign(f1,  'flight.in');  { 输入文件名串与文件变量连接，输入文件读准备 }
    reset(f1);
    assign(f2,  'flight.out'); { 输出文件名串与文件变量连接，输出文件写准备 }
    rewrite(f2);
    readln(f1,  n);                { 读入待规划的航线数 }
    n2 := 1;                       { 从第一条航线开始规划 }
    while n2 <= n do               { 若尚有未规划的航线，则循环 }
    begin
        readln(f1,  k);            { 读当前航线的地域数 }
        for i := 1 to k do         { 读每个地域的长度、低空风速和高空风速 }
        readln(f1,  idata[i, 1],  idata[i, 2],  idata[i, 3]);
        for j := 20 to 40 do       { 计算飞机由地面升至地域1的每一个高度的耗油量 }
        data[1, j] := getgallon(1, j, 0);
        for i := 2 to k do         { 从地域2开始依次搜索每一个地域i }
          for j := 20 to 40 do { 依次搜索当前地域i的每一个高度j }
            begin
            data[i, j] := 1e20;
            {在确定地域i的飞行高度为j的情况下，依次设定地域i-1的飞行高度l }
            for l := 20 to 40 do
              begin
              { 计算data[i, j]=min{data[i-1, l]+ 飞机由地域i-1的l高度升（降）
                至地域i的j高度的耗油量，将满足上式的l存入记忆表data2[i, j] }
                t := getgallon(i, j, l)+data[i-1, l];
                if t < data[i, j] then
                    begin
                     data[i, j] := t;
                     data2[i, j] := l;
                    end;  {then}
              end;  {for}
        end;  {for}
      t := 1e20;
```

```
        for j := 20 to 40 do
        {依次设定地域k的飞行高度；将耗油量最少的飞行高度存入data3[k]}
          if data[k，j] < t then
          begin
              t := data[k，j];
              data3[k] := j;
          end;  {for}
          for i := k downto 2 do  { 从地域k出发，依次倒推每一个地域的飞行高度 }
          data3[i-1] := data2[i，data3[i]];
          write(f2，'Flight'，n2，':');
          for i := 1 to k do write(f2，data3[i]:3);{依次输出每一个地域的飞行高度}
          writeln(f2，''，data[k，data3[k]]:0:0); { 输出总耗油量 }
          inc(n2);                                  { 累计已规划的航线数 }
      end;  {while}
      close(f1);                                   { 关闭输入文件和输出文件 }
      close(f2);
end.  {main}
```

§1.3 是金还是铅

试　题

【英文原稿】

Input file：leadgold.in

How to make gold from lead has baffled alchemists for centuries. At the last Alchemists Club Meeting (ACM)，a sensational breakthrough was announced. By mixing the three chemicals Algolene， Basicine and Cobolase in the correct ratio， one can create a mixture that transforms lead into gold. Since Algolene，Basicine and Cobolase (or A,B,C for short) are generally not sold individually， but rather mixed into solutions， this may not be easy as it seems.

Consider the following example. Two mixtures of Algolene， Basicine and Cobolase are available，in ratios of 1:2:3 and 3:7:1，respectively. By mixing these solutions in a ratio 3:4:5, if we additionally had a solution of ratio 2:1:2， then 3:4:5 mixture would be possible by combining eight parts of 1:2:3， one part of 3:7:1 and five parts of 2:1:2.

Determining which mixing ratio we can obtain from a given set of solutions is no trivial task. But， as the ACM has shown， it is possibly a very profitable one. You must write a program to find mixing ratios.

Input

The input file contains several test cases. The first line of each test case contains an integer n ($0 \leqslant n < 100$) that represents the number of mixtures in the test case. The next n lines each contains three non-negative integers a_i, b_i, c_i, specifying the ratio a_i, b_i, c_i in which A, B, C occur in the i^{th} mixture. At least one of these integers is positive for each mixture.

Finally, there is one line containing three not-negative integers a, b, c, which specify the ratio $a:b:c$ in the desired solution. At least one of these integers is positive.

The input file is terminated with the integer 0 on a line by itself following the last test case.

Output

For each test case, output the word "Mixture", followed by the ordinal number of the test case. On the next line, if it is possible to obtain the desired solution by mixing the input solutions, output the word "possible".Otherwise, output the word "impossible".

Input sample

```
2
1 2 3
3 7 1
3 4 5
3
1 2 3
3 7 1
2 1 2
3 4 5
0
```

Output sample

```
Mixture 1
impossible

Mixture 2
possible
```

【中文译稿】

输入文件名： leadgold.in

好几个世纪以来，如何从铅得到金一直困扰着炼金家们。在最近的一次炼金俱乐部会议（ACM）上，宣布了一个重大的突破。通过将三种化学药品 Algolene、Basicine 及 Cobolase

（简称 A、B、C）以正确的比例混和，可以生成一种能将铅转换成金的混合物，由于 A、B、C 一般不单独出售，而是混合成溶液出售，所以这个问题并不像看起来那么简单。

考虑以下的例子。有两种由 A、B、及 C 组成的混合物，比例分别为 1∶2∶3 和 3∶7∶1。将这两种混合物再以 1∶2 的比例混合，我们得到一种 A、B、C 的溶液，比例为 7∶16∶5。但没有办法能将这两种混合物混合成比例为 3∶4∶5 的新溶液。但如果我们增加一种溶液，其含有 A、B、C 三种物质的比例为 2∶1∶2，那么将八份的 1∶2∶3、一份的 3∶7∶1 和五份的 2∶1∶2 混合后就可以得到 3∶4∶5 的混合物。

决定将给定的一组溶液以何种比例混合并不是件轻松的任务。但是，正如 ACM 所表明的，这可能是一件利润丰厚的任务。你必须写一个程序以寻找混合比例。

输入

输入文件包含多个测试数据。每个测试数据的第一行包含一个整数 $n(0\leqslant n<100)$,代表了测试数据中混合物的数目。接下来的 n 行每行包含三个非负整数 a_i、b_i、c_i，指明了 A、B、C 在第 i 种混合物中出现的比例为 a_i：b_i：c_i。每种混合物的三个整数中至少有一个是正的。

最后，有包含三个非负的整数 a、b、c 的一行，指明了所需溶液的比例为 a∶b∶c。这三个整数中至少有一个是正的。

输入文件由最后一个测试数据之后仅含整数 0 的一行结束。

输出

对于每一个测试数据，输出单词“Mixture”,其后跟着测试数据的编号。在下一行中，如果能通过混合输入的溶液得到所需溶液，则输出单词“possible”.否则的话，输出单词“impossible”。

输入范例

```
2
1 2 3
3 7 1
3 4 5
3
1 2 3
3 7 1
2 1 2
3 4 5
0
```

输出范例

```
Mixture 1
```

impossible

Mixture 2
possible

算 法 分 析

该题实际上是一门线性代数题，要求选手通过计算机编程求解。

设 x_i 为第 i 种溶液取出的比例($x_i \geqslant 0$，$1 \leqslant i \leqslant n$)；

a_i，b_i，c_i 为第 i 种溶液中 A、B、C 三种物质的比例(a_i，b_i，$c_i \geqslant 0$，$1 \leqslant i \leqslant n$)；

x，y，z 为混合溶液中 A、B、C 三种物质的比例。

能否从铅中提炼出金子，就是判断下述非齐次线性方程组

$$a_1x_1+a_2x_2+\cdots\cdots+a_nx_n=x$$
$$b_1x_1+b_2x_2+\cdots\cdots+b_nx_n=y$$
$$c_1x_1+c_2x_2+\cdots\cdots+c_nx_n=z$$

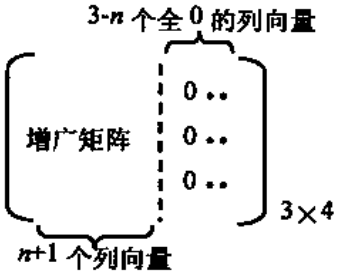
是否有解。

该非齐次线性方程组的增广矩阵为

常量 系数矩阵

$$A=\begin{bmatrix} x & a_1 \ldots & a_n \\ y & b_1 \ldots & b_n \\ z & c_1 \ldots & c_n \end{bmatrix}$$

如果 $n<3$，则增添$(3-n)$个全零的列向量，使得系数矩阵的规模扩充至 3×4。



这个非齐次线性方程组有解的充分必要条件是**增广矩阵与系数矩阵的秩相等**。所谓矩阵的秩就是矩阵中最大一个非零子行列式的规模。显然矩阵 A 的秩小于等于 3。正因为如此，我们每次仅取三种溶液，其他溶液不取，设这三种溶液的序号分别为 i，j，$k(i \neq j \neq k$，$1 \leqslant i$，j，$k \leqslant n)$，即

$$a_ix_i+a_jx_j+a_kx_k=x$$
$$b_ix_i+b_jx_j+b_kx_k=y$$
$$c_ix_i+b_jx_j+c_kx_k=z$$

混合溶液中 A、B、C 的比例

↓

对应的增广矩阵为 $B=\begin{bmatrix} x\ a_i\ a_j\ a_k \\ y\ b_i\ b_j\ b_k \\ z\ c_i\ c_j\ c_k \end{bmatrix}=\left[\begin{array}{c|ccc} b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{array}\right]$

↓

i，j，k 三种溶液中 A，B，C 的比例

我们采用主元消去法对矩阵 B 进行初等变换，使其系数矩阵变成一个单位矩阵

$$B'=\begin{bmatrix} x'100 \\ y'010 \\ z'001 \end{bmatrix}$$

注意，变换后非零的行向量可能会减少，初始时设行数 p_m=3。

我们从第 1 行开始搜索每一个行向量，若第 i 行（$1\leqslant i\leqslant p_m$）中的三个系数 b_{i1}，b_{i2}，b_{i3} 全 0，则分析同行的常量 b_{i0}：

1. 若 $b_{i0}<>0$，则不可能使 $b_{i1}*x_i+b_{i2}*x_j+b_{i3}*x_k=b_{i0}<>0$，因此方程组无解；

2. 若 $b_{i0}=0$，则消去第 i 行（i 行向量与 p_m 行向量对换，$p_m\leftarrow p_m-1$;$i\leftarrow i-1$）；

如果 l 行向量中的一个系数 $b_{lk}<>0$，则称 b_{lk} 为主元。通过下述变换关系式进行消元。变换后矩阵各元素为

$$b'_{ij}=\begin{cases} b_{lj}/b_{lk} & i=l \\ b_{ij}-(b_{ij}/b_{lk})b_{lk} & i\neq l \end{cases}$$

$$(1\leqslant i\leqslant p_m,\quad 1\leqslant j,\quad k\leqslant 3)$$

使得第 k 列向量中除第 l 元素 b_{lk}=1 外，其余元素为 0 。

上述一直进行至无解退出或者行指针 $i\geqslant p_m$ 为止（目前系数矩阵为单位矩阵或者产生 0 矩阵）。最后分析目前方程的常量 $b_{1,0}$，…，$bp_{m,0}$：如果都大于等于 0，说明变换后矩阵对应的方程组有解且符合题意。根据线性代数中矩阵初等变换的等价原理**“若矩阵 B 经有限次初等变换后变换成矩阵 B'，则 B 的行向量组与 B' 的行向量组等价”**，因此可以断定

$$a_ix_i+a_jx_j+a_kx_k=x$$
$$b_ix_i+b_jx_j+b_kx_k=y$$
$$c_ix_i+c_jx_j+c_kx_k=z$$
$$x_p=0 \qquad (p\neq i、j、k,\ 1\leqslant p\leqslant n)$$

有解。

例如，有五种溶液:

第一种溶液中 A、B、C 三种物质的比例为 a_1=1，b_1=2，c_1=3;

第二种溶液中 A、B、C 的比例为 a_2=3，b_2=7，c_2=2;

第三种溶液中 A、B、C 的比例为 a_3=2，b_3=1，c_3=2;

第四种溶液中 A、B、C 的比例为 a_4=1，b_4=2，c_4=3；

第五种溶液中 A、B、C 的比例为 a_5=2，b_5=3，c_5=4。

如果五种溶液配置成 A、B、C 的比例为 x=3，y=4，z=5 的特殊混合溶液，便可以提炼金子。求解过程如下。先列出对应的非齐次方程组和增广矩阵

$$x_1+3x_2+2x_3+x_4+2x_5=3$$

$$2x_1+7x_2+x_3+2x_4+3x_5=4 \qquad A=\begin{bmatrix}3&1&3&2&1&2\\4&2&7&1&2&3\\5&3&2&2&3&4\end{bmatrix}$$

$$3x_1+2x_2+2x_3+3x_4+4x_5=5$$

令 x_4=x_5=0，　得出

$$B=\begin{bmatrix}3&\underline{1}&3&2\\4&2&7&1\\5&3&2&2\end{bmatrix}\rightarrow\begin{bmatrix}3&1&3&2\\-2&0&\underline{1}&-3\\-4&0&-7&-4\end{bmatrix}\rightarrow\begin{bmatrix}9&10&11\\-2&0&1&-3\\-18&0&0&\underline{-25}\end{bmatrix}\rightarrow\begin{bmatrix}\frac{27}{25}&1&0&0\\\frac{4}{25}&0&1&0\\\frac{18}{25}&0&0&1\end{bmatrix}$$

即按照 $x_1=\frac{27}{25}$、$x_2=\frac{4}{25}$、$x_3=\frac{18}{25}$、x_4=x_5=0 的比例提取 5 种溶液，便可产生满足要求的混合溶液。

上述消元法可描述如下：

设 i 为行向量序号；　p_m 为目前非零的行向量数；

```
pm←3;      i←0;
while i<pm do
  begin
   i←i+1;
   if bi1‥bin 全 0
     then begin
          if bi0=0
            then begin
                 i 行向量与 pm 行向量对换;  pm←pm-1;  i←i-1;
                  end  {then}
            else  无解退出
          end{then}
      else begin{bi1‥bin 中有 bij<>0(1≤j≤n)}
          for r:=0 to n do bir←bir/bij;
          for r:=1 to pm do   if r<>i then for s:=0 to n do brs←brs-brj*bis;
          end;  {else}
     end;  {while}
if b1, 0‥bpm, 0 非全零 then  方程有解退出;
```

现有 n 种溶液。如果 n<3，则增添 3-n 个{0，0，0}作为列向量，然后按照上述算法进行消元，判断混合方案是否存在。问题是 n>3 时，究竟应该取哪三种溶液无法预知，因此只能枚举 n 种溶液中取 3 种溶液的全部组合。设三种溶液的一个组合序列为 m，其中 m[i]为组合中第 i 种溶液的一个序号（1≤i≤3）。规定组合中的 3 个溶液序号按递增顺序排列，

即 m[i]∈{m[i−1]+1 ‥ n−3+i}。只要其中一个组合 m 对应的方程组

$a_{m[1]}x_{m[1]}+a_{m[2]}x_{m[2]}+a_{m[3]}x_{m[3]}=x$

$b_{m[1]}x_{m[1]}+b_{m[2]}x_{m[2]}+b_{m[3]}x_{m[3]}=y$

$c_{m[1]}x_{m[1]}+c_{m[2]}x_{m[2]}+c_{m[3]}x_{m[3]}=z$

有解，即可断定混合方案存在。如果 c(n，3)个组合对应的方程组都无解，则断定混合方案不存在。为此，我们设计了一个 solve(s)过程，其中 s 为组合 m 的元素序号（1≤s≤3）：

```
procedure  solve (s);
 begin
  if  s>3
    then begin 对 m 组合对应的方程组进行消元变换，判断是否有解；end {then}
    else  for i:=m[s−1]+1 to n−(3−s) do
            begin
              m[s] ←i;
              solve(s+1);
              if m 组合对应的方程组有解 then exit;{then}
              m[s]←0;
            end; {for}
 end;  {solve}
```

在主程序中调用 solve(1)后，如果一个 m 组合对应的方程组有解，则输出“possible”；否则输出“impossible”。

程 序 题 解

```
program leadgold (input ,  output);
const  MaxN                    = 100;              { 溶液数的上限 }
       M                       = 3;                { 秩的上限或物质数 }
       inputname               = 'leadgold.in';  { 输入文件名串 }
       outputname              = 'leadgold.out'; { 输出文件名串 }
type   Texpr                   = array[0 .. Maxn] of real;
var    list                    : Array[1 .. m] of TExpr;{ 增广矩阵。其中list[i，0]:
       混合溶液中第i种物质的比例;list[i，j]：第j种溶液中第i种物质的比例（i>0）}
       n ，  p                 : integer;          { 溶液数，编号 }
       infp ，  outfp          : text;             { 输入文件变量和输出文件变量 }
       Possible                : Boolean;          { 混合方案存在标志 }
       master                  : Array[0 .. maxn] of integer;
       { n种溶液中取3种溶液的组合，其中master[i]为组合中第i个元素的溶液序号 }
Procedure initialize;  { 建立增广矩阵list }
var    i ，  j                 : integer;
begin
  fillchar (list ，  sizeof(list) ，  0);{ 增广矩阵初始化 }
  for i :=1 to n do                        { 顺序读入每一种溶液的信息 }
    begin
```

```
      for j :=1 to m do                    { 读入第i种溶液中三种物质的比例 }
        read (infp , list[j , i]);
      readln (infp);
    end; {for}
  for j :=1 to m do                        {读入混合溶液中三种物质的比例 }
    read (infp , list[j , 0]);
  readln (infp);
  if m >= n                                  { 若m≥n，则，增添（m-n）个全零的列向量 }
    then n :=m;
  possible :=false;                        { 混合方案的存在标志初始化 }
 fillchar (master , sizeof(master) , 0);{ 当前组合初始化 }
end; {initialize}

Procedure Check;      { 建立当前组合对应的子矩阵，通过消元变换判断是否有解 }
var     CExpr                  : Array[1 .. m] of Texpr;
{ 将组合中三种溶液对应的列向量取出构成一个3×4的矩阵，其中
              cexpr[i, j]=⎧ list[i, master[j]]   1≤j≤3
                          ⎩ list[i, 0]           j=0 }
Procedure GetCExpr;                          { 建立当前组合对应的子矩阵cexpr }
var     i , j , k              : integer;
begin
  fillchar (cexpr , sizeof(Cexpr) , 0);
  for i :=1 to m do
    begin
      for j :=1 to m do
        cexpr[i , j] :=list[i , master[j]];
      cexpr[i , 0] :=list[i , 0];
    end;  {for}
end; {GetCexpr}

Function HaveansCExpr : Boolean;
var     i , j , k , r , s    : integer;     { 辅助变量 }
        pm                   : integer;     { 非零的行向量数 }
        troot                : real;        { 辅助变量 }


Procedure SwapExpr (var expr1 , expr2 : TExpr);{行向量expr1和行向量expr2交换 }
var     tmp                  : TExpr;
```

```
begin
  tmp :=Expr1; expr1 :=Expr2; expr2 :=Tmp;
end; {SwapExpr}

begin
  pm :=m; i :=0;                                {非零的行向量数和行向量序号初始化 }
  While i < pm do
    begin
      inc (i);                                  { 搜索下一个行向量 }
      j :=1;                                    {统计第i个行向量中除常量外的零元素个数}
      While (j <= m) and (CExpr[i ,  j] = 0) do
        inc (j);
      if j > m                                  {若第i个行向量中除常量外都为零元素}
        then begin
               if CExpr[i ,  0] = 0             { 若常量亦为0，则消去第i行向量 }
                 then begin
                        swapExpr (CExpr[i] ,  CExpr[pm]);
                        dec (pm);
                        dec (i);
                      end   {then}
                 else begin                     { 否则返回方程无解信息 }
                        haveansCExpr :=False;
                        exit;
                      end; {else}
             end   {then}
        else begin                              {否则cexpr[i，j]为主元，进行消元运算}
               troot :=CExpr[i ,  j];
               {cexpr中的第i行元素除以主元，使得cexpr[i，j]=1}
               for r :=0 to m do
               {逐行运算，使得cexpr中的第j列上除[i，j]元素为1外，其余元素为0 }
                 CExpr[i ,  r] :=CExpr[i ,  r] / troot;
               for s :=1 to pm do
                 if s <> i then
                 begin
                   troot :=Cexpr[s ,  j];
                   for r :=0 to m do
                     CExpr[s ,  r] :=CExpr[s ,  r] -CExpr[i ,  r] * troot;
                 end;   {then}
             end;   {else}
```

```
    end;  {while}
  i :=1;                          { 判断cexpr的第0列上是否存在负元素 }
  While (i <= pm) and (Cexpr[i ,  0] >= 0)
    do inc (i);
  haveansCexpr :=i > pm;    { 若全部正整数，则返回有解信息 }
end; {HaveansCexpr}

begin
  GetCexpr;                       { 建立当前组合对应的子矩阵 }
  Possible :=HaveansCExpr;  { 通过对子矩阵的消元变换，判断方程是否有解 }
end; {check}

Procedure Solve (Step : integer);{从当前组合的第step个元素出发，判断方程是否有解 }
var     i ,  j ,  k                : integer;
begin
 if step> m{若当前组合形成，则对该组合对应的子矩阵进行消元变换，判断方程是否有解}
    then begin
           Check;
         end   {then}
    else begin
         for i :=master[step - 1] + 1 to n - (m - step) do
          begin                   { 否则枚举组合中第step个元素的所有可能值 }
            master[step] :=i;
           solve (step + 1);{ 递归搜索组合的下一个元素 }
            if possible then exit;  { 若方程有解，则回溯 }
               master[step] :=0;     { 组合中第step个元素值恢复为0 }
             end;  {for}
         end;  {else}
end; {Solve}

Procedure Printout;                       { 输出混合方案是否存在的信息 }
begin
  writeln (outfp ,  'Mixture ' ,  p);
  if possible
    then writeln (outfp ,  'possible')
    else writeln (outfp ,  'impossible');
end; {Printout}

begin
```

```
 Assign (infp ,  inputname); reset (infp);
{输入文件名串与文件变量连接，输入文件读准备 }
assign (outfp ,  outputname); rewrite (outfp);
{ 输出文件名串与文件变量连接，输出文件写准备 }
 p :=0;                              { 编号初始化 }
 repeat
   readln (infp ,  n);               { 读溶液数 }
   if n > 0                          { 若文件未结束 }
     then begin
            initialize;              { 建立增广矩阵 }
            inc (p);                 { 累计编号 }
            Solve (1);               { 从组合的第1个元素出发，判断方程是否有解 }
            Printout;                { 输出结果 }
          end;  {then}
  until n = 0;                       { 直至文件读完 }
  close (outfp);                     { 关闭输入文件和输出文件 }
  close (infp);
end. {main}
```

§1.4 通过关键字匹配检索网页

试　题

【英文原稿】

Input file: page.in

Anyone who has used the World Wide Web is familiar with search engines used to find pages matching a user-generated query. Many of these engines are quite sophisticated, using advanced algorithms and parallel searching techniques to provide fast, accurate responses.

This problem is somewhat simpler. A group of web pages has been classified by associating a list of keyword, given in decreasing order of relevance, with each page (i.e., the order of keywords is from the most specific keyword to the least specific). For example, a page on programming in Smalltalk has the keywords Smalltalk, programming, and computers in that order; the most relevant keyword is Smalltalk.

Queries also include a list of keywords, again from most to least relevant. For example, in a query consisting of the keyword Smalltalk following by the keyword computer, Smalltalk is more important than computer.

In this problem you are to determine the top five (or fewer) pages that match each of an arbitrary number of queries. To determine the strength of the relationship between a query and a web page, assume the keywords for each page and each query are assigned integer weights, in descending order, starting with N, where N is the maximum number of keywords allowed for a web page and query. The strength of the relationship is the sum of the products of the weights associated with each keyword that appears both in the web page list and the query list. For example, assume the following web pages and keyword lists:

Page 1: Smalltalk, programming, computers

Page 2: computers, programming

Page 3: computers, Smalltalk

For N equal 8, a query with keywords Smalltalk and programming in that order yields a strength rating of 113 for Page 1(8*8+7*7), 49 for Page 2(7*7), and 56 for Page 3(8*7). A query with keywords Smalltalk and computers yields a strength rating of 106 for Page 1(8*8+7*6), 56 for Page 2(7*8), and 112 for Pages 3(8*7+7*8).

Input

Input data consist of one line for each web page and query. A line consists of a code letter followed by a list of keywords. Code letter P, Q, and E denote a page, a query, and the end of file respectively. Code letters and keywords are separated by at least one space. Ps and Qs may occur in any order. Pages are added sequentially starting with one. Each page has at least one but no more than 8 keywords. Each word consists of no more than 20 alphabetic characters. The case of characters in the keywords is not significant. There will be a maximum of 25 pages in the input.

Each query also has a list of between one and eight keywords. Again, a keyword has no more than 20 alphabetic characters, case being insignificant. Number the queries sequentially starting with one.

Output

For each query, identify the 5(or fewer) pages read so far that are most relevant to the query. Print a single line containing the query identifier, a colon, and the page identifiers of the five most relevant pages in the decreasing order of relevance. Page identifiers consist of the letter "P" followed by the page number. Query identifiers consist of the letter "Q" followed by the query number. If several pages have the same relevance, list them by increasing page number. Do not list pages that have no relationship (zero strength), even if fewer than five pages are identified.

Input sample

P Smalltalk programming computers

P computers programming

P computers Smalltalk
P FORTRAN programming
P COBOL programming
P programming
Q Smalltalk
Q programming
Q computers
Q Smalltalk computers
Q Smalltalk programming
Q cooking French
E

Output sample

Query Pages
Q1: P1 P3
Q2: P6 P1 P2 P4 P5
Q3: P2 P3 P1
Q4 : P3 P1 P2
Q5: P1 P3 P6 P2 P4
Q6:

【中文译稿】

输入文件名：page.in

任何使用过万维网的人都熟悉用来寻找与用户生成的查询相匹配的页的搜索引擎。许多这样的搜索引擎已经非常成熟，使用高级算法和并行搜索技术以提供快速、精确的反应。

这个问题相对简单一些。一组网页已经用关键字列表分了类，关键字是按相关程度降序给出的（也就是说，关键字的顺序是从最相关的关键字到最不相关的关键字）。例如，关于用 Smalltalk 编程的一页有这样顺序的关键字 Smalltalk、programming 以及 computers；最相关的关键字是 Smalltalk。

查询也包括一个关键字列表，同样是从最相关到最不相关。例如，在一个由关键字 Smalltalk 之后接着关键字 computer 所组成的查询中，Smalltalk 比 computer 更重要。

在这个问题里，对于任意多个查询中的每个查询，你要去决定与之最匹配的 5（或更少）页。为了决定一个查询和一个网页之间的相关程度，假定对每页和每个查询的关键字都从 N 开始降序分配整数权值。这里 N 是一个网页或查询所允许的最大关键字数。相关程度为每个在网页列表和查询列表中同时出现的关键字所分配的权值的乘积之和。例如，假定有如下网页的关键字列表：

第一页：Smalltalk， programming ， computers
第二页：computers， programming

第三页 computers， Smalltalk

对于N等于8，一个关键字顺序为Smalltalk和programming的查询与第一页的相关程度是113(8×8+7×7)，与第二页的相关程度是49(7×7)，与第三页的相关程度是56(7×8)。关键字顺序为Smalltalk和computers的查询与第一页的相关程度为106（8×8+7×6），与第二页的相关程度为56（7×8），与第三页的相关程度是112（8×7+7×8）。

输入

输入数据由每个网页或查询各占一行组成。每一行包括一个代码字符及关键字的列表组成。代码字符P、Q和E分别指示了一页、一个查询和文件的结束。代码字符及关键字之间至少有一个空格隔开。P和Q的行可以以任意顺序出现。页编号从1开始按顺序增加。每页都有至少1个但不超过8个关键字。每个关键字由不超过20个字母字符组成。关键字中的字符不区分大小写。输入中至多有25页。

每个查询也有一个1个到8个关键字组成的列表。同样，每个关键字有不超过20个与大小写无关的字母字符。从1开始将查询按顺序编号。

输出

对于每个查询，确定到目前为止所读入的与查询最相关的5（或更少）页。打印一行，包括查询标识符、一个冒号、以及按相关程度降序排列的最相关的5页的标识符。页标识符由字母“P”及其后的页编号组成。查询标识符由字母“Q”及其后的查询编号组成。如果有若干页有相同的相关程度，则将它们按页编号顺序列出。不要列出没有任何关联（相关程度为0）的页，即使被标识的页少于5页。

输入范例

```
P Smalltalk programming computers
P computers programming
P computers Smalltalk
P FORTRAN programming
P COBOL programming
P programming
Q Smalltalk
Q programming
Q computers
Q Smalltalk computers
Q Smalltalk programming
Q cooking French
E
```

输出范例

```
Query    Pages
Q1:      P1  P3
Q2:      P6  P1  P2  P4  P5
Q3:      P2  P3  P1
Q4:      P3  P1  P2
Q5:      P1  P3  P6  P2  P4
Q6:
```

算 法 分 析

一、输入关键字

文件 f 中的 web 网页 P 和用户需求 Q 由关键字组成，关键字中的字母和数符大小写不分，关键字与关键字之间由空格分隔，最后一个关键字尾为换行符。每读入一个关键字前必须略过前面无用的空格符和换行符，其中换行符表明当前的 web 网页 P 或者用户需求 Q 已经结束。每读入关键字的一个字符，必须将之统一转换成大写形式后存入(今后匹配需要)，这个过程一直进行到再次读入空格符或者换行符为止。

我们用一个过程 Getkeyword(var keyword)输入关键字。其中变量参数 keyword 返回关键字串。若 keyword=''，表明当前的一个 web 网页 P 或用户需求 Q 输入完毕。

```
procedure getkeyword (var keyword);
    begin
     keyword←'';
     ch←'';
     while not eoln (f) and (ch='') do 从文件 f 中读一个字符 ch;
     if ch=''then   exit;
     keyword←upcase (ch);
     while (not eoln(f)) and   (ch<>'') do
      begin
        从 f 文件中读一个字符 ch;
        if ch <> ''then keyword←keyword+upcase(ch);
      end;{while}
   end;   {getkeyword}
```

二、输入 web 网页组和用户需求信息

web 网页组由若干个关键字序列组成，每一个关键字序列的开头字母为'P'。Web 网页组的结束标志为'E'。我们将 web 网页组存入 list 数组，其中 list[*i*, *j*]为第 *i* 个 web 网页 P_i 中的第 *j* 个关键字。由于 web 网页数和每一个 web 网页中的关键字个数未预先定义，因此设 key[*i*]为第 *i* 个 web 网页 P_i 的关键字个数；

n：web 网页组中的网页数。

初始时，key 数组和 n 清零。

```
n←0;
repeat
 从文件 f 中读一个字符 ch;
 if ch='p'
  then begin
       n←n+1
       repeat
        getkeyword(onekey);
        if onekey<>''
          then begin
               key[n]←key[n]+1;
               list[n，key[n]]←onekey;
               end; {then}
       until  onekey='';  {一个 web 网页读毕}
       end; {then}
   从文件 f 中空读一个回车;
 until  ch='E';
```

显然，上述算法计算出的 n 为 web 网页组中的网页数。

用户需求组也由若干个用户需求组成，每一个用户需求的开头字母为'Q'，用户需求组的结束标志为'E'。我们将当前一个用户需求 Q 存入 query 数组，其中 query[i]为当前用户需求 Q 的第 i 个关键字。由于未预先定义序列中的关键字个数，因此设 q 为关键字数。初始时 q=0。

```
q←0;
repeat
 getkeyword (onekey);
 if onekey<>''then
   begin
    q ←q+1;  query[q]←onekey;
   end;  {then}
until  onekey='';
```

显然，上述算法计算出的 q 为当前用户需求中的关键字个数。

三、计算 web 网页组的权值 rate

输入一个用户需求 query 后，将其中的每一个关键字分别与每一个 web 网页 j（$1\leqslant j\leqslant n$）中的关键字匹配。如果 query[i]与 list[j，k]不匹配，则该关键字的匹配权值为 0；否则该关键字为公共关键字，其匹配权值为

$$(8-(i-1))\times(8-(k-1))=(9-i)\times(9-k) \qquad (1\leqslant i\leqslant q,\quad 1\leqslant k\leqslant key[j],\quad 1\leqslant j\leqslant n)$$

上述计算公式与题意稍有不同的原因是，query 和 list[j]中第 1 个关键字的序号为 1，而试题中 web 网页和用户需求的第 1 个关键字序号为 0。将 web 网页 j 中的所有公共关键字的匹配权值累加起来，记为该网页对于用户需求的匹配权值 rate[j]　($1\leqslant j\leqslant n$)

$$rate[j]=\sum_{\substack{1\leqslant i\leqslant key[j]\\ 1\leqslant k\leqslant q}}(9-i)\times(9-k)\mid list[j,\ i]=query[k]$$

我们按下述算法计算所有 web 网页的匹配权值 rate[1]，rate[2]，…，rate[n]:

```
rate 数组清零;
for i:=1 to n do
  for j:=1 to key[i] do
    for k:=1 to q do
      if list[i, j]=query[k] then rate[i]←rate[i]+(9-j)*(9-k);
```

四、输出对于当前用户需求的前 5 个匹配权值最大的 web 网页号

由于 rate[*i*]=0 表明第 *i* 个 web 网页与当前用户需求 Q 无公共关键字，因此前 5 个匹配权值最大的 web 网页中若出现 rate 值为 0，则该网页不予输出。

设 *i* 为选中的 web 网页数；choose[*i*]为 web 网页 *i* 被选中的标志。每一次在目前未被选中(choose 值为 false)且 rate 值大于 0 的 web 网页中，选择 rate 值最大的一个网页 *j*，choose[*j*]←true。初始时所有网页的 choose 值设为 false。

```
choose 数组初始化为 false;  i←0;
while  (i<5)and(i<n) do
  begin
   k←0;
   for j:=1 to n do
    if (choose[j]=false) and (rate[j]>0) and ((k=0) or (rate[j]>rate[k])) then k←j;
   if k<>0
    then begin
          i←i+1;choose[k]←true;
          输出选中的网页号 k;
          end;  {then}
       else break;
   end;  {while}
```

有了上述基础，我们不难得出算法流程：

```
输入 web 网页组;
 repeat
  从文件 f 中读一个字符 ch;
  if ch='Q'
    then  begin
          用户需求的个数 m+1;
          输入用户需求 query;
          计算每一个 web 网页的权值;
          计算和输出对于 query 的 5 个匹配权值最大的网页序号;
          end; {then}
  until ch='E';
  关闭输入文件和输出文件;
```

程 序 题 解

```
{$A+, B-, D+, E+, F-, G-, i+, l+, N-, O-, P-, Q+, R+, S+, T-, V+, X+}
{$M 16384, 0, 655360}
Program Page (input ,  output);
```

```
const   MaxN                =25;          { web网页组中网页数的上限 }
        MaxKeylen           =20;          { 关键字长度的上限 }
        Maxkeyn             =8;           { 一个web网页中关键字数的上限 }
        inputname           ='Page.in';{ 输入文件名串 }
        Outputname          ='Page.out';{ 输出文件名串 }
        PRN                 =5;           { 输出的web网页数的上限 }

type    Tkey                =String[maxKeylen];        { 关键字类型 }
        TKeySet             =array[1..maxkeyn]of Tkey;{网页或用户需求序列的类型}
        Arrtype             =array[1 .. maxn] of longint;
                             { 匹配权值序列和网页中关键字数的类型 }

var     list                :Array[1 .. MaxN] of TkeySet;
                             { web网页组。list[i，j]：网页i中的第j个关键字 }
        KeyN ，  rate        :Arrtype;
        { keyn[i]：第i个web网页的关键字个数；Yate[i]：第i个web网页的匹配权值 }
        n                    :integer;     { 网页组中的网页数 }
        infp ，  outfp       :text;        { 输入文件变量和输出文件变量 }

Procedure GetKeyword (var rekey : Tkey); { 输入一个关键字rekey }
var     getchar              :char;
begin
  rekey :='';       { 关键字初始化为空 }
  getchar :=' ';  { 略过关键字前的无用空格或换行符 }
  while (not eoln (infp)) and (getchar = ' ') do
    read (infp ，  getchar);
 if getchar = "{ 若读入换行符，则返回关键字为空，表明读完一个网页（或用户需求）}
    then exit;
  rekey :=rekey + Upcase (Getchar);  { 第i个读入字符转换成大写后记入关键字 }
  While (not eoln (infp)) and (Getchar <> ' ') do
 { 依次读入字符，转换成大写后存入关键字，直至读入空格或换行符为止 }
    begin
      read (infp ，  getchar);
      if getchar <> ' '
        then rekey :=rekey + Upcase (Getchar);
    end;  {while}
end; {Getkeyword}

Procedure GetPage;                          { 读入一个web网页组 }
```

```
var     i ,  j ,  k                : integer;
        head                       : char;          { 关键字序列的开头字母 }
        onekey                     : TKey;          { 关键字 }
begin
  n :=0; fillchar (list ,  sizeof(list) ,  0);{ 网页数和网页组初始化 }
  fillchar (Keyn ,  sizeof(Keyn) ,  0);    {各网页的关键字个数和匹配权值初始化}
  fillchar (rate ,  sizeof(Rate) ,  0);
  assign (infp ,  inputname);       {输入文件名串与文件变量连接，输入文件读准备}
  reset (infp);
  repeat
    read (infp ,  head);                  { 读关键字序列的开头字母 }
    if head = 'P'                         { 若是web网页标志，则累计网页数}
      then begin
             inc (n);
             repeat
              GetKeyword (onekey);    { 读网页的一个关键字 }
              if onekey<>''
              { 若该网页未读完，则累计网页的关键字数，该关键字进入网页 }
                  then begin
                         inc (keyn[n]);
                         list[n ,  keyn[n]] :=onekey;
                       end;  {then}
             until onekey = '';
           end;  {then}
    readln (infp);
  until head = 'E';                       { 直至web网页组输入完毕 }
  close (infp);                           { 关闭输入文件 }
end; {Getpage}

Procedure Solve;                          {对输入的用户要求序列，计算和输出匹配结果 }
var     choose                  : Arrtype;{ choose[i]：网页i被选中标志 }
        i ,  j ,  k ,  m          : integer;
        head                    : char;   { 关键字序列的开头字母 }
        onekeyword              : Tkey;   { 关键字 }
        Query                   : TKeyset;{ 用户的关键字序列 }
        QKN                     : integer;{用户关键字序列中的关键字个数 }
begin
  m :=0;                                  { 编号初始化 }
  assign (infp ,  inputname); reset (infp);
```

```
{输入文件名串与文件变量连接，输入文件读准备 }
assign (outfp ， outputname); rewrite (outfp);
{ 输出文件名串与文件变量连接，输出文件写准备 }
writeln (outfp ， 'Query ' ， 'Pages');

repeat
  read (infp ， head);                          { 读关键字序列的开头字母 }
  if head = 'Q'                                 { 若是用户的关键字序列 }
    then begin
           inc (m);                              { 累计编号 }
           fillchar (Query ， sizeof(Query) ， 0);
           { 网页组的匹配权值序列、选中标志序列、用户关键字序列初始化 }
           fillchar (choose ， sizeof(choose) ， 0);
           fillchar (rate ， sizeof(rate) ， 0);
           qkn :=0;                              { 用户的关键字个数清零 }
           write (outfp ， 'Q' ， m ， ':');
           write (outfp ， ' ');
           repeat
             Getkeyword (onekeyword);      { 读用户的一个关键字 }
             if onekeyword <> ''
            { 若用户的关键字序列未读完，则累计关键字个数，关键字进入 query表 }
               then begin
                      inc (qkn);
                      Query[Qkn] :=onekeyword;
                    end;  {then}
           until onekeyword = '';           { 直至用户的关键字序列输入完毕 }
           for i :=1 to n do                { 顺序搜索每一个网页 }
             for j :=1 to KeyN[i] do        { 顺序搜索当前网页i的每一个关键字}
               for k :=1 to qkn do          { 顺序搜索用户的每一个关键字}
                 if list[i ， j] = query[k]
                  { 若网页i的关键字与用户的关键字匹配，则累计网页i的匹配权值 }
                   then begin
                          rate[i] :=rate[i] + (9 - j) * (9 - k);
                        end; {then}
           i :=0;                                { 选中的网页个数初始化 }
           while (i < Prn) and (i < n) do { 若未全部选完，则循环 }
             begin
               k :=0;                          { 选中的网页序号初始化 }
               for j :=1 to n do
```

```
                { 在目前未选中且匹配权值大于0的网页中选择一个权值最大的网页k }
                  if (choose[j] = 0) and (rate[j] > 0)
                    and ((k = 0) or (rate[j] > rate[k]))
                  then k :=j;
                if k <> 0
              { 若选中一个网页，则累计选中次数，设置该网页选中标志并输出网页序号 }
                  then begin
                         inc (i);
                         choose[k] :=1;
                         write (outfp , ' P' , k);
                       end  {then}
                  else break;      { 否则未选中网页的匹配权值全0，退出 }
              end;  {while}
            writeln (outfp);
          end;
    readln (infp);
  until head = 'E';                  { 直至用户关键字序列输入完毕 }
  close (infp); close (outfp);       { 关闭输入文件和输出文件 }
end; {Solve}

begin
  GetPage;                           { 输入网页组 }
  Solve;                             { 输入每一个用户要求序列，计算输出匹配结果 }
end. {main}
```

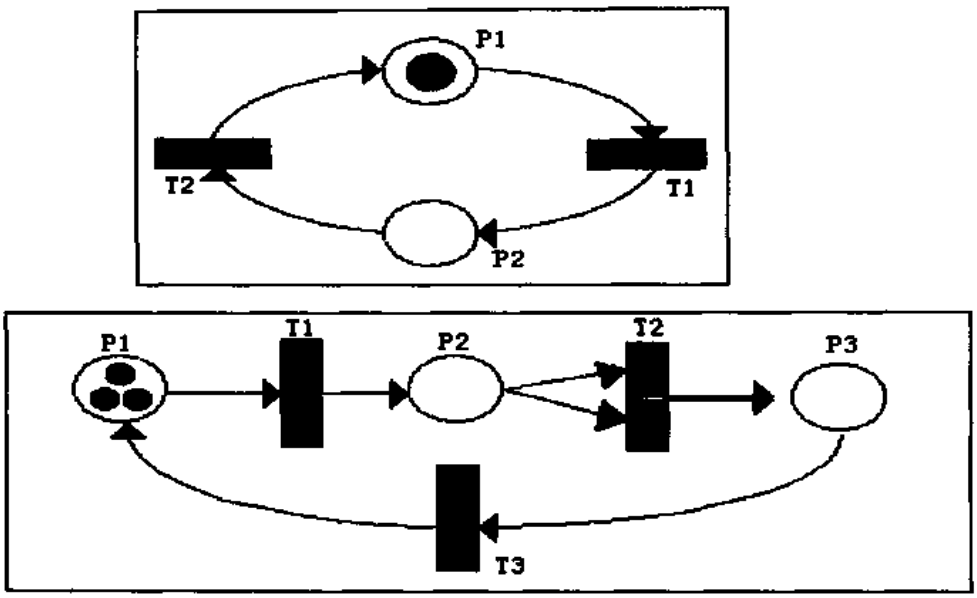
§1.5 Petri 网络模拟

试　题

【英文原稿】

Input file：petri.in

A Petri net is a computational model used to illustrate concurrent activity. Each Petri net contains some number of places (represented by circles)，transitions(represented by black rectangles)， and directed edges used to connect places to transitions，and transitions to places. Each place can hold zero or more tokens (represented by black dots). Here are two examples:



In the first Petri net above, there are two places (P_1 and P_2) and two transitions (T_1 and T_2). P_1 initially has one token; P_2 has none. P_1 is an input place for transition T_1, and P_2 is an output place for T_1. In the second example there are three places and three transitions, with three tokens in $P1$. T_2 has two input places, both of which are P_2.

Operation of a Petri net

Each transition in a Petri net is either enabled or disabled. A transition is enabled if there is at least one token in each of its input places. Any transition can fire whenever it is enabled. If multiple transitions are enabled, any one of them may fire. When a transition fires, one token is removed from each of the input places, and one token is added to each of the output places. This is effectively done atomically, as one action. When there are no enabled transitions, a Petri net is said to be dead.

In the top example only T_1 is enabled. When it fires one token is removed from P_1, and one token is added to P_2. Then T_2 is enabled. When it fires one token is removed from P_2, and one token is added to P_1. Clearly this Petri net will repeat this cycle forever.

The bottom example is more interesting. T_1 is enabled and fires, effectively moving a token to P_2. At this point T_1 is still the only enabled transition (T_2 requires that P_2 have two tokens before it is enabled). T_1 fires again, leaving one token in P_1 and two tokens in P_2. Now both T_1 and T_2 are enabled. Assume T_2 fires, removing two tokens from P_2 and adding one token to P_3. Now T_1 and T_3 are enabled. Continuing until no more transitions are enabled, you should see that only one token will be left in P_2 after 9 transition firings. (Note that if T_1 had fired instead of T_2 when both were enabled, this result would have been the same after 9 firings.)

In this problem you will be presented with description of one or more Petri nets. For each you are to simulate some specified number of transition firings , NF, and then report the number

of tokens remaining in the place .If the net becomes dead before NF transition firings, you are to report that fact as well.

Input

Each Petri net description will first contain an integer NP (0<NP<100) followed by NP integer specifying the number of tokens initially in each of the places numbered 1, 2, ..., NP. Next there will appear an integer NT (0<NT<100) specifying the number of transitions. Then for each transition (in increasing numerical order 1, 2, ..., NT) there will appear a list of integers terminated by zero. The negative numbers in the list will represent the input places, so the number $-n$ indicates there is an input place at n. The positive numbers in the list will indicate the output places, so the number p indicates an output place at p. There will be at least one input place and at least one output place for each transition. Finally, after the description of all NT transitions, there will appear an integer indicating the maximum number of firings you are to simulate, NF. The input will contain one or more Petri net descriptions followed by a zero.

Output

For each Petri net description in the input display three lines of output. On the first line indicate the number of the input case (numbered sequentially starting with 1) and whether or not NF transitions were able to fire. if so, indicate the net is still live after NF firings. Otherwise indicate the net is dead, and the number of firings which were completed. In either case, on the second line give the identities of the places which contain one or more tokens after the simulation, and the number of tokens each such place contains. This list should be in ascending order. The third line of output for each set should be blank.

The input data will be selected to guarantee the uniqueness of the correct output displays.

Input sample

```
2
 1 0
2
 -1 2 0
 -2 1 0
100
3
 3 0 0
3
 -1 2 0
 -2 -2 3 0
 -3 1 0
```

100
0

Output sample

Case 1: still live after 100 transitions
Places with tokens: 1 (1)

Case 2: dead after 9 transitions
Places with tokens: 2 (1)

【中文译稿】

输入文件名：petri.in

Petri 网是用来描述并发事件的一种计算模型。每个 Petri 网包括一些位置（用圆圈表示），变迁（用黑色矩形表示）和连接位置到变迁、变迁到位置的有向弧。每个位置拥有零个或多个的权标（用黑点表示）。图 1.5-1 是两个例子。

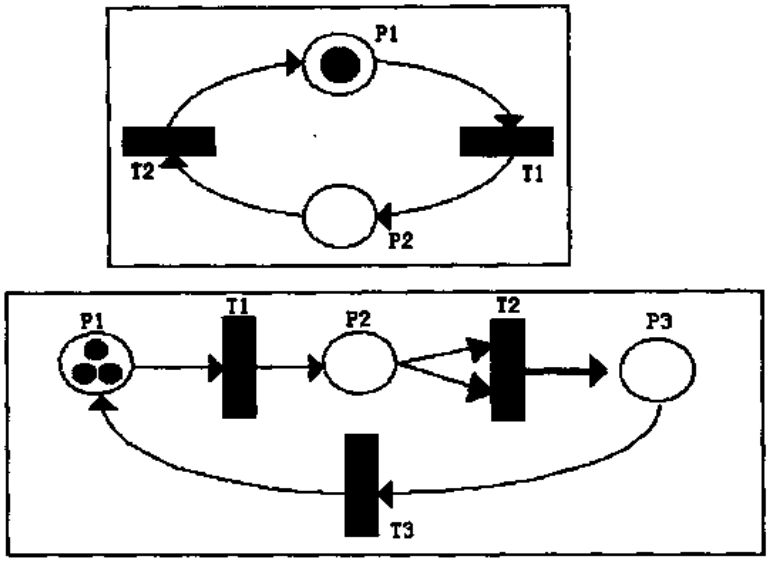


图 1.5-1

第一个 Petri 网有两个位置（P_1 和 P_2）和两个变迁（T_1 和 T_2）。初始时 P_1 有一个权标，P_2 没有权标。P_1 是变迁 T_1 的一个输入，P_2 是变迁 T_1 的一个输出。第二个例子中有三个位置、三个变迁以及 P_1 的三个权标，T_2 有两个输入位置，它们同为 P_2。

Petri 网的操作

Petri 网中的每一个变迁或者就绪或者失效。如果一个变迁的每一个输入位置中都至少有一个权标，这个变迁是就绪的。任何一个处于就绪状态的变迁都可以触发。如果有多个

变迁就绪，则其中的任何一个都可以触发。当一个变迁触发时，它的每个输入位置均减少一个权标，而每个输出位置将增加一个权标：以上过程将一次性自动有效地完成。当一个Petri 网中不存在就绪的变迁时，我们称这个 Petri 网停滞。

图 1.5-1 上方的例子中只有 T_1 就绪。当它触发时，P_1 移走一个权标同时有一个权标加入 P_2 中。这时 T_2 处于就绪状态。当它触发时 P_2 减少一个权标而且 P_1 增加一个权标。很明显，这个 Petri 网将这样永远循环下去。

底下的例子更加有趣。T_1 处于就绪状态并触发，P_2 移入一个权标。此时，T1 仍然是就绪的变迁（T_2 需要 P_2 有两个权标后才能就绪）。T_1 再一次触发，P_1 减少一个权标而 P_2 增加一个权标。这时 T_1 和 T_2 均处于触发状态。假设 T_2 触发，从 P_2 移走两个权标并给 P_3 加入一个权标。这时 T_1 和 T_3 均处于触发状态。如此继续下去直到没有就绪的状态，你可以看到经过 9 次变迁触发后只有一个权标留在 P_2 中（注意如果当 T_1、T_2 均就绪时用 T_1 触发代替 T_2 触发。则 9 次变迁触发后的结果也是一样的。）

本题中将给你一个或多个 Petri 网的描述，对于每一个 Petri 网，你要模拟给定次数 NF 的变迁触发，并报告此时存在于位置中的权标数。如果在 NF 变迁触发前该网已经停滞，则报告这一事实。

输入

在一个 Petri 网的描述中，首先是一个整数 NP（0<NP<100），接着有 NP 个整数，分别表示初始时每个位置（编号依次为 1，2，…，NP）中的权标数。接下来是一个整数 NT（0<NT<100）表示变迁的个数。对于每一个变迁（升序排列 1，2，…, NT）有一个以 0 结束的整数序列。序列中负数代表输入位置，即数 $-n$ 表示有一个编号为 n 的输入位置。序列中整数代表输出位置，即数 p 表示有一个编号为 p 的输出位置。每一个变迁至少有一个输入位置和一个输出位置。在所有 NT 个变迁的描述之后，最后是一个表示你要模拟的最大触发次数的整数 NF。输入文件包括一个或多个 petri 网的描述并以 0 结束。

输出

对于输入的每一个 Petri 网输出有三行。第一行为输入序号（从 1 开始连续的数）以及 NF 次变迁触发能否完成。如果能，显示网络在 NF 次触发后还能运转。否则显示网络停滞和已完成的触发数。不论哪种情况，第二行给出那些经过模拟后还含有权标的位置的标识以及它们所含的权标数。这个序列应按升序排列。第三行为一个空行。

每个输入数据都能保证有唯一的输出。

输入范例

```
2
 1 0
2
 -1 2 0
 -2 1 0
```

```
100
3
 3 0 0
3
 -1 2 0
 -2 -2 3 0
 -3 1 0
100
0
```

输出范例

```
Case 1: still live after 100 transitions
Places with tokens: 1 (1)

Case 2: dead after 9 transitions
Places with tokens: 2 (1)
```

算 法 分 析

一、输入 Petri 网的信息

Petri 网的信息包括：

1．位置中权标的情况

设位置的个数为 NP，每一个位置初始时的权标个数存放在 hold 表中，其中 hold[*i*]为位置 *i* 中的权标个数（1≤*i*≤NP）；

2．变迁的情况

设变迁的个数为 NT，每一个变迁的输入位置和输出位置信息存放在 trans 表中。由于一个变迁中某一位置输入或输出位置可能出现多次，因此设

trans[*i*，1，*j*]：在变迁 *i* 中，位置 *j* 作为输入的次数；

trans[*i*，2，*j*]：在变迁 *i* 中，位置 *j* 作为输出的次数；

（1≤*i*≤NT，　1≤*j*≤NP）

根据输入格式，trans 表的构造方法如下：

```
for i:=1 to nt do
 begin
   repeat
    读一个位置号 j;
    if j>0 then inc (trans[i，2，j])
           else inc (trans[i，1，j])
   until j:=0;
 end; {for}
```

二、确定一个就绪的变迁

所谓就绪的变迁是指每一个输入位置都有至少一个权标。如果变迁 *i* 中每一个输入位置 *j* 仅有一个权标的话，则变迁 *i* 中所有输入位置 *j* 的权标累加起来应为 trans[*i*，1，*j*]个。要使变迁 *i* 就绪，输入位置 *j* 中的权标数 hold[*j*]必须大于等于 trans[*i*，1，*j*]。

如果变迁上的每一个输入位置满足上述条件，则确定该变迁是就绪的；否则该变迁是不可触发的。我们可以通过下述算法在网络中寻找一个就绪的变迁 rep：

```
rep←0;
for i:=1 to nt do
 begin
  j←1;
  while (j≤np)and(hold[j]≥trans[i，1j]) do j←j+1;
  if j>np then begim   rep←i; break; end; {then}
 end; {for}
```

如果上述算法得出 rep=0，即所有的位置不可触发，则表明 petri 网已经“停滞”。

三、对就绪的变迁进行一次触发

如果找到一个就绪的变迁 *rep*(*rep*>0)，即可对该变迁进行一次触发：

对于变迁 *rep* 上的 trans[*rep*，1，*i*]个输入位置 *i* 来说，由于每一个输入位置减少一个权标，因此使得位置 *i* 的权标数 hold[*i*]减少了 trans[*rep*，1，*i*]个；对于变迁 *rep* 中的 trans[*rep*，2，*i*]个输出位置 *i* 来说，由于每一个输出位置增加一个权标，因此使得位置 *i* 的权标数 hold[*i*]增加了 trans[*rep*，2，*i*]个。

对变迁 *rep* 上的每一个位置进行上述操作，即

```
for i:=1 to np do
  begin
   dec (hold[i]，trans[rep，1，i]);
   inc (hold[i]，trans[rep，2，i]);
  end;{for}
```

便是一次触发。如果网络不“停滞”的话，这样的触发一共要进行 NF 次。

有了上述基础，便不难得出算法：

```
repeat
 寻找一个就绪的变迁 rep;
 if rep>0
   then begin
        触发次数 t+1;
        if t≤NF then  对变迁 rep 进行一次触发;
        end; {then}
until (rep=0) or (t>NF);
```

程序题解

```
{$A+, B-, D+, E+, F-, G-, i+, l+, N-, O-, P-, Q+, R+, S+, T-, V+, X+}
{$M 16384, 0, 655360}
Program PeTri (input , output);
const  MaxNp                = 100;                    { 位置数的上限 }
       MaxNt                = 100;                    { 变迁数的上限 }
       inputname            = 'Petri.in';             { 输入文件名串 }
       Outputname           = 'Petri.out';            {输出文件名串 }

type   arrPtype             = array[1 .. maxNp] of integer;
       arrTtype             = array[1 .. maxNt] of integer;
var    Trans                : Array[1 .. maxNt , 1 .. 2 , 1 .. MaxNp] of integer;
{ trans[i, 1, j]: 在变迁i中，位置j作为输入位置的次数;
  trans[i, 2, j]: 在变迁i中，位置j作为输出位置的次数 }
       np , nt              : integer;                { 位置数和变迁数 }
       hold                 : ArrPtype;               { hold[i]: 位置i的权标数 }
       infp , outfp         : text;                   { 输入文件名串和输出文件名串 }
       times                : integer;                { 触发次数 }
       nf                   : integer;                { 允许触发的次数 }
       m                    : integer;                { 编号 }

Procedure initialize;
var    i , j , k            : integer;
       getp                 : integer;
begin
  fillchar (trans , sizeof(trans) , 0);
  fillchar (hold , sizeof(hold) , 0);
  times :=0;                                           {触发移次数初始化 }
  for i :=1 to np do                                   { 输入每一个位置的权标个数}
    read (infp , hold[i]);
  readln (infp);
  readln (infp , nt);                                  { 输入变迁数 }
  for i :=1 to nt do                                   { 依次输入每一个变迁的信息}
    begin
      repeat
        read (infp , getp);                            { 读变迁i上的一个位置序号}
        if getp > 0          { 若位置为输出位置，则累计该输出位置在变迁i中的次数 }
```

```
            then begin
                    inc (Trans[i ,  2 ,  getp]);
                  end  {then}
            else if getp < 0 { 若位置为输入位置，则累计该输入位置在变迁i中的次数 }
                    then begin  {for}
                            inc (Trans[i ,  1 ,  -getp]);
                          end;{then}
        until getp = 0;                                    { 直至变迁i中的信息输入完毕 }
      end;  {for}
    readln (infp ,  nf);
end; {initialize}

Procedure Solve;
var     i ,  j ,  k              : integer;          { 辅助变量 }
        runt                     : integer;          { 就绪的变迁序号 }
Procedure GetRun (var ret : integer);          { 计算和返回一个就绪着的变迁序号}
var     i ,  j ,  k              : integer;
begin
  ret :=0;                                           { 就绪着的变迁序号初始化 }
  for i :=1 to nt do                                 { 顺序搜索每一个输入位置}
    begin
      j :=1;                          { 判断变迁i中的每一个输入位置是否都有至少一个权标}
      While (j <= np) and (hold[j] >= trans[i ,  1 ,  j])
        do inc (j);
      if j  > np                      { 如果是，则变迁i作为就绪的变迁ret返回 }
        then begin
               ret :=i;
               exit;
             end;  {then}
    end;  {foe}
end; {Getrun}

Procedure DoRun (int : integer);{ 对就绪的变迁int进行一次触发 }
var     i ,  j ,  k              : integer;
begin
 for i:=1 to np do
 {变迁int上的每一个输入位置减少一个权标、每一个输出位置增加一个权标 }
    begin
      dec (hold[i] ,  trans[int ,  1 ,  i]);
```

```
        inc (hold[i] ,  trans[int ,  2 ,  i]);
    end;  {for}
end; {DoRun}

begin
  repeat
    GetRun (runt);                    { 寻找一个就绪的变迁 runt }
    if runt > 0
        then begin{若就绪的变迁存在，则累计触发次数。若该次数未超过nf，则对
                   中转位置run进行一次触发 }
            inc (times);
            if times <= nf then DoRun (runt);
          end;  {then}
  until (runt = 0) or (times > nf);{ 直至petr网“停滞”或者触发次数超过nf为止 }
end; {Solve}

Procedure Printout;                   { 输出结果 }
var    i ,  j                  : integer;
begin
  write (outfp ,  'Case ' ,  m ,  ':');
  if times > nf                       { 若触发次数超过nf，则输出“可继续运转”信息 }
    then begin
        writeln (outfp ,  ' still live after ' ,  nf ,  ' transitions');
      end
    else begin                        { 否则输出“第times次触发后停滞”的信息 }
        writeln (outfp ,  ' dead after ' ,  times ,  ' transitions');
      end;  {else}
  write (outfp ,  'Place with tokens:');
  for i :=1 to np do                  { 输出每一个位置最后的权标数 }
    if hold[i] > 0
      then write (outfp ,  '' ,  i ,  '(' ,  hold[i] ,  ')');
  writeln (outfp);
  writeln (outfp);
end; {Printout}

begin
  m :=0;                              { 编号初始化 }
  assign (infp ,  inputname); reset (infp);
  { 输入文件名串与文件变量连接，输入文件读准备 }
```

```
  assign (outfp ,  outputname); rewrite (outfp);
  { 输出文件名串与文件变量连接，输出文件写准备 }
  repeat
    readln (infp ,  np);              { 读位置数 }
    if np > 0                          { 若文件未读完，则累计编号 }
      then begin
             inc (m);
             initalize;               { 读入petri网信息 }
             Solve;                    { 计算触发过程 }
             Printout;                { 输出结果 }
           end;  {then}
  until Np =0;                         { 直至文件输入完毕 }
  close (outfp);                       { 关闭输入文件和输出文件 }
  close (infp);
end. {main}
```

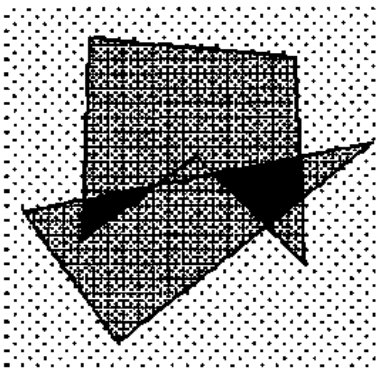
§1.6 多边形的相交

试　题

【英文原稿】

Input file：poly.in

Most drawing or illustration programs have simple tools for creating polygon objects. The better ones can find the regions that are the intersections of two polygons. The picture below shows two polygons，one is a pentagon and the other is a triangle. Their intersection is shown by the two dark regions.



IBM has just hired you as a member of a programming team that will create a very

sophisticated drawing/illustration program. Your task is to write the part of the program that deals with polygon intersections. Your boss has told you to delay work on the user interface and focus only on the geometric representations of the intersections.

A polygon in the Cartesian plane can be represented by a sequence of points that are its vertices. The vertices in the sequence appear in the order in which they are visited when traveling clockwise around the polygon's boundary, so any two adjacent vertices in the sequence are the endpoints of a line segment that is one of the polygon's sides. The last and the first vertices in the sequence are also endpoints of a side. Vertices are identified by their x- and y- coordinates. Assume the following about each polygon.

- No point will occur as a vertex (on the same polygon) more than once.
- Two sides can intersect only at a common endpoint (vertex).
- The angle between any two sides with a common vertex has a measure that is greater than 0 and less than 360.
- The polygon has at least 3 vertices.

The intersection of two polygons consists of 0 or more connected regions. Your problem is to take two polygons and determine the regions of their intersection that are polygons satisfying the criteria above.

Input

The input contains several data sets, each consisting of two polygons. Each polygon appears as a sequence of number:

$$n \quad x_1 \quad y_1 \quad x_2 \quad y_2 \ldots x_n \quad y_n$$

Where the integer n is the number of vertices of the polygon, and the real coordinates (x_1, y_1) through (x_n, y_n) are the boundary vertices. The end of input is indicated by two 0's for the values of n. These two 0's merely mark the end of data and should not be treated as an additional data set.

Output

For each data set, your program should output its number (Data set 1, Data set 2, etc.), and the number of regions in the intersection of its two polygons. Label each region in the data set (Region 1, Region 2, etc.)and list its vertices in the order they appear when they are visited going either clockwise or counterclockwise around the boundary of the region. The first vertex printed should be the vertex with smallest x-coordinate (to break ties, use the smallest y-coordinate). No region may include degenerate parts (consisting of adjacent sides whose angle of intersection is 0). If the three endpoints of two adjacent sides are collinear, the two sides should be merged into a single side. Print each vertex in the standard form (x, y), where x and y have two digits to the right of the decimal.

The following sample input contains exactly one data set. (The data set corresponds to the

illustration at the beginning of this problem description.)

Input sample

```
3  2  1  0.5  3.5  8  5
5  1.5  3  2  7  6.5  6.5  6.5  3.25  4  4.5
0
0
```

Output sample

```
Data Set 1
Number of intersection regions: 2
Region 1: (1.50，  3.00) (1.59，  3.72) (3.25，  4.05)
Region 2: (4.43，  4.29) (6.50，  4.70) (6.50，  4.00) (5.86，  3.57)
```

【中文译稿】

输入文件名：poly.in

大多数画图演示软件都有简单的建立多边形对象的工具。有些好的软件可以找出两个多边形的相交部分。图 1.6-1 有两个多边形，一个五边形，一个三角形。它们的相交部分用阴影表示。

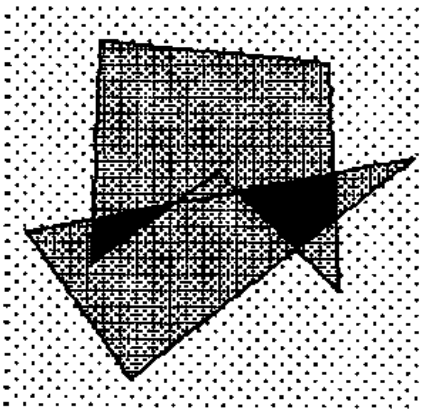


图 1.6-1

IBM 雇用你作为程序设计小组的一员来设计一个非常复杂的画图演示程序。你的任务是编写一个程序判断两个多边形的相交部分。你的老板让你先别作用户界面，以便集中精力完成相交的几何运算。

多边形可以用平面上一系列交点表示。这些点按照多边形上顺时针顺序排列，相邻两点是多边形一条边的两个端点，这些交点用（x，y）坐标表示。多边形有下列假设：

- 作为交点，没有点在同一多边形上出现两次或两次以上。
- 两边仅可能在顶点(交点)上相交。

●具有共同顶点的两条边的夹角在 0 到 360 之间。

●多边形至少有 3 个顶点。

相交部分可以是 0 或多个区域。你的任务是按照上面的规则计算出它们的相交部分。

输入

每个数据包含两行。每行为一个多边形。

以如下一系列数字表示：

n x_1 y_1 x_2 y_2 … x_n y_n

其中 n 是多边形的顶点数，（x_1，y_1）到(x_n，y_n)是每个顶点的坐标。输入数据以两个 0 结束。

输出

对于每项输入数据，你的程序应先输出其编号（Data set 1，Data set 2 等）和相交区域的数目。对每个区域输出一个编号（Region 1，Region 2 等）和每个顶点的坐标（按顺时针或逆时针排列）。输出的第一个顶点应取横坐标最小的（如相同则取纵坐标最小的）；不可使输出的任何区域包含夹角为 0 的相邻边。如果相邻两边的三个端点共线，则这两边应合并成一边。每个顶点输出格式为（x，y），其中 x 和 y 精确到小数点后 2 位。

输入范例

```
3  2  1  0.5  3.5  8  5
5  1.5  3  2  7  6.5  6.5  6.5  3.25  4  4.5
0
0
```

输出范例

```
Data Set 1
Number of intersection regions: 2
Region 1: (1.50，  3.00) (1.59，  3.72) (3.25，  4.05)
Region 2: (4.43，  4.29) (6.50，  4.70) (6.50，  4.00) (5.86，  3.57)
```

算 法 分 析

一、几个几何问题的分析

计算相交区域涉及几个几何问题。由于计算叉积和线段中点、判断两条线段是否相交、点是否在多边形内部等问题已在 § 1.1 晶体的清理一节中作了介绍，因此这里不再赘述。除此之外，还有

1．求两条相交线段的交点

设线段 $\overline{P_1P_2}$ 与 $\overline{P_3P_4}$ 相交，交点为 P。

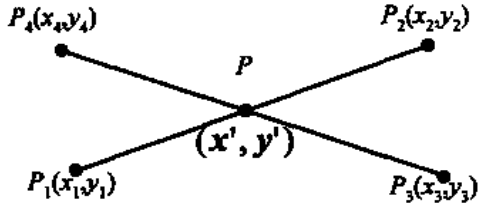


图 1.6-2

$\overline{P_1P_2}$ 所在的直线方程为 $a_1x+b_1y+c_1=0$，其中 $a_1=y_1-y_2$　$b_1=x_2-x_1$　$c_1=x_1*y_2-y_1*x_2$

$\overline{P_3P_4}$ 所在的直线方程为 $a_2x+b_2y+c_2=0$，其中 $a_2=y_3-y_4$　$b_2=x_4-x_3$　$c_2=x_3*y_4-y_3*x_4$

由于 (x',y') 经过 $\overline{P_1P_2}$ 和 $\overline{P_3P_4}$，因此

$$a_1x'+b_1y'+c_1=0$$
$$a_2x'+b_2y'+c_2=0$$

由此得出

$$x'=\frac{c_2b_1-b_2c_1}{b_2a_1-b_1a_2}\qquad y'=\frac{c_1a_2-c_2a_1}{b_2a_1-b_1a_2}$$

讨论：

(1) 若 $a_1b_2-a_2b_1\neq 0$，则 $\overline{P_1P_2}$ 与 $\overline{P_3P_4}$ 的交点为 (x',y')

(2) 若 $a_1b_2-a_2b_1=0$，则分析，如果（$c_2b_1-b_2c_1\neq 0$）或者（$c_1a_2-c_2a_1\neq 0$），则 $\overline{P_1P_2}$ 与 $\overline{P_3P_4}$ 平行；如果（$c_2b_1-c_1b_2=0$）并且（$c_1a_2-c_2a_1=0$），则 $\overline{P_1P_2}$ 与 $\overline{P_3P_4}$ 重合；

显然，如果两个多边形有边相交，则交点势必属于相交区域上的顶点。值得提醒的是，按上述公式交点 P 未必同时在线段 $\overline{P_1P_2}$ 和 $\overline{P_3P_4}$ 上，它仅是这两条线段所在的直线方程的交点（如图 1.6-2）。

2．判断顶点 P 是否在多边形 Q 的内部或者边上

设 $P=(x',y')$。过 P 点作一条斜率为 π 的测试射线，其直线方程为 $\pi(x-x')-(y-y')=0$。分析多边形 Q 的每一条边：

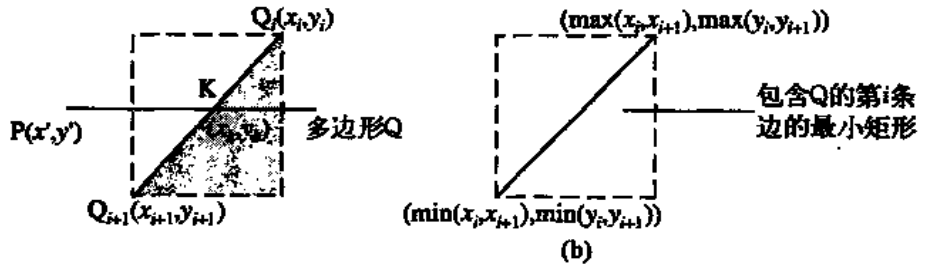


图 1.6-3

如果测试射线 l 与 Q 的第 i 条边有唯一交点 K，并且 K 在测试射线的右方向上（K 位于包含 Q 的第 i 条边的最小矩阵内且 $x_k \geqslant x'$）。

① 如果 K 与 P 重合，则断定顶点 P 在多边形 Q 的边上。

② 否则说明测试射线在过 P 的右方向上且与多边形 Q 相交，相交次数＋1。

如果测试射线在过 P 的右方向上与多边形 Q 相交奇数次，则断定顶点 P 在多边形 Q 内部。

```
相交次数←0;
过 p 作一条射线 l=π(x-x')-(y-y');
for i:=1 to nq do
  begin
   if l 与多边形 q 的第 i 条边有唯一交点 k
     then begin
          if (k 在包含第 i 条边的最小矩形内)and(xk≥x')
            then begin
                 if k 与 p 重合
                  then begin 断定 p 在多边形 q 的边上；退出计算；end{then
                 相交次数＋1；
                 end; {then}
             end; {then}
     end; {for}
  if 相交次数为奇 then 断定 p 在多边形 q 的内部;
```

3．判别相邻两边的三个端点是否共线

设 $\overline{P_1P_2}$ 和 $\overline{P_2P_3}$ 为两条相邻边

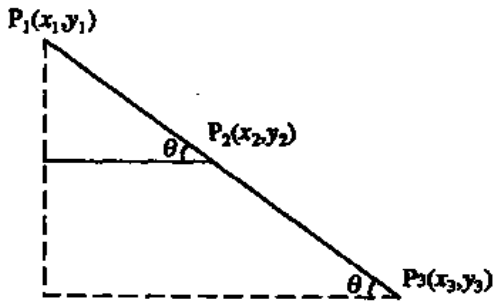


图 1.6-4

如果两条相邻边同时满足下述两个条件

(1) P_1、P_2 和 P_3 顺时针方向排列；

(2) 作一条线段 $\overline{P_1P_3}$，其斜率与线段 $\overline{P_1P_2}$ 的斜率相等。即

$$\frac{x_1-x_3}{y_1-y_3}=\frac{x_1-x_2}{y_1-y_2}$$

则判定 $\overline{P_1P_2}$ 与 $\overline{P_2P_3}$ 共线。

由于输出格式规定“不可使输出的任何区域包含夹角为 0 的相邻边。如果相邻两边的

三个端点共线，两边应合并成一边”。因此每产生一个相交多边形，在输出前必须对多边形中夹角为 0 的相邻边进行合并，即撤消位于相邻两边中间的一个端点。

设当前块中相交多边形的顶点序号按顺时针排列存放在 stack[*fr*]‥stack[*sp*]。若撤消序列中的顶点 *i* ，则 stack[*i*]← −stack[*i*];

t—相交多边形的顶点个数。初始时 *t*=*sp*−*fr*+1。每撤消一个顶点，则 *t*←*t*−1;

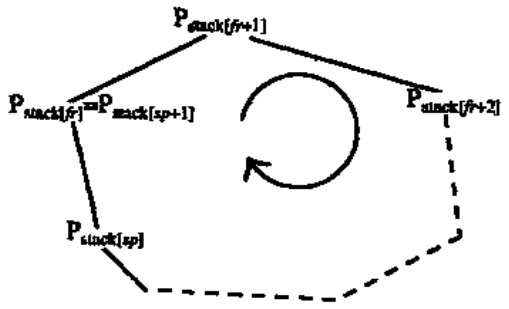


图 1.6-5

输出相交多边形前应作如下调整：

```
stack[sp+1]←stack[fr]  {使得相交多边形闭合}
t←sp-fr+1;
l←sp;
for i:=fr to sp do
 if P_STACK[L]P_STACK[i] 与 P_STACK[i]P_STACK[i+1] 共线
  then begin
       t←t-1;  stack[i]←-stack[i];
        end{then}
  else l←i;
stack[sp+1]←0;
```

4. 判断边是否位于多边形内部

设边为 $\overline{P_1P_2}$ ，该边未与任何多边形边相交。如果 $\overline{P_1P_2}$ 的两个端点（x_1，y_1）和（x_2，y_2）分别位于多边形边上，且该边的中点

$$x=\frac{x_1+x_2}{2} \qquad y=\frac{y_1+y_2}{2}$$

在多边形内部，则 $\overline{P_1P_2}$ 在多边形内（图 1.6-6(a)）；否则 $\overline{P_1P_2}$ 在多边形外（图 1.6-(b)）。

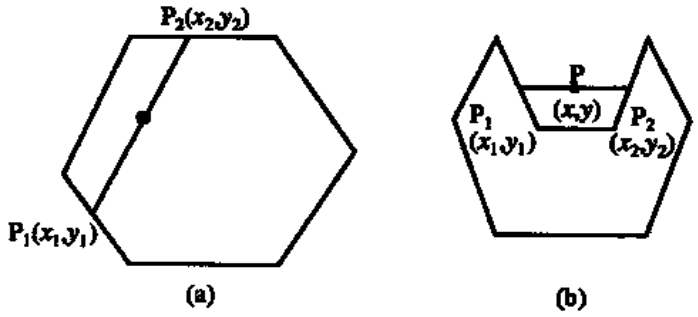


图 1.6-6

5．规定两个多边形相交信息的方向

一个多边形与另一个多边形相交的所有交点以及进入另一多边形内部的端点是有序的。按照输出格式，规定这些顶点按顺时针方向排列：

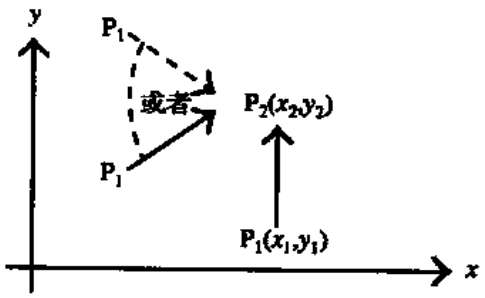


图 1.6-7

若 P_1 在 P_2 的左方（$x_1<x_2$）或者 P_1 在 P_2 的正下方（$x_1=x_2$）and($y_1<y_2$)，则称 P_1 在 P_2 的逆时针方向。

设一条多边形边上的相交信息为 $P=\{P_1, \cdots, P_n\}$，可通过下述运算将 P 中的顶点按顺时针方向排列：

```
for i:=1 to n-1 do
  for j:=i+1 to n do if Pj 在 Pi 的逆时针方向 then Pi 与 Pj 交换；
```

二、构造相交信息图 g

设多边形 $K=\{K_1, \cdots, K_{nk}\}$，多边形 $Q=\{Q_1 \cdots Q_{nq}\}$。

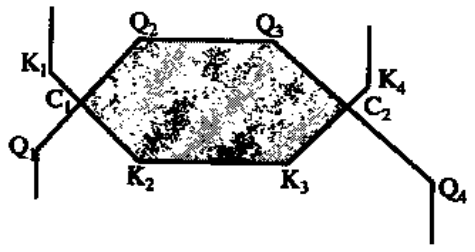


图 1.6-8

多边形 K 和 Q 相交区域的顶点来自：

1．多边形内部所含的另一个多边形边的顶点。例如多边形 Q 内所含的 K_2、K_3 和多边形 K 内所含的 Q_2、Q_3；

2. 两条多边形边相交的交点。例如 $\overline{K_1K_2}$ 和 $\overline{Q_1Q_2}$ 的交点 C_1，$\overline{Q_3Q_4}$ 和 $\overline{K_3K_4}$ 的交点 C_2；

每一个多边形将各自的交点和进入另一多边形的边端点组合起来，顺时针排列和相连。若形成的边在另一个多边形内部或边上，则该边一定属于相交多边形上的边。例如图 1.6-8 中，

$\overline{C_1Q_2}$　$\overline{Q_2Q_3}$　$\overline{Q_3C_2}$　(位于多边形 K)

$\overline{C_2K_3}$　$\overline{K_3K_2}$　$\overline{K_2C_1}$　(位于多边形 Q)

显然，按上述方法将两个多边形相交区域的所有顶点连接起来，便可形成一个由若干块组成的图 Q。设：

VP 表为存贮相交区域的不同顶点坐标（即两条多边形边相交的交点和位于另一多边形边上或内部的边端点），其下标指明了相交区域的顶点序号。VP 表中的顶点个数为 V_n；

SP 表为暂存当前边在相交区域的信息，即交点和位于另一多边形边上或内部的端点在 VP 表中的序号。SP 表中的顶点序号个数为 S_n；

值得提醒的是，VP 表中的顶点坐标互不相同。若当前相交区域的顶点与 VP 表中的某顶点重合，则该顶点不再进入 VP 表，而 VP 表中重合顶点的序号必须进入 SP 表。因此 V_n 与 S_n 不一定相同。

布尔矩阵 g 为相交区域信息图，图中的顶点序号存贮在 SP 表中。

$$g[i,\ j]=\begin{cases}\text{True} & \text{VP[}i\text{]与 VP[}j\text{]间有边相连}\\ \text{False} & \text{VP[}i\text{]与 VP[}j\text{]间无边相连}\end{cases}$$

构造 g 图的方法如下：

```
g 图初始化为 false;
Vp 表清零;    Vn←0;
for k:=1 to 2 do  {顺序搜索两个多边形}
 begin
  q←3-k;   {确定另一个多边形的序号}
  for i:=1 to nk do  {分析多边形 k 的每一条边}
    begin
     SP 表清零;    sn←0
     if 多边形 k 第 i 条边的端点 ki 在多边形 q 内部或边上;
       Then begin ki 的坐标进入 VP 表; 其 VP 表的序号进入 SP 表; end ; {then}
     if 多边形 k 第 i 条边的端点 ki+1 在多边形 q 内部或边上;
       Then begin ki+1 的坐标进入 VP 表，其 VP 表的序号进入 SP 表; end {then}
     For j:=1 to nq do    {分析多边形 q 的每一条边}
       if 多边形 k 的第 i 条边与多边形 q 的第 j 条边相交;
         Then begin 交点坐标进入 VP 表，其 VP 表的序号进入 SP 表; end; {then}
     VP[sp[1]]‥VP[sp[sn]]按顺时针方向排列;
    For j:=1 to sn-1 do
```

```
    if SP[j]<>SP[j+1] and(VP [SP[j]]VP[SP[j+1]])在多边形 q 内部或边上
      then begin g[sp[j],sp[j+1]]←true; g[sp[j+1],sp[j]]←true;end;{then}
    end; {for}
  end;   {for}
```

三、搜索相交区域

计算相交区域实际上是求 g 图中的块，一个块至少含有一个相交多边形。

1．深度优先搜索 g 图中的一个块

设 stack 栈为顺序存贮已访问的顶点，其中 stack[*i*]为第 *i* 个已访问的顶点在 VP 表中的编号。栈首指针为 *sp*，当前相交多边形顶点编号序列的指针为 *fr*，即 vp[stack[*fr*]]‥vp[stack[*sp*]]为一个相交多边形的顶点坐标序列；

首先，我们从 VP 表中取得一个未访问顶点 root：

```
root←1;
while (root<vn) and (顶点 root 已访问) do   root←root+1;
root←root mod (vn+1);
```

置顶点 root 已访问标志并进入 stack 栈；然后搜索一个与 root 相连且未访问的顶点 *i*，顶点 *i* 置访问标志后入栈；再从顶点 *i* 出发，继续搜索一个与之相连且未访问过的顶点……依次类推，直至找到一个已访问过的顶点 *K* 为止。此时从 stack[1]出发，在栈中寻找一个已访问过的顶点 *K*，即 stack[*fr*]=*k*。

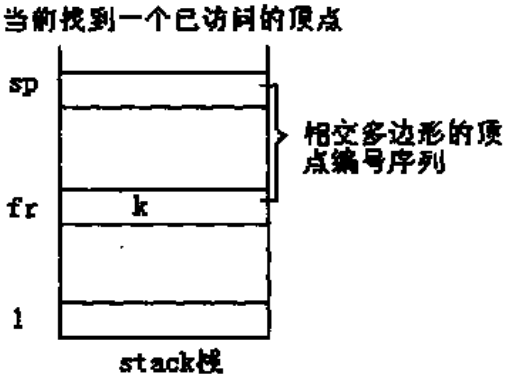


图 1.6-9

由此得出 stack[*fr*]‥stack[*sp*]为一个相交多边形的顶点编号序列。按照输出格式的要求，对序列中每相邻的三个端点进行判别，将夹角为 0 的相邻边合并成一条。输出时，应注意两点：

① 输出的第一个顶点应是横坐标最小（如相同，则取纵坐标最小的）。设该顶点在 stack 栈的下标为 *prs*。Prs 的计算方法如下：

```
prs←0;
for i:=fr to sp do
  if (stack[i]>0) and ((prs=0)or
```

(vp[stack[i]]在 vp[stack[prs]]的左方或者正下方)then prs←i;

② 相交多边形的编号序列为 stack[*fr*]‥stack[*sp*]，序列中的顶点个数为 *sp*-*fr*+1。由于序列表示一个闭合的多边形且指定 stack[prs]为第 1 个顶点编号，因此从 stack 栈中第 *prs* 个顶点数起，第 *j*（1≤*j*≤*sp*-*fr*+1）个顶点的序号 *prj* 应为

$$prj=\begin{cases} prs+j-1 & prj\leqslant sp \\ fr+(prs+j-1)-sp-1 & prj>sp \end{cases}$$

如果 stack[*prj*]>0，应输出 VP[stack[*prj*]]的坐标。

依次类推上述过程，直至求出当前块的所有相交多边形为止。为此，我们设计了一个过程 dfs (*root*)，从当前已访问顶点 *root* 出发，深度优先搜索 g 图中经过顶点 *root* 的一个块，并输出块中的所有相交多边形：

```
procedure dfs (root);
  begin
   for i:=1 to vn do
    if (g [root，i])
       then begin
          if 顶点 i 未访问
            then begin
                 置顶点 i 访问标志；
                 sp←sp+1;    stack[sp]←i;
                 dfs(i);
                 stack[sp]←0;  sp←sp-1;  {恢复 stack 栈递归前的状态}
                 end {then}
           else begin
                 从 stack[1]出发寻找 stack[fr]=i 的 fr;
                 调整 stack[fr]‥stack[sp]，将共线的三个相邻顶点的中间
                 顶点去掉（取负值）；
                 if 调整 stack[fr]‥stack[sp]后的顶点个数 t≥3
                   then begin
                        相交多边形数 total←total+1;
                        求应输出的第一个顶点序号 prs;
                        for j:=1 to sp-fr+1 do
                          begin
                          计算 stack 栈中第 prs 个顶点数起的第 j 个顶点的序号 prj;
                          if stack[prj]>0 then 输出 vp[stack[prj]]的坐标；
                          end; {for}
                    end;{then}
                   for j:=fr to sp do if stack[j]<0 then stack[j]←-stack[j];
              end; {else}
        end;{then}
end; {dfs}
```

2．求 g 图中的所有相交多边形

有了深度优先搜索一个块的基础，整个算法便变得显而易见了；每从 VP 表中取出一个未访问的顶点 *root*，*root* 进入 stack 栈并设置访问标志，然后通过递归调用 dfs(*root*)，

计算经过 *root* 的一个块，输出块中的所有相交多边形。如此循环往复，直至 VP 表中每一个顶点都已访问为止。此时的 *total* 便为 g 图中相交多边形的个数：

```
total←0;
置所有顶点未访问标志;
repeat
  从 vp 表中取出一个未访问的顶点 root;
  if root<>0
     then begin
          stack 栈清零;
          sp←1;  stack[sp]←root; 顶点 root 置访问标志;
          dfs (root);
          end; {then}
until root=0;
输出相交多边形的个数 total;
```

程 序 题 解

```
{$A+, B-, D+, E-, F-, G-, i-, l-, N-, O-, P-, Q-, R-, S-, T-, V-, X-}
{$M 65520, 0, 655360}
program poly (input ,  output);
const  maxn                  = 150;          { 两个多边形顶点数的上限 }
       maxgn                  = 240;          { 相交区域信息图中顶点数的上限 }
       inputname              = 'poly.in';    { 输入文件名串 }
       outputname             = 'poly.out';   { 输出文件史串 }

type   tpos                   = record        { 坐标类型 }
                                  x , y : real;
                                end;
       tline                  = record        { 直线方程类型ax+by+c }
                                  a , b , c : real;
                                end;
       tgraph          = array[1 .. maxgn , 1 .. maxgn] of boolean;{ 图的存贮类型 }
var    m                       : integer;      { 编号 }
       infp , outfp            : text;         { 输入文件变量和输出文件变量 }
       plist                   : array[1 .. 2 , 0 .. maxn + 1] of tpos;
      { 多边形的顶点序列表。其中plist[i, j]为多边形i中顶点j的坐标 }
       llist                   : array[1 .. 2 , 1 .. maxn] of tline;
      { 多边形的边序列表。其中llist[i, j]为多边形i 中第j条边的直线方程 }
       n                       : array[1 .. 2] of integer; {n[i]为多边形i的顶点数 }
       total                   : integer;      { 相交多边形的个数 }
```

```
function min (com1 ， com2 : real) : real; {计算和返回实数com1和com2中的较小者 }
begin
  if com1 < com2
    then min :=com1
    else min :=com2;
end; {min}

function max (com1 ， com2 : real) : real;  { 计算和返回实数com1和com2中的较大者 }
begin
  if com1 > com2
    then max :=com1
    else max :=com2;
end; {max}

function equalp (p1 ， p2 : tpos) : boolean;{ 若顶点p1和顶占p2重合，则返回true }
begin
  equalp :=(p1.x = p2.x) and (p1.y = p2.y);
end; {equalp}


procedure getline (p1 ， p2 : tpos; var rel : tline);
{ 计算和返回经过p1和p2两点的直线方程rel }
begin
   with rel do
    begin
      a :=p2.y-p1.y; b :=p1.x-p2.x; c :=p1.y*(p2.x-p1.x)-p1.x*(p2.y-p1.y);
    end;
end; {getline}

procedure initalize;                          { 输入两个多边形的顶点信息 }
var    i ， j ， k              : integer;
begin

  fillchar (plist ， sizeof(plist) ， 0);     { 两个多边形的顶点序列表初始化 }
  fillchar (n ， sizeof(n) ， 0);               { 两个多边形的顶点数初始化 }
  for i :=1 to 2 do                             { 依次读入每个多边形的信息 }
    begin
      read (infp ， n[i]);                      { 读入多边形i的顶点数 }
      for j :=1 to n[i] do                      { 依次读入多边形i中每个顶点的坐标 }
```

```
        read (infp , plist[i , j].x , plist[i , j].y);
      plist[i , 0] :=plist[i , n[i]];      { 使得多边形i闭合}
      plist[i , n[i] + 1] :=plist[i , 1];
      for j :=1 to n[i] do                  { 计算多边形i中每条边的直线方程 }
        getline (plist[i , j] , plist[i , j + 1] , llist[i , j]);
    end;  {then}
end; {initalize}

function caninline (p1 , p2 , p3 : tpos) : boolean;
{ 判断p1是否在包含p2、p3的最小矩形内。若是，返回true;否则返回false }
begin
  caninline :=(p1.x <= max (p2.x , p3.x)) and (p1.x >= min (p2.x , p3.x))
            and(p1.y <= max (p2.y , p3.y)) and (p1.y >= min (p2.y , p3.y))
end; {caninline}

procedure getcrossp (l1 , l2 : tline; var rep : tpos; var rec : integer);
{ 输入直线l1和l2。若两线不相交，则返回rec=0;若两线重合，则返回rec=2;
  若两线相交，则返回rec=1和交点rep }
var    d0 , d1 , d2          : real;
begin
  d0 :=l1.a*l2.b-l1.b*l2.a;
  d1 :=l1.b*l2.c-l1.c*l2.b;
  d2 :=l1.c*l2.a-l1.a*l2.c;
  if (d0 = 0)
    then begin
           if (d1 <> 0) or (d2 <> 0)
             then rec :=0
             else rec :=2;
         end  {then}
    else begin
           rep.x :=d1/d0;
           rep.y :=d2/d0;
           rec :=1;
         end;  {else}
end; {getcrossp}

function contain (comp : tpos; pp : integer) : boolean;
{ 判断顶点comp是否在第pp个多边形内部或者边上。若是，返回true;否则返回false }
var    i , j , k             : integer;  { 辅助变量 }
```

```
        testl                    : tline;     { 测试射线 }
        crossp                   : tpos;      { 交点 }
        counter                  : integer;   { 相交次数 }
        code                     : integer;   { 辅助变量 }

begin
  with testl do                              { 构造一条经过comp且与斜率为 π 的测试射线 }
    begin
      a :=pi; b :=-1; c:=comp.y - pi * comp.x;
    end;
  counter :=0;                               { 相交次数初始化 }
  for i :=1 to n[pp] do                      { 分析多边形pp的每一条边}
    begin
      getcrossp (testl , llist[pp , i] , crossp , code);
      { 计算测试射线与第i条边的交点crossp }
      if (code = 1)                          { 若有唯一交点 }
         then begin
              if caninline (crossp , plist[pp , i] , plist[pp , i + 1])
               and (crossp.x >= comp.x) { 若交点在测试射线上 }
              then begin
                 if equalp (crossp , comp){若comp在多边形pp的第i条边上，则返回true }
                    then begin
                         contain :=true;
                         exit;
                         end;  {then}
                 inc (counter);              { 计算测试射线与多边形pp相交的次数}
                   end;   {then}
                 end;  { then}
    end;  {for}
 contain :=odd (counter);{当且仅当相交次数为奇数时，comp在多边形pp 内部，返回true }
end; {contain}

procedure solve;                             { 计算和输出相交区域 }
var     g                    : tgraph;       { 相交区域信息图 }
        gn                   : integer;      { g图中的顶点数 }
        vlist                : array[1 .. maxgn] of tpos;
        { g图中的顶点序列。vlist[i]：g图中第i个顶点的坐标 }

function lessp (p1 , p2 : tpos) : boolean;
```

```
{ 若p1在p2的左方或者正下方，则返回true;否则返回false }
begin
  lessp :=(p1.x < p2.x) or ((p1.x = p2.x) and (p1.y < p2.y));
end; {lessp}

procedure makegraph;                              { 构造相交区域信息图g }
var    i , j , k , q          : integer;    { 辅助变量 }
       checkcnt               : array[1 .. 2 , 0 .. maxn + 1] of boolean;
       { checkcnt[i，j]=true 表示多边形i中的顶点j在另一个多边形内部或者边上 }
       sortv                  : array[1 .. maxgn] of integer;
       { 当前边的交点和进入另一个多边形的端点在vlist表中的序号 }
       sn                     : integer;        { sortv表的顶点个数 }
       crossp , midp          : tpos;           { 辅助顶点 }
       code                   : integer;        { 辅助变量 }

procedure getcheckcnt;                            { 计算checkcnt表 }
var    i , j , k              : integer;
begin
  fillchar (checkcnt , sizeof(checkcnt) , 0);
  for i :=1 to 2 do                               { 搜索每一个多边形 }
    begin
      for j :=0 to n[i] + 1 do                    { 搜索多边形i上的每一个顶点 }
        begin
          if contain (plist[i , j] , 3 - i)
       { 若多边形i上的顶点j在另一个多边形内部或边上，则置checkcnt[i，j]=true }
              then begin
                     checkcnt[i , j] :=true;
                   end;  {then}
        end;  {for}
    end;  {for}
end; {getcheckcnt}

procedure clearvlist;                             { vlist表初始化 }
begin
  fillchar (vlist , sizeof(vlist) , 0);
end; {clearvlist}

procedure clearsortv; { sortv表清零，表中的顶点个数初始化为0 }
begin
```

```
  fillchar (sortv ,  sizeof(sortv) ,  0);
  sn :=0;
end; {clearsortv}

procedure sortlist;   { 将sortv表中的顶点序号按顺时针方向排列 }
var     i ,  j ,  k                : integer;
procedure swapnum (var com1 ,  com2 : integer);   { 交换整数com1和com2}
var     tmp                        : integer;
begin
  tmp :=com1; com1 :=com2; com2 :=tmp;
end; {swapnum}

begin
  for i:=1 to sn - 1 do
    for j :=i + 1 to sn do
      if lessp (vlist[sortv[j]] ,  vlist[sortv[i]])
        then swapnum (sortv[j] ,  sortv[i]);
end; {sortlist}

procedure invlist (inp : tpos);
{若顶点inp在vlist表中，则序号存入sortv表；否则inp进入vlist表。其序号存入sortv表}
var     i ,  j ,  k                : integer;
begin
  i :=1;                                  { 计算vlist表中与顶点inp重合的序号i }
  while (i <= gn) and (not equalp (inp ,  vlist[i]))
    do inc (i);
  if i <= gn                              { 若vlist[i]与顶点inp重合，则序号i存入sortv表 }
    then begin
           inc (sn);
           sortv[sn] :=i;
         end{then}
    else begin                            { 否则inp进入vlist表，其序号存入sortv表 }
           inc (gn);
           if gn >= maxgn
             then writeln ('*');  { 显示vlist表溢出 }
           vlist[gn] :=inp;
           inc (sn);
           sortv[sn] :=gn;
         end;   {else}
```

```
end; {invlist}

begin
  getcheckcnt;                          { 计算checkcnt表 }
  fillchar (g ,  sizeof(g) ,  0);{ g图初始化 }
  gn :=0;
  clearvlist;  { 清除vlist表 }
  for k :=1 to 2 do                     { 顺序搜索每一个多边形 }
    begin
      q :=3 - k;                        { 确定另一个多边形的序号 }
      for i :=1 to n[k] do              { 顺序搜索多边形k的每一条边 }
        begin
          clearsortv;                   { 清除sortv表 }
          if checkcnt[k ,  i]{ 若多边形k的顶点i在多边形q内部或者边上，则顶点i的
          坐标进入vlist表，其序号进入sortv表 }
            then invlist (plist[k ,  i]);
          if checkcnt[k ,  i + 1]{ 若多边形k的顶点i+1在多边形q内部或者边上，则
          顶点i+1的坐标进入vlist表，其序号进入sortv表 }
            then invlist (plist[k ,  i + 1]);
          for j :=1 to n[q] do    { 搜索多边形q的每一条边 }
            begin
              getcrossp (llist[k ,  i] ,  llist[q ,  j] ,  crossp ,  code);
              if code = 1 { 若多边形k的第i条边与多边形q的第j条边有唯一交点crossp }
                then begin
                       if caninline (crossp ,  plist[k ,  i] ,  plist[k ,  i + 1])
                      and caninline (crossp ,  plist[q ,  j] ,  plist[q ,  j + 1])
                       then begin{ 则交点坐标进入vlist表，其序号进入sortv表 }
                              invlist (crossp);
                            end; {then}
                     end; {then}
            end; {for}
          sortlist;                     { 将sortv中的顶点序号按顺时针方向排列 }
          for j :=1 to sn - 1 do { 顺序搜索sortv表中的每一对顶点 }
            if sortv[j] <> sortv[j+1] then{ 若顶点j与顶点j+1不重合，且相连边的中
            点在多边形q内部或者边上，则确定该边为相交区域的一条边存入g图 }
            begin
              midp.x :=(vlist[sortv[j]].x + vlist[sortv[j+1]].x)/2;
              midp.y :=(vlist[sortv[j]].y + vlist[sortv[j+1]].y)/2;
              if contain (midp ,  q)
```

```
                    then begin
                            g[sortv[j] ,  sortv[j+1]] :=true;
                            g[sortv[j+1] ,  sortv[j]] :=true;
                          end; {then}
                end; {then}
            end; {for}
        end; {for}
end; {makegraph}

procedure printout (flag : integer);
{
                              flag={ 1 输出相交多边形数
                                   { 2 输出相交多边形的顶点序列
}
var     i ,  j ,  k              : integer;  { 辅助变量 }
        blank                    : array[1 .. maxgn] of boolean;
                                   { blank[i]=true，表明g图中顶点i已访问 }
        stack                    : array[0 .. maxgn] of integer;
                                   { 栈。顺序存放已访问的顶点 }
        sp                       : integer;  { 栈顶指针 }
        root                     : integer;  { 当前深度优先搜索树的根 }

procedure getroot;
{ 在g图中寻找一个未访问过的顶点root。若root=0，则g图中的所有顶点已访问 }
var     i ,  j ,  k              : integer;
begin
  root :=1;
  while (root <= gn) and (blank[root]) do inc (root);
  root :=root mod (gn + 1);
end; {getroot}


procedure dfsvisit (rp : integer);
{ 从未访问的顶点rp出发，深度优先搜索g图中的一个块，输出块中的所有相交多边形 }
var     i ,  j ,  k              : integer;
        prp ,  fromp             : integer;
        prj ,  prs               : integer;
```

```
procedure changestack;{ 调整，将stack栈中相邻三个顶点共线的中间一个顶点去掉 }
var     i ，  j ，  k                  : integer;
        lp                            : integer;

function oneline (p1 ，  p2 ，  p3 :tpos) : boolean;
{ 若P1P2 }的斜率与P1P3 的斜率相等，则返回true；否则返回false }
begin
    oneline :=((p1.y-p2.y)*(p1.x-p3.x)=(p1.y-p3.y)*(p1.x-p2.x));
end; {oneline}

begin
  stack[sp + 1] :=stack[fromp];                { 闭合当前的相交多边形 }
  prp :=sp - fromp + 1;                        { 计算相交多边形中的顶点个数 }
  lp :=sp;  { 从stack栈中的第sp个顶点出发，顺序分析相交多边形的每三个相邻顶点 }
  for i :=fromp to sp do
    if oneline (vlist[stack[lp]] ，  vlist[stack[i]] ，  vlist[stack[i + 1]])
      then begin
{若当前相邻的三个顶点共线，则去掉中间的一个顶点，使得相交多边形的边数减少一条}
                dec (prp);
                stack[i] :=-stack[i];
              end   {then}
      else lp :=i;
  stack[sp + 1] :=0;
end; {changestack}

procedure getprs;
{ 在当前相交多边形中取出横坐标最小（如相同，则取纵坐标最小）有一个顶点prs }
var     i ，  j ，  k                  : integer;
begin
  prs :=0;
  for i :=fromp to sp do
    if (stack[i] > 0)
      and ((prs = 0) or (lessp (vlist[stack[i]] ，  vlist[stack[prs]])))
    then prs :=i;
end; {getprs}

begin
  for i :=1 to gn do                  { 顺序搜索g图中的每一个顶点 }
```

```
if (g[rp ,  i])                { 若顶点rp与顶点i间有边相连 }
  then begin
        if not blank[i]    { 若顶点i未访问，则顶点i置访问标志并入栈 }
          then begin
                 blank[i] :=true;
                 inc (sp);
                 stack[sp] :=i;
                 dfsvisit (i);    { 从顶点i出发，继续深度优先搜索当前块 }
                 stack[sp] :=0;   { 恢复递归前的stack栈状态 }
                 dec (sp);
               end   {then}
          else begin                  { 否则顶点i已访问，输出一个相交多边形 }
               fromp :=1;             { 计算当前相交多边形在stack栈的指针from }
               while (fromp <= sp) and (stack[fromp] <> i) do   inc (fromp);
               changestack; {调整。将stack栈中相邻三个顶点共线的中间顶点去掉}
               if (prp >= 3){若调整后相交多边形的边数≥3，则累计相交多边形数}
                    then begin
                           inc (total);
                           if flag = 2
                             then begin
                               write (outfp ,  'region ' ,  total ,  ':');
                               getprs;
                             {计算当前相交多边形中应输出的第1个顶点序号prs}
                               for j :=1 to sp - fromp + 1 do
                               begin{ 计算从prs数起的第j个顶点的序号prj }
                                  prj :=prs + j - 1;
                                  if prj > sp then prj :=fromp + prj - sp - 1;
                                   if (flag = 2) and (stack[prj] > 0)
                           {若stack栈中第prj个顶点未去掉，则输出该顶点坐标}
                                       then begin
                                              with vlist[stack[prj]] do
                                               begin
                                                 write (outfp ,  '(' ,
                                                          x :0:2 ,  ', ' ,
                                                           y:0:2 ,  ')');
                                               end;{with}
                                       end; {for}
                               end;{then}
                               if flag = 2   then writeln (outfp);
```

```
                                    end; {then}
                                end; {then}
                            for j :=fromp to sp do  { 恢复去掉的顶点 }
                                if stack[j] < 0 then stack[j] :=-stack[j];
                            end; {else}
                    end   {then}
end; {dfsvisit}

begin
  fillchar (blank ,  sizeof(blank) ,  0);  { 置g图中所有的顶点未访问 }
  repeat
    getroot;                                    { 寻找g图中一个未访问的标志 }
    if root <> 0                                { 若存在一个未访问的顶点 }
      then begin
             fillchar (stack , sizeof(stack) ,  0);{ stack栈清零 }
             sp :=1; stack[1] :=root;              { 顶点root入栈并置访问标志 }
             blank[root] :=true;
             dfsvisit (root);
           {从root出发，深度优先搜索经过root的一个块，并输出块中的所有相交多边形}
           end;{then}
  until root = 0;                                  { 直至g图中的所有顶点已访问 }
  if flag = 1                                      { 输出相交多边形的个数}
    then writeln (outfp ,  'number of intersection regions: ' ,  total);
end; {printout}

begin
  makegraph;                                       { 构造相交区域信息图g }
  total :=0;
  printout(1); {print total number}                {输出每一个相交多边形的顶点序列}
  total :=0;
  printout(2); {print all regions }                { 输出相交多边形的个数 }
end; {solve}

begin
  assign (infp ,  inputname); reset (infp);
 { 输入文件名串与文件变量连接，输入文件读准备 }
  assign (outfp ,  outputname); rewrite (outfp);
 { 输出文件名串与文件变量连接，输出文件写准备 }
  m :=0;                                           { 编号初始化 }
```

```
  repeat
    initalize;                                  { 输入两个多边形的信息 }
    if n[1] > 0                                 {若文件未读完，则累计编号 }
      then begin
             inc (m);
             writeln (outfp ,  'data set ' ,  m);
             solve;                               { 计算和输出相交多边形 }
           end;  {then}
  until n[1] = 0;                               {直至文件输入完毕 }
  close (outfp);                                { 关闭输入文件和输出文件 }
  close (infp);
end. {main}
```

§1.7 奇异的结构

试　题

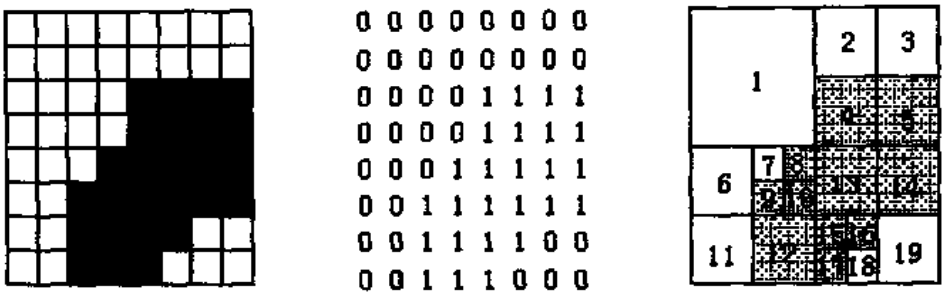
【英文原稿】

Input file: spatial.in

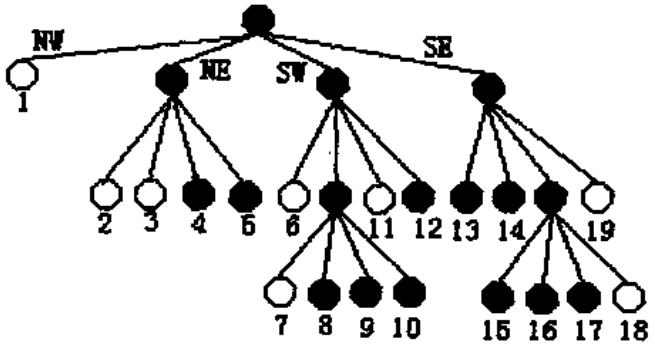
Computer graphics，image processing，and GIS (geographic information systems) all make use of a data structure called a quadtree. Quadtrees represent regional or block data efficiently and support efficient algorithms for operations like the union and intersection of images.

A quadtree for a black and white image is constricted by successively dividing the image into four equal quadrants. If all the pixels in a quadrant are the same color (all black or all white) the division process for that quadrant stops. Quadrants that contain both black and write pixels are subdivided into four equal quadrants and this process continues until each subquadrant consist of either all black or all white pixels. It is entirely possible that some subquadrants consist of a single pixel.

For example， using 0 for white and 1 for black， the region on the left below is represented by the matrix of zeros and ones in the middle. The matrix is divided into subquadrants as shown on the right where gray squares represent subquadrants that consist entirely of black pixels.



A quadtree is constructed from the block structure of an image. The root of the tree represents the entire array of pixels. Each non-leaf node of a quadtree has four children, corresponding to the four subquadrants of the region represented by the node. Leaf nodes represent regions that consist of pixels of the same color and thus are not subdivided. For example, the image shown above, with the block structure on the right, is represented by the quadtree below.



Leaf nodes are white if they correspond to a block of all white pixels, and black if they correspond to a block of all black pixels. In the tree, each leaf node is numbered corresponding to the block it represents in the diagram above. The branches of a non-leaf node are ordered from left-to-right as shown for the northwest, northeast, southwest, and southeast quadrants (or upper-left, upper-right, lower-left, lower-right) respectively.

A tree can be represented by a sequence of number representing the root-to-leaf paths of black nodes. Each path is a base 5 number constructed by labeling branches with 1, 2, 3, or 4 with NW=1, NE=2, SW=3, SE=4, and with the least significant digit of the base 5 number corresponding to the branch from the root. For example, the node labeled 4 has path NE, SW which is 32_5 (base 5) or 17_{10} (base 10); the node labeled 12 has path SW, SE or $43_5=23_{10}$; and the node labeled 15 has path SE, SW, NW or $134_5=44_{10}$. The entire tree is represented by the sequence of number (in base 10)

9 14 17 22 23 44 63 69 88 94 113

Write a program that converts images into root-to-leaf paths and converts root-to-leaf paths into images.

Input

The input contains one or more images. Each image is square, and the data for an image starts with an integer n, where $|n|$ is the length of a side of the square (always a power of two, with $|n|<64$) followed by a representation of the image. A representation is either a sequence of n^2 zeros and ones comprised of $|n|$ lines of $|n|$ digits per line, or the sequence of number that represent the root-to-leaf paths of each black node quadtree that represents the image.

If n is positive, the zero/one representation follows; if n is negative, the sequence of black node path numbers(in base 10)follows. The sequence is terminated by the number -1. A one-node tree that represents an all-black image is represented by the number 0. A one-node tree that represents an all-white image is represented by an empty sequence (no numbers).

The end of data is signaled by a value of 0 for n.

Output

For each in the input, first output the number of the image, as shown in the sample output. Then output the alternate form of the image.

If the image is represented by zeros and ones, the output consists of root-to-leaf paths of all black nodes in the quadtree that represents the image. The values should be base 10 representations of the base 5 path numbers, and the values should be printed in sorted order. If there are more than 12 black nodes, print a newline after every 12 nodes. The total number of black nodes should be printed after the path numbers.

If the image is represented by the root-to-leaf paths of black nodes, the output consists of an ASCII representation of the image with the character '.' used for white/zero and the character '*' used for black/one. There should be n characters per line for an $n \times n$ image.

Input sample

```
8
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 1 1 1 1 1
0 0 1 1 1 1 1 1
0 0 1 1 1 1 0 0
0 0 1 1 1 0 0 0
```

-8
9 14 17 22 23 44 63 69 88 94 113 -1
2
0 0
0 0
-4
0 -1
0

Output sample

image 1
9 14 17 22 23 44 63 69 88 94 113
Total number of black nodes=11

image 2
. . . . . .
. . . . . .
. . . . ****
. . . . ****
. . . *****
. . ******
. . ****. .
. . ***. . .

image 3
Total number of black nodes=0

image 4
****
****
****
****

【中文译稿】

输入文件名：spatial.in

计算机图形学、图像处理、地理信息系统都会用到一种数据结构叫做四叉树。四叉树有效地表示了地区数据；支持代数操作，例如图像的并和交。

黑白图像的四叉树是用把图像递归分成四等分的方法建立的：如果每 1/4 小块中的所

有点都有相同颜色（全黑或全白），它的分割工作到此结束。如果一个 1/4 小块中含有黑白两色的点，它就要再四等分。这样的操作直至每小块都由同色的点构成为止。一个小块只含有一点是完全可能的。

例如：用 0 表示白，用 1 表示黑。左图可用中图中由 0 和 1 构成的矩阵表示，它被分割成小块后由右图所示，其中灰方块表示全黑的小块。

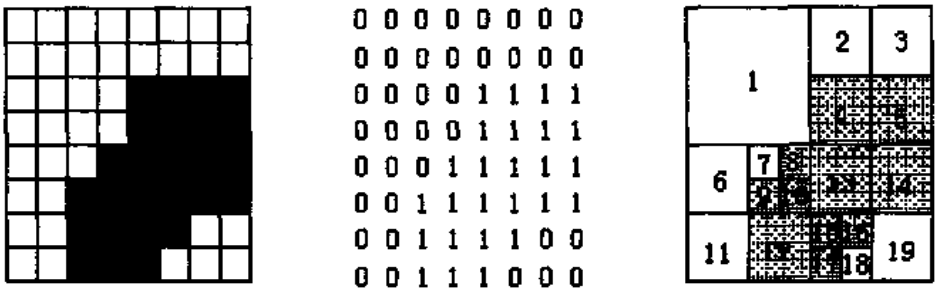


图 1.7-1

四叉树就是从图 1.7-1 右边这种块状结构建立的，树根表示点的全集，每个非叶结点都有 4 个子结点，分别表示这个结点所表示方块的四个等分小块，叶结点表示由于含同色点从而未被分割的方块，例如，图 1.7-1 右图所示的块状结构，可由如图 1.7-2 所示四叉树表示。

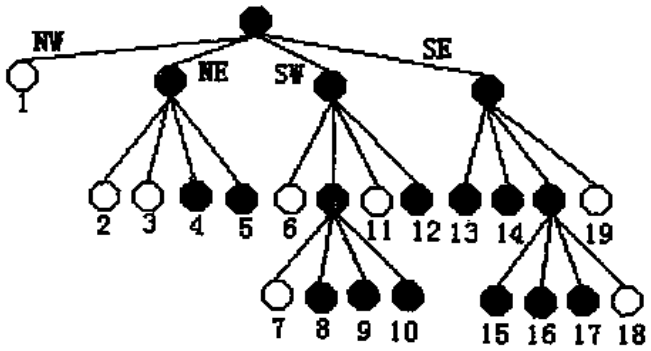


图 1.7-2

如果一个叶结点对应的方块都由白点构成，则涂为白色；如果它对应的方块都由黑点构成，则涂为黑色。树中每个叶结点按图 1.7-1 右图所示顺序编号。非叶结点的四个子结点的顺序是左上、右上、左下、右下。

四叉树可由表示从根结点到黑叶结点路径的数字序列来表示。每条路径是一个 5 进制的数，每位的 1、2、3、4 分别表示左上、右上、左下、右下，最低位对应的是由树根出

发的那一段。例如，4 号结点的路径为右上、左下，可用 5 进制数 32 表示，即十进制数 17；12 号结点的路径为左下、右下，可用 5 进制数 43 表示，即十进制数 23；15 号结点的路径为右下、左下、左上，用 5 进制数 134 表示，即十进制数 44。整棵树由这些十进制数构成的序列来表示。例如：

9 14 17 22 23 44 63 69 88 94 113

写一个程序，将图像和数字序列这两种格式互相转换。

输入

输入包括 1 个或多个图像。每个图像都是正方形，其数据由整数 n 开始，其中 $|n|$ 是正方形的边长（总是 2 的幂，并且 $|n|<64$），接着是这个图像的表示，它要么是一个长为 n^2 的 0-1 序列（分为 $|n|$ 行，每行 $|n|$ 位），要么是与图像对应的四叉树叶路径的黑叶结点的数字序列。

如果图像 n 是正的，则用 0-1 序列表示；如 n 是负的，则由表示根叶路径的数字序列表示，以-1 结束。一个单节点树（表示一个全黑的图像）用 0 表示，一个表示全白图像的单节点树用一个空序列表示。

输入数据 n=0 时为结束。

输出

对输入的每一个图像，先按输出范例所示输出其编号，再输出图像转换格式后的结果。

如果图像原用 0-1 表示，输出表示其四叉树根叶路径的数字序列，应为 10 进制，并要升序输出。如果序列长度超过 12，每行输出 12 个数，最后应输出黑节点的个数。

如果图像原用数字序列表示，输出图像的 ASCII 表示，‘.’表示白或 0，‘*’表示黑或 1。对一个 $n\times n$ 的图像，每行应有 n 个字符。

输入范例

```
8
00000000
00000000
00001111
00001111
00011111
00111111
00111100
00111000
-8
9 14 17 22 23 44 63 69 88 94 113 -1
2
00
```

0 0
-4
0 -1
0

输出范例

image 1
9 14 17 22 23 44 63 69 88 94 113
Total number of black nodes=11

image 2
. . . . . .
. . . . . .
. . . . ****
. . . . ****
. . . *****
. . ******
. . ****. .
. . ***. . .

image 3
Total number of black nodes=0

image 4
****
****
****
****

算 法 分 析

一、黑白图像转换为编码序列

1. 黑白图像对应一棵四叉树

黑白图像对应一棵四叉树，可通过对黑白图像递归分成四等分的方法建立这棵树：黑白图像作为根节点，第 1 次四等分后的四个 1/4 小块作为其儿子，其中左上小块作为第 1 个儿子，连接根节点的边编号为 1；右上小块作为第 2 个儿子，连接根节点的边编号为 2；左下小块作为第 3 个儿子，连接根节点的边编号为 3；；右下小块作为第 4 个儿子，连接

根节点的边编号为 4。如果 1/4 小块全白，则对应一个虚叶节点；如果 1/4 小块全黑，则对应一个实叶节点。实叶节点和虚叶节点对应的小块不再被四等分。如果 1/4 小块含有黑白两色的点，则对应一个非叶节点，需要继续对它四等分，由此扩展出 4 个子节点，4 个子节点的位置仍然按照对应等分小块左上、右上、左下、右下的顺序排列，连接父节点的边依次编号为 1、2、3、4。

依据上述办法不断对每一个非叶子节点对应的子图像分割，直至四等分后的每一个小块都由同色点构成（即分割出的 4 个子节点或为实叶节点或为虚叶节点为止）。

2. 图像编码的计算方法

将实叶节点通往根节点路径上的边编号组合起来，形成一个 5 进制数序列

$$(a_k a_{k-1}....a_1)_5$$

其中 a_i 表示第 i 次四等分后小块的位置序号（1：左上，2：右上，3：左下， 4：右下）。将这个 5 进制数据转换为十进制数

$$(a_k a_{k-1}....a_1)_5=\sum_{i=1}^{k} a_i \times 5^{i-1}$$

即为该实叶节点的编码，其中 5^{i-1} 表示第 i 次四等分的权。所有实叶结点的编码按升序要求排成一个序列，这个数字序列就是黑白图像的编码。

设当前待分割小块的左上角为 (i, j)，规模为 $s\times s$，简记为（i，j，s），该块由第 k 次分割产生，其通往根节点路径上所经过的边编号对应十进制数 d（简称路径值）。

若该块全 0，则对应虚叶节点。由于虚叶节点不存在编码，因此必须回溯至父节点，计算其他儿子的路径值；

若该块全 1，则对应实叶节点按升序要求将 d 插入图像编码序列 code 中；

若该块由 0、1 组成，则对应非叶节点，继续对该块进行第 k+1 次四等分：

左上的 1/4 块为（i，j，$\lfloor s/2 \rfloor$），路径值为 $d+5^k\times 1$;

右上的 1/4 块为（i，$j+\lfloor s/2 \rfloor$，$\lfloor s/2 \rfloor$），路径值为 $d+5^k\times 2$;

左下的 1/4 块为（$i+\lfloor s/2 \rfloor$，j，$\lfloor s/2 \rfloor$），路径值为 $d+5^k\times 3$;

右下的 1/4 块为（$i+\lfloor s/2 \rfloor$，$j+\lfloor s/2 \rfloor$，$\lfloor s/2 \rfloor$）路径值为 $d+5^k\times 4$;

显然上述转换过程是递归的，可以通过前序遍历四叉树的方法计算黑白图像的编码值。为此，我们设计了一个过程 pccode(k，d，i，j，s)，其中参值 k 表示四等分次数；d 表示目前路径值；i,j,s 表示待分割的子块，其左上角为（i，j），规模为 $s\times s$.

```
procedure pccode (k,d,i,j,s)
 begin
   if (i,j,s)块全 0 then exit;
   if (i,j,s)块全 1;
      then 按升序要求将 d 插入图像编码序列 code
      else begin
      pccode (k+1，d+5^k*1，i，j，⌊s/2⌋);
      pccode (k+1，d+5^k*2，i，j+⌊s/2⌋，⌊s/2⌋);
      pccode (k+1，d+5^k*3，i+⌊s/2⌋，j，⌊s/2⌋);
```

```
            pccode (k+1, d+5**4, i+⌊s/2⌋, j+⌊s/2⌋, ⌊s/2⌋);
                end; {else}
        end; {pccode}
```

由于待编码的图像是一个以（1，1）为左上角、规模为 $n\times n$ 的 01 方阵，因此主程序调用 pccode(0，0，1，1，n)后得出的 code 序列即为图像对应的编码。

二、编码序列转换为黑白图像

1. 一个实叶节点编码转换为全 1 子矩阵的办法

图像编码序列中的每一个十进制数 d 对应一个实叶节点的编码，我们通过除 5 取余法（即用 5 不断整除要转换的十进制数，直至商为 0 为止，将所得的各次余数以最后余数为最高位数依次排列，即得到所转换的五进制数），将一个实叶节点编码转换为五进制数。该数表示对（1，1）为左上角，规模为 $n\times n$ 的图像进行若干次四等分后产生一个全 1 子矩阵的过程：

d mod 5 表示第一次四等分产生的 1/4 块序号；$\lfloor d/5\rfloor$mod 5 表示第二次四等分产生的 1/4 块序号……直至$\lfloor d/5^k\rfloor$=0 为止，即图像经 k 次四等分后产生一个实叶节点，对应的全 1 子矩阵的序号为$\lfloor d/5^{k-1}\rfloor$mod 5。

显然，实叶节点编码转换为全 1 子矩阵的办法与图像转换为编码的方法一样，也是递归的，也可以通过前序遍历的方法转换。为此，我们设计了一个 fillmage(d，i，j，s)过程，其中值参 d 为实叶节点编码；i，j 表示图像左上角坐标；s 表示图像规模（图像简记为（i，j，s），d 将转换成（i，j，s）中的一个全 1 子矩阵）：

```
procedure fillmage (d, i, j, s);
 begin
  if d=0
    then 子矩阵（i，j，s）全填 1
    else begin
           k←d mode 5;
           case k do
           1: fillmage(⌊d/5⌋, i, j, ⌊s/2⌋);
           2: fillmage(⌊d/5⌋, i, j+⌊s/2⌋, ⌊s/2⌋);
           3: fillmage(⌊d/5⌋, i+⌊s/2⌋, j, ⌊s/2⌋);
           4: fillmage(⌊d/5⌋, i+⌊s/2⌋, j+⌊s/2⌋, ⌊s/2⌋);
           end; (case)
         end; {else}
  end; {fillmage}
```

显然只要调用 fillmag(d，1，1，n)，便可以将一个实叶节点编码 d 对应的了矩阵全填 1。

2. 将编码序列转换为图像

由于一个实叶节点编码对应一个全 1 子块、一个由实叶节点编码组成的序列表示一个完整的 01 矩阵，因此只要对序列中的每一个编码调用一次 fillmag 过程，便可完成转换：

以（1，1）为左上角的 n×n 矩阵清零；

```
repeat
  读一个实叶节点编码 c;
  if c≥0 then  fillmag(c, 1, 1, n);
until c<0;
```

按照输出格式，需将矩阵中的元素 1 改换成‘*’，表示黑点；将矩阵中的元素 0 改换成‘.’，表示白点；使得转换后的黑白图像更加形象：

```
for i:=1 to n do
  begin
    for j:=1 to n do
      if 矩阵的（i, j）元素为 1 then 输出‘*’
                                  else 输出‘.’;
    换行;
  end; {for}
```

程 序 题 解

```
{$a+, b-, d+, e+, f-, g-, i+, l+, n-, o-, p-, q+, r+, s+, t-, v+, x+}
{$M 16384, 0, 655360}
program spatial (input , output);
const  maxn              = 32;                              { 图像规模的上限 }
       deg               : array[0 .. 5] of longint         { deg[i]=5^i }
                         =(1 , 5 , 25 , 125 , 625 , 3125);
       inputname         ='spatial.in';                     { 输入文件名串 }
       outputname        ='spatial.out';                    { 输出文件名串 }

type   timage            = array[1 .. maxn , 1 .. maxn] of byte;{ 图像类型 }

var    n , m             : integer;                         { 图像规模，编号 }
       infp , outfp      : text;                            { 输入文件变量，输出文件变量 }
       image             : timage;                          { 图像矩阵 }
       code              : array[1 .. maxn * maxn] of integer;{ 图像的编码序列 }
       clen              : integer;                         { code序列的长度 }

procedure getimage;                                         { 读入图像 }
var    i , j             : integer;
       getchar           : char;
begin
  fillchar (image , sizeof(image) , 0);
  for i :=1 to n do                                         { 逐行读入 }
    begin
      for j :=1 to n do                                     { 逐列读入i行的元素 }
```

```
          begin
            read (infp , getchar);
            image[i , j] :=ord(getchar = '1');
          end; {for}
        readln (infp);
      end; {for}
end; {getimage}


procedure putcode;                                    { 将图像转换为编码序列并输出 }
var     i , j                   : integer;
function zeron (comi , comj , coms : integer) : integer;
{ 计算和返回以(comi，comj)为左上角，规模为coms*coms的子矩阵中0的个数 }
var     i , j                   : integer;
        ren                     : integer;
begin
  ren :=0;
  for i :=comi to comi + coms - 1 do
    for j :=comj to comj + coms - 1 do
      if image[i , j] = 0
        then inc (ren);
  zeron :=ren;
end; {nozero}

procedure addcode (incode : integer);
{ 按照升序要求将实叶节点的编码incode插入编码序列表code中 }
var     i , j                   : integer;
begin
  i :=1;
  while (i <= clen) and (code[i] < incode) do
    inc (i);
  for j :=clen downto i do
    code[j + 1] :=code[j];
  code[i] :=incode;
  inc (clen);
end; {addcode}

procedure pcode (step , df , li , lj , ls : integer);
{ 从step次四等分后的矩阵(li，lj ，ls)（图像的目标路径值为df）出发，递归计算图像的编
```

```
码序列 }
var     flag                    : integer;
begin
  flag :=zeron (li ,  lj ,  ls);              { 计算子矩阵中0的个数 }
  if flag = ls * ls                           { 若子矩阵全0，则回溯 }
    then exit;
if flag = 0        { 若子矩阵全1，则将实叶节点编码按升序要求插入编码序列表code中 }
    then begin
           addcode (df);
         end{then}
    else begin    { 否则递归处理左上子矩阵、右上子矩阵、左下子矩阵、右下子矩阵 }
         pcode (step+1,  df+deg[step]*1,  li,  lj,  ls div 2);
         pcode (step+1,  df+deg[step]*2,  li,  lj+(ls div 2),  ls div 2);
         pcode (step+1,  df+deg[step]*3,  li+(ls div 2),  lj,  ls div 2);
         pcode (step+1,  df+deg[step]*4,  li+(ls div 2),  lj+(ls div 2),  ls div 2);
         end;  {else}
end; {pcode}

begin
  fillchar (code ,  sizeof(code),  0);     { 编码序列表初始化 }
  clen :=0;                                   { 编码序列表中的编码个数初始化 }
  pcode (0 ,  0 ,  1 ,  1 ,  n);             { 从n*n的图像出发递归计算编码序列 }
  writeln (outfp ,  'image ' ,  m);
  if clen > 0                                 {若存在实叶节点的编码，则输出编码序列}
    then begin
           for i :=1 to clen - 1 do
             write (outfp ,  code[i] ,  ' ');
           writeln (outfp ,  code[clen]);
         end;  {then}
  writeln (outfp ,  'total number of black nodes = ' ,  clen);{ 输出实叶节点数 }
  writeln (outfp);
end; {putcode}

procedure getcode;     { 将编码序列转换为图像 }
var     i ,  j ,  k              : integer;
        getc                     : integer;
procedure fillimage (df ,  li ,  lj ,  ls : integer);
{ 根据实叶节点编码df值，将矩阵（$l_i$，$l_j$，$l_s$）中的某个子矩阵全填1}
var     i ,  j ,  k              : integer;
```

```
begin
  if df = 0              { 递归边界。将子矩阵(li，lj，ls)全填1 }
    then begin
           for i :=li to li + ls - 1 do
             for j :=lj to lj + ls - 1 do
               image[i ，  j] :=1;
         end
    else begin          { 否则计算边编号 }
           k :=df mod 5;
           case k of{根据边编号分情形处理左上矩阵、右上矩阵、左下矩阵、右下矩阵}
            1: fillimage (df div 5，  li，  lj，  ls div 2);
            2 : fillimage (df div 5，  li，  lj+ls div 2，  ls div 2);
            3 : fillimage (df div 5，  li+ls div 2，  lj，  ls div 2);
            4 : fillimage (df div 5，  li+ls div 2，  lj+ls div 2，  ls div 2);
           end;  {case}
         end;  {else}
end; {fillimage}

begin
  fillchar (image ，  sizeof(image) ，  0);              { 图像初始化 }
  repeat
    read (infp ，  getc);                                { 读一个实叶节点编码 }
    if getc >= 0
      then begin
             fillimage (getc ，  1 ，  1 ，  n);         { 将对应的子矩阵全填1}
           end;  {then}
  until getc < 0;                                        { 直至读完编码序列 }
  readln (infp);
end; {getcode}

procedure putimage;{ 将01矩阵中的0转换为白点标志，1转换为黑点标志 }
var    i ，  j                  : integer;
       putchar                 : char;
begin
  writeln (outfp ，  'image ' ，  m);
  for i :=1 to n do
    begin
      for j :=1 to n do
        begin
```

```
            if image[i , j] = 0
              then putchar :='.'
              else putchar :='*';
            write (outfp , putchar);
          end;  {for}
        writeln (outfp);
      end;  {for}
    writeln (outfp);
end; {putimage}

begin
  assign (infp , inputname); reset (infp);
  { 输入文件名串与文件变量连接，输入文件读准备 }
  assign (outfp , outputname); rewrite (outfp);
  { 输出文件名串与文件变量连接，输出文件写准备 }
  m :=0;                                    { 编号初始化 }
  repeat
    readln (infp , n);                      { 读图像规模 }
    if n > 0                                { 若图像转换为编码序列，则累计编号}
      then begin
             inc (m);
             getimage;                      { 读入图像信息 }
             putcode;                       { 计算和输出图像对应的编码序列 }
           end  {then}
      else if n < 0                         { 若编码序列转换为图像，则累计编号 }
             then begin
                    inc (m);
                    n :=-n;
                    getcode;                { 读入编码序列 }
                    putimage;               { 计算和输出编码序列对应的图像 }
                  end;  {then}
  until n = 0;                              { 直至文件输入完毕 }
  close (outfp);                            { 关闭输入文件和输出文件 }
  close (infp);
end. {main}
```

§1.8 指 数 塔

试 题

【英文原稿】

Input file： tower.in

One of the many problems in computer-generated graphics is realistically modeling the "orderly randomness" of things like mountain ranges and city skylines. A new student intern at a graphics company had an idea to use fluctuations in number representations to model height. In this problem you will compute several such number representations and show the "skylines" they produce.

Let n be any positive integer, and let b be an integer greater than or equal to 2. The complete base-b expansion of n is obtained as follows. First write the usual base-b expansion of n, which is just a sum of powers of b, each mltiplied by a coefficient between 1 and $b-1$, omitting terms with zero coefficients. For example, if n=20000 and b=3, the base-3 expansion of 20000 is given by

$$20000=3^9+3^5+2*3^3+2*3^2+2$$

To obtain the complete base-b expansion, we apply the same procedure to the exponents until all numbers are represented in base b. For n=20000 and b=3 we would have

$$20000=3^{3^2}+3^{3+2}+2*3^3+2*3^2+2$$

As another example, consider n=16647 and b=2. The resulting expansion is

$$16647=2^{2^{2+1}+2^2+2}+2^{2^{2+1}}+2^2+2+1$$

The rising and falling heights of the number form the number's "skyline."

For each pair of integers n and b in the input, display the complete base-b representation of n. Your display should use multiple output lines for different exponent heights. The display must begin with n=, followed by the expansion. Answers should use an asterisk (*) as the multiplication symbol between coefficients and powers of b. Zero terms must not be printed, and unnecessary coefficients and exponents must not be shown (for example, display 1 instead of b^0, b^2 instead of $1*b^2$,b instead of b^1). To assist in accurately viewing the skyline of the number, the display must show one character (either a digit, +, or *) per column of the multi-line display, and there must be no unnecessary spaces. The correct format is illustrated in the sample output shown below.

Answers must be displayed using no more than 80 columns. Expansions requiring more than 80 columns must be split between terms, and a second set of display lines used to show the

remaining portion of the expansion. The second part of the answer must begin in the same column as the previous part of the answer. See the sample output for an example.

Input

input is a sequence of pairs of integers, n and b, followed by a pair of zeros. Each value for n will be positive, and each value for b will be greater than or equal to 2. No value will exceed the maximum signed size for the machine.

Output

For each input pair, n and b, print the complete base-b expansion of n as described above. Print a line containing

n in complete base b:

preceding each expansive. Separate the output for consecutive pairs by a line of hyphens. All coefficients, bases, and exponents are to be displayed as standard base 10 integers. The expansion for each input pair will require at most two standard screen widths, allowing for indentation and splitting between terms of the expansion.

Input sample

```
20000 3
16647 2
1000 12
85026244 3
0 0
```

Output sample

```
20000 in complete base 3:
          2
        3     3+2     3       2
20000=3   +3      +2*3   +2*3   +2
------------------------------------------------------------

16447 in complete base 2:
          2+1   2          2+1
        2    +2   +2    2         2
16647=2               +2       +2   +2+1
------------------------------------------------------------

1000 in complete base 12:
            2
1000=6*12   +11*12+4
```

85026244 in complete base 3:

$$85026244=3^{3^2}+2*3^3+2*3^2+2*3^3+2*3^2+2*3^3+2*3^2+2*3^3+2*3+1$$

$$3^{2*3+2}+2*3^{2*3+1}+3^{+2*3}+3^{+3}+2*3^{+2*3}+3^{+3+1}$$

【中文译稿】

输入文件名：tower.in

计算机生成图形学中有一个问题，是如何构造“有序的随机物”，例如山脉或是城市 的天空背景轮廓线。一个图形学公司有了一个主意：利用数字表示法中的数字起伏来代表模型的高度。在这个问题中你将计算几个这样的数字表示，然后显示它们代表的“城市天空背景轮廓线”。

设 n 是任意正整数，b 是一个大于等于 2 的整数，n 的以 b 为底的完全展开是用下述方法得到的：先写出通常的 n 的以 b 为底的展开，即一系列 b 的幂与一个属于 1 和 b-1 之间的整系数乘积的总和，省略掉系数为 0 的项，例如，如果 n=20000，b=3，则 n 的以 3 为底的展开如下：

$$20000=3^9+3^5+2*3^3+2*3^2+2$$

要得到以 b 为底的完全展开，我们对指数也进行同样的处理，直到所有的数都以 b 为底表示。对于 n=20000 和 b=3，结果是：

$$20000=3^{3^2}+3^{3+2}+2*3^3+2*3^2+2$$

又例如，考虑 n=16647，b=2，结果是：

$$16647=2^{2^{2+1}+2^2+2}+2^{2^{2+1}}+2^2+2+1$$

其中数学高度的升降构成了它的“地平线”。

对输入中的每对 n，b，显示出 n 的以 b 为底的完全展开。输出从 n=开始，右边根据指数高度用多行显示，然后是其展开，结果中应用*表示在系数和 b 的幂间的乘号，系数为 0 的项目不可打印，不必要的系数和指数不能显示（例如，要用 1 代替 b^0，b^2 代替 $1*b^2$，b 代替 b^1）。为使“地平线”的显示准确，输出中每个字符（可以是数码、+或*）占一列，不应有不必要的空格。输出如范例中所示：输出不能超过 80 列，需要超过 80 列的展开必须在项间分隔开来，并用第二组显示行来显示展开的剩余部分。参照输出范例的格式。

输入

输入是一个由整数对 n、b 构成的序列，一行一个整数对，每个 n 值都是正的，每个 b 值都不小于 2，数值的大小不会超过机器所能表示的范围。输入的最后一行为 0 0。

输出

对输入中每对 n 和 b，按上述格式输出 n 的以 b 为底的完全展开，在每个展开前输出一行：n in complete base b:

在各组输出间输出一行连字符，起分隔作用。所有系数、底数、指数都要以 1 0 进制数表示，每组输出最多要求屏幕宽度的 2 倍，包括在项间分开和行首缩进所需的额外空间。

输入范例

```
20000 3
16647 2
1000 12
85026244 3
0 0
```

输出范例

```
20000 in complete base 3:
          2
         3     3+2     3      2
20000=3   +3     +2*3   +2*3 +2
--------------------------------------------------------------------------------
16447 in complete base 2:
          2+1   2          2+1
         2    +2   +2    2        2
16647=2              +2       +2   2+1
--------------------------------------------------------------------------------
1000 in complete base 12:
           2
1000=6*12   +11*12+4
--------------------------------------------------------------------------------
85026244 in complete base 3:
             2                2            2            2            2           2
            3   +2*3+1       3   +2*3     3   +3+2     3   +3+1      3   +2      3 +1
85026244=3              +2*3         +2*3         +2*3         +2*3         +2*3
                  2
                 3          2*3+2    2*3+1    3
               +2*3    +2*3        +3       +2*3   +3+1
```

算 法 分 析

我们将 n 写成以 b 为底的展开式，对展开式中大于 1 的指数也写成以 b 为底的展开式……依次类推，直至展开式中的所有数都是以 b 为底表示的，这个展开式就称为 n 的以 b 为底的完全展开式。

一、展开式的组成方式

首先将 n 转换成 b 的指数形式

$$N=(N_{n-1}\,N_{n-2}\cdots N_0)_b$$
$$=N_{n-1}\times b^{n-1}+N_{n-2}\times b^{n-2}+\cdots+N_1\times b+N_0$$

其中系数 $N_i(0\leqslant i\leqslant n-1)$表示第 i 位的数码，可以是 0～b-1 中的任何一个整数，由具体的数 N 来确定；b^i 为第 i 位的权幂，其中 b 为幂的底数，i 为指数；由系数 N_i，权幂的底数 b 以及 N_i 与 b 间的‘*’和每两项之间的‘+’组成 n 的一个以 b 为底的展开式。组成方式：

1．按由左至右的顺序分析每一项，省略系数为 0 的项；

2．除第一个系数非 0 的项外，其他项的前面加一个‘+’；

3．除 N_0 外其他为 1 的系数（N_i=1，i>0）将被忽略；

4．除 N_0 外，其他项可表示为

$$\begin{cases} n_i\times b & n_i>1 \quad i=n-1 \\ b & n_i=1 \quad i=n-1 \\ +n_i\times b & n_i>1 \quad 0<i<n-1 \\ b & n_i=1 \quad 0<i<n-1 \end{cases}$$

若系数 N_i>0 且指数 i>1，则依据上述办法写出 i 的以 b 为底的一个展开，形成上一行展开式，对这一行展开式中的指数也作同样的处理……。直至当前行中的所有数都是以 b 为底表示的。

二、构造展开式矩阵

由完全展开式的输出格式可以看出，各个项是顺序输出的。只有当前面一个项、包括它指数部分的展开式输出完毕，才开始输出当前项。输出中每一个字符（数码、‘+’或‘*’）占一列，不应有不必要的空格。

例如 n=16647， b=2， 结果是

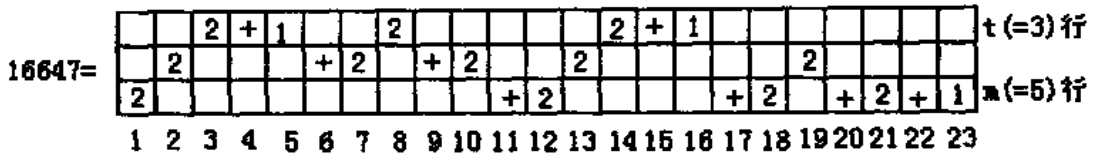


图 1.8-1

显然，上述完全展开式构成一个矩阵，其中最底一行为系数、底数组成的项以及连接的运算符（‘*’或‘+’），由此往上的各行分别列出各项指数以 b 为底的展开过程，每一列仅一个字符。根据展开式的定义和组成方式，我们可以通过下列算法构造完全展开式矩阵 map：

其中 m 为 map 中最底行的行号。由于完全表达式的行数一般不会超过 5，因此设 m=5；

t 为 map 中最顶行的行号。初始时 t=m;每生成展开式的一行，t←t-1。最后的 m-t+1 即为指数高度；

d 为展开式的列宽。由于 map 中的每一列仅允许一个字符，因此每往 map 中填一个字符，d←d+1。初始时 d=0；

map 矩阵的构造由过程 solve(p，n)完成。其中值参 p 为当前行号；n 为待展开的十进制数。

```
Procedure solve (p，n);
 begin
   t←min (p，t)
   将 n 转换成（Nk…N0）b;
   for i:=k downto 0 do
    if Ni>0 then
     begin
       if Ni不是第 1 个非零系数 then begin d←d+1; map[p，d]←'+';end; {then}
       if (Ni>1)or(i=0) then begin d←d+1; map[p，d]←'Ni'; end; {then}
       if i>0 then
           begin
           if Ni>1 then begin d←d+1; map[p，d]←'*'; end; {then}
           d←d+1; map[p，d]←'b';
           if i>1 then solve (p-1，i);
           end; {then}
      end; {then}
 end; {solve}
```

显然，初始时 map 矩阵清零。然后调用 solve(5，n)便可构造出 n 的以 b 为底的完全展开式矩阵 map。

三、输出完全展开式

由于打印的顺序由上而下、由左及右，因此我们总是从 map 中的第 t 行开始逐行打印

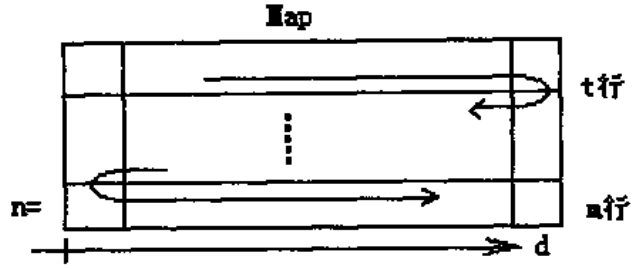


图 1.8-2

在打印 map 中的 $t\cdots m-1$ 行元素时，必须将光标移至 length('*n*=')位置后才能开始打印该行，而打印 map 中的第 m 行元素前必须先打印 '*n*=' 。问题是行宽总是有限的。行宽的最大值设为 80 列，即展开式的一组显示行充其量为（80-length('*n*=')）个字符宽。当 d 大于这个宽度上限时，必须采取分段打印的办法。而分段又不能“一刀切”，随意将一个完整的项拦腰截断，因此必须将当前一组显示行中最后一个可能被截断的项和展开式的剩余部分移至下一组显示行打印。

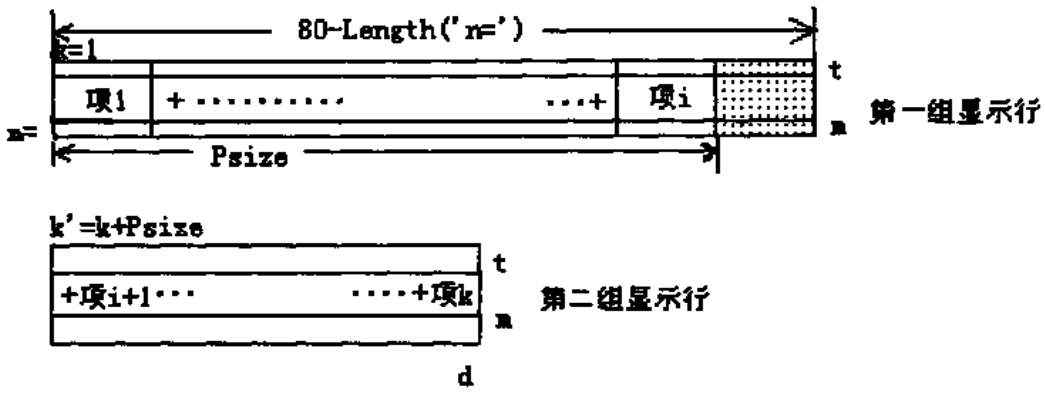


图 1.8-3

设 k 为当前一组显示行的首字符在 map 中的列位置。初始时 k=0。如果（k+(80-length('*n*='))）≤d 时，说明当前一组显示行还不足以打印完全展开式的剩余部分。最后一组显示行的长度为 $d-k$。

psize 为当前一组显示行的实际宽度。计算方法为

```
j←80-length ('n=');
while(k+j+1≤d)and(map[m，k+j+1]<>'+') do j←j-1;
psize←min{j，d-k}
```

若 $d-k>j$，则说明当前一组显示行非最后一组显示行，其长度为 j；若 $d-k\leqslant j$，则说明当前一组显示行为最后一组显示行，其长度为 $d-k$。

每打印完一组显示行后，$k\leftarrow k+psize$，即下一组显示行从 map[t，k+psize]字符开始打印。

在打印当前一组显示行的 map[i，$k+j$]字符时，若发现 map[i，$k+j$]··map[i，$psize+k$]间

全为空格，则说明 i 行已打印完毕，应从 map[i+1，k+1]字符开始打印 i+1 行：

```
p←k+j;
while (p≤psize+k)and(map[i，p]='')do p←k+1;
if p>psize+k then 转向打印 i+1 行；
```

由此得出算法：

```
置 'n=' 未打印标志； k←0;
repeat
  计算当前一组显示行的实际宽度 psize;
  for i:=t to m do
   begin
    for j:=1 to psize do
     begin
      if map[i，k+j]‥map[i，k+psize]全为 '' then break;
      if j=1 then begin
          if (i=m)and('n='未打印)
                then begin 置 'n=' 打印标志；打印 'n=' ; end; {then}
                else 光标移至 length('n=')列；
                   end; {then}
      打印 map[i，k+j];
     end; {for}
    换行；
   end;  {for}
 k←k+psize;
until k=d;
```

程 序 题 解

```
program tower (input ,  output);
const  inputname          = 'tower.in';  { 输入文件名串 }
       outputname         = 'tower.out'; { 输出文件名串 }
       maxt               = 5;           { 完全展开式矩阵的底行序号 }
       maxp               = 1000;        { 完全展开式矩阵列宽的上限 }
       maxsize            = 100;

type   tmap               = array[1 .. maxt ,  1 .. maxp] of char;
       tlst               = array[0 .. maxsize] of longint;
var    infp ,  outfp      : text;        { 输入文件变量和输出文件变量 }
       n ,  b ,  m        : longint;     { 待转换的十进制数、底数、编号 }
       t ,  p             : longint;     { 完全展开式矩阵的顶行序号，列宽 }
       map                : tmap;        { 完全展开式矩阵 }

procedure printline (prc : char; prtimes :integer);{ 向文件输出prtimes个prc字符}
var    i ,  j             : integer;
```

```
begin
  for i :=1 to prtimes do
    write (outfp ,  prc);
  writeln (outfp);
end; {printline}

procedure addmap (addi : integer; instr : string);
{ 在map的第addi行添入字串instr，计算列宽p }
var     i ,  j                  : integer;
        inlen                   : byte absolute instr;      { instr的串长 }
begin
  for i :=1 to inlen do
    begin
      inc (p);
      map[addi ,  p] :=instr[i];
    end;  {for}
  if addi < t then t :=addi;
end; {addmap}

procedure solve (nowi : integer; nown : longint);
{ 从map的第nowi行出发，构造nown的以b为底的完全展开式 }
var     blst                : tlst;          { b进制数序列 }
        bn                  : integer;       { blst中的数码个数 }
        i ,  j ,  k         : integer;
        nstr                : string;

procedure getblst (inn : longint; var relst : tlst; var ren : integer);
{ 将十进制数inn转换成b进制数序列relst，其数码个数为ren }
var     i ,  j ,  k             : integer;
begin
  fillchar (relst ,  sizeof(relst) ,  0);
  ren :=-1;
  while inn > 0 do
    begin
      inc (ren);
      relst[ren] :=inn mod b;
      inn :=inn div b;
    end; {while}
end; {getblst}
```

```
begin
  if nowi<t then t:=nowi;  { 计算map的顶行序号 }
  getblst (nown ,  blst ,  bn);
 (将十进制数nown转换成b进制数序列blst，其数码个数为bn}
  k :=0;
  for i :=bn downto 0 do    { 由高位开始顺序搜索每一位b进制数码 }
    if blst[i] > 0 then      { 若第i位数码大于0 }
    begin
      inc (k);
      if k > 1                    { 若第i位非最高位，则map的nowi行添一个‘+’ }
        then addmap (nowi ,  '+');
      str (blst[i] ,  nstr);
      if (blst[i] > 1) or (i = 0)
      { 若i位为最低位或者该位数码大于1，则数码添入map的nowi行 }
        then addmap (nowi ,  nstr);
      str (b ,  nstr);
      if i > 0                     { 若当前位非最低位 }
        then begin
          if (blst[i] > 1)  { 若当前位数码>1，则‘*’添入map的nowi行 }
              then addmap (nowi ,  '*');
              addmap (nowi ,  nstr);{  ‘b’添入map的nowi行 }
               if i > 1
      { 若位数(权幂的指数)大于1，则递归上一行，将指数i转换成以b为底的完全展开式 }
                  then begin
                         solve (nowi - 1 ,  i);
                       end;  {then}
              end; {then}
    end;  {then}
end; {solve}

procedure printmap;  { 输出n的以b为底的完全展开式 }
var    i ,  j ,  k ,  l    : integer;
       headp              : boolean; {  ‘n=’打印标志 }
       size ,  psize      : integer;{一组显示行的最大列宽，当前组显示行的实际列宽}
       nstr               : string;  { ‘n=’串 }
       testp              : integer; {辅助变量 }

function min(com1 ,  com2 : integer) : integer;{计算和返回整数com1和com2中的较小者 }
```

```
begin
  if com1 < com2
    then min :=com1
    else min :=com2;
end; {min}

begin
  headp :=false;
  str (n ,  nstr);
  nstr :=nstr + ' = ';          { 存贮 'n=' 串 }
  size :=80 - length(nstr);  { 计算一组显示行的最大列宽 }
  k :=0;
  repeat
    j :=size;                        { 计算当前一组显示行的实际列宽 }
    while (k + j + 1 <= p) and (map[maxt ,  k + j + 1] <> '+') do
      dec (j);
    psize :=min (j ,  p - k);
    { 完全展开式矩阵map中第t··maxt行、第k+1··k+psize列为当前一组显示行 }
    for i :=t to maxt do       { 顺序输出当前一组显示行的t行··maxt行 }
      begin
      for j :=1 to psize do  { 顺序输出当前一组显示行中第i行的每一个字符 }
        begin
        testp :=k + j;{判断i行的第j个字符后是否全为空格，若是，则准备输出下一行}
        while (testp <= psize + k) and (map[i ,  testp] = ' ') do inc (testp);
        if testp > psize + k then break;
        if j = 1                              { 若准备输出第i行的第1个字符 }
          then begin
            if (i = maxt) and (not headp){若位于底行且 'n=' 未输出，则先输出 'n=' }
              then begin
                   headp :=true;
                   write (outfp ,  nstr);
                   end   {then}
              else begin                      {否则右移leagth('n=')个字符位置w }
                   for l :=1 to length(nstr) do write (outfp ,  ' ');
                   end;  {else}
                end;  {then}
         write (outfp ,  map[i ,  k + j]);{ 输出当前一组显示行中i行的第j个字符 }
        end;  {for}
      writeln (outfp);
```

```
        end;   {for}
      k :=k + psize;                          { 计算下一组显示行的起始列位置 }
    until k = p;                              { 直至完全展开式输出完毕 }
end; {printmap}

begin
  m :=0;                                      { 编号初始化 }
  assign (infp ,  inputname); reset (infp);
  { 输入文件名串与文件变量连接，输入文件读准备 }
  assign (outfp ,  outputname); rewrite (outfp);
 { 输出文件名串与文件变量连接，输出文件写准备 }
  repeat
    readln (infp ,  n ,  b);                  { 读入待转换的十进制数和底数 }
    if b > 0
      then begin
             fillchar (map , sizeof(map) ,  #32);{ 完全展开式矩阵初始化 }
             t :=maxt; p :=0;                 { 矩阵的顶行序号和列宽初始化 }
             inc (m);                         { 累计编号 }
             if m > 1 then printline ('-' ,  50);{每两个输出结果间用 '……' 分隔}
             writeln (outfp ,  n ,  ' in complete base ' ,  b ,  ':');
             solve (maxt ,  n);  { 从map的底行出发构造n的以b为底的完全展开式 }
             printmap;                        { 输出完全展开式 }
           end;   {then}
  until (n = 0) and (b = 0);                  { 直至文件输入完毕 }
  close (outfp);                              { 关闭输入文件和输出文件 }
  close (infp);
end. {main}
```

第二章 '97 国际大学生计算机程序设计竞赛试题解析

§2.1 系 统 依 赖

试 题

【英文原稿】

Input file: depend.in

Components of computer systems often have dependencies----other components that must be installed before they will function properly. These dependencies are frequently shared by multiple components. For example, both the TELNET client program and the FTP client program require that the TCP/IP networking software be installed before they can operate. If you install TCP/IP and the TELENT client program, and later decide to add the FTP client program, you do not need to reinstall TCP/IP.

For some components it would not be a problem if the components on which they depended were reinstalled; it would just waste some resources. But for others, like TCP/IP, some component configuration may be destroyed if the component was reinstalled.

It is useful to be able to remove components that are no longer needed. When this is done, components that only support the removed components may also be removed, freeing up disk space, memory, and other resources. But a supporting component, not explicitly installed, may be removed only if all components which depend on it are also removed. For example, removing the FTP client program and TCP/IP would mean the TELENT client program, which was not removed, would no longer operate. Likewise, removing TCP/IP by itself would cause the failure of both the TELNET and the FTP client programs. Also if we installed TCP/IP to support our own development, then installed the TELNET client (which depends on TCP/IP) and then still later Removed the TELENT client, we would not want TCP/IP to be Removed.

We want a program to automate the process of adding and removing components. To do this we will maintain a record of installed components and component dependencies. A component can be installed explicitly in response to a command (unless it is already installed), or implicitly if it is needed for some other component being installed. Likewise, a component, not explicitly installed, can be explicitly Removed in response to a command (if it is not needed to support other components) or implicitly removed if it is no longer needed to support another component.

Input

The input will contain a sequence of commands (as described below), each on a separate line containing no more than eighty characters. Item names are case sensitive, and each is no longer than ten characters. The command names (DEPEND, INSTALL, REMOVE and LIST) always appear in uppercase starting in column one, and item names are separated from the command name and each other by one or more spaces. All appropriate DEPEND commands will appear before the occurrence of any INSTALL command that uses them. There will be no circular dependencies. The end of the input is marked by a line containing only the word END.

Command Parameters	Semantics/Response
DEPEND item1 item2 [item3 ....]	Item1depends Item2 (andItem3)
INSTALL item1	Install item1 and components which it depends
REMOVE item1	Remove item1,if possible, Remove all which it depends
LIST	List all components in current system

Output

Echo each line of input. Follow each echoed INSTALL or REMOVE line with the actions taken in response, making certain that the actions are given in the proper order. Also identify exceptional conditions (see *Expected Output*, below, for examples of all cases). For the LIST command, display the names of the currently installed components. No output, except the echo, is produced for a DEPEND command or the line containing END. There will be at most one dependency list per item.

Input sample

```
DEPEND TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER TCPIP HTML
INSTALL NETCARD
```

```
INSTALL TELNET
INSTALL foo
REMOVE NETCARD
INSTALL BROWSER
INSTALL DNS
LIST
REMOVE TELNET
REMOVE NETCARD
REMOVE DNS
REMOVE NETCARD
INSTALL NETCARD
REMOVE TCPIP
REMOVE BROWSER
REMOVE TCPIP
LIST
END
```

Output sample

```
DEPEND TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER TCPIP HTML
INSTALL NETCARD
  Installing NETCARD
INSTALL TELNET
  Installing TCPIP
  Installing TELNET
INSTALL foo
  Installing foo
REMOVE NETCARD
  NETCARD is still needed.
INSTALL BROWSER
  Installing HTML
  Installing BROWSER
INSTALL DNS
  Installing DNS
LIST
  HTML
```

```
  BROWSER
  DNS
  NETCARD
  Foo
  TCPIP
  TELNET
REMOVE TELNET
  Removing TELNET
REMOVE NETCARD
  NETCARD is still needed.
REMOVE DNS
  Removing DNS
REMOVE NETCARD
  NETCARD is still needed.
INSTALL NETCARD
  NETCARD is already installed.
REMOVE TCPIP
  TCPIP is still needed.
REMOVE BROWSER
  Removing BROWSER
  Removing HTML
  Removing TCPIP
REMOVE TCPIP
  TCPIP is not installed.
LIST
  NETCARD
  foo
END
```

【中文译稿】

输入文件名：depend.in

计算机系统中安装的组件通常都需要一些预先安装的其他组件的支持，才能正常工作。这些被依赖的组件一般来说将被多个组件共享。例如，若想使用 TELNET 和 FTP 的客户终端程序，那就需要预先安装 TCP/IP 网络协议软件。如果你首先安装了 TCP/IP 和 TELNET 客户终端程序，那么以后如果你要安装 FTP 客户终端程序的话，就不需要再安装 TCP/IP 了。对于一些组件来说，当他们依赖的组件被重装后，将不会发生什么问题，仅仅丢失一些资源而已。但是另一些组件，例如 TCP/IP，如果被重装后，一些组件的配置将被破坏。

如果一个组件在不需要的时候能够从系统中将它卸载，这是一个非常有用的功能。而且在卸载时，那些仅仅支持这个被卸载组件的那些组件也应当被卸载，并释放相应的磁盘空间、内存以及其他资源。但是，对于一个并非被显式地安装的支持组件，我们只有在所有依赖它的组件都被卸载后，才能将其从系统中卸载。例如卸载了客户终端程序 FTP 和网络协议软件 TCP/IP，将导致尚未被卸载的客户终端程序 TELNET 无法使用。同样地，卸载 TCP/IP 将导致 TELNET 和 FTP 无法使用。并且，如果我们安装网络协议 TCP/IP 来支持自己开发软件的话，那么安装 TELNET 客户终端程序之后再卸载 TELNET 客户终端程序时，我们也不希望将 TCP/IP 卸载。

我们希望编制一个可以自动处理增加/卸载组件的程序。而要实现这个目的，我们首先要维护一个已被安装的组件之间相互依赖关系的记录。一个组件可以被一个命令显式地安装(除非它已经被安装了)或者被隐式安装，如果它被其他组件所需要的话。同样地，一个如果并非被显式安装的组件，可以用一个命令将其显式地卸载(如果不需要支持其他组件的话)，或者被隐式删除，如果它不再需要支持其他组件。

输入

输入将包含一系列的命令(见下表)，每行不超过 80 个字符。组件的名字有大小写区别，并且不超过 10 个字符。而命令(DEPEND,INSTALL,REMOVE 和 LIST)总是在每行的开始且为大写字符。每个组件的名字之间以及组件名与命令之间将由一个或多个空格隔开。每个适当的 DEPEND 命令将在任何需要使用他们的 INSTALL 命令出现之前出现。这里将没有循环依赖。输入的结尾将以一个 END 命令结束。

命令参数	语义/应答
DEPEND item1 item2 [item3 ....]	Item1 依赖 Item2 (和 Item3)
INSTALL item1	安装 item1 及其所依赖的组件
REMOVE item1	卸载 item1,如果可能的话，也卸载其所依赖的组件
LIST	列出当前安装的所有组件

输出

每一行输入将被按原样输出。对于每一个 INSTALL 或 REMOVE 命令将在其相应输出后添加一个表明其执行动作的输出。并且应表示出所有异常情况(可参考下面列出的例子中表明的所期望的输出)。对于 LIST 命令，应显示当前已经安装的各个组件。对于 DEPEND 和 END 命令，除了其命令本身，将没有额外的输出。对于每个组件将最多只有一个依赖表。

输入范例

```
DEPEND TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
```

```
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER TCPIP HTML
INSTALL NETCARD
INSTALL TELNET
INSTALL foo
REMOVE NETCARD
INSTALL BROWSER
INSTALL DNS
LIST
REMOVE TELNET
REMOVE NETCARD
REMOVE DNS
REMOVE NETCARD
INSTALL NETCARD
REMOVE TCPIP
REMOVE BROWSER
REMOVE TCPIP
LIST
END
```

输出范例

```
DEPEND TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER TCPIP HTML
INSTALL NETCARD
  Installing NETCARD
INSTALL TELNET
  Installing TCPIP
  Installing TELNET
INSTALL foo
  Installing foo
REMOVE NETCARD
  NETCARD is still needed.
INSTALL BROWSER
  Installing HTML
  Installing BROWSER
INSTALL DNS
```

```
  Installing DNS
LIST
  HTML
  BROWSER
  DNS
  NETCARD
  Foo
  TCPIP
  TELNET
REMOVE TELNET
  Removing TELNET
REMOVE NETCARD
  NETCARD is still needed.
REMOVE DNS
  Removing DNS
REMOVE NETCARD
  NETCARD is still needed.
INSTALL NETCARD
  NETCARD is already installed.
REMOVE TCPIP
  TCPIP is still needed.
REMOVE BROWSER
  Removing BROWSEk
  Removing HTML
  Removing TCPIP
REMOVE TCPIP
  TCPIP is not installed.
LIST
  NETCARD
  foo
END
```

算 法 分 析

一、Depend 命令的处理

在安装/卸载组件前必须明确系统中所含的组件以及它们之间的互相依赖关系。这件事由 Depend 命令解决:

Depend__item1__item2…item*p*

item1 依赖 item2、…、item*p*,或者说 item2、…、item*p* 共享 item1。

系统的所有组件包含在 Depend 命令组中。一条 Depend 命令为一个字串 Line。从命令字后的第 1 个空格开始依次列出各个组件名。组件名与组件名之间由空格隔开。不同的 Depend 命令之间可以含有相同的组件名。我们在读入 Depend 命令组的同时建立一个组件序列表 items，其中 items[*i*]为第 *i* 个组件名，序列的表尾指针为 item。每读一条 Depend 命令串,先取出‘item1’并在串中裁去‘Depend__item1__’,使得剩余子串变为‘item2__……__item*p*’；然后再取出‘item2’,并裁去‘item2__’,……依次类推,直至取出‘item *p*’后剩余子串空为止。每取一个组件名后搜索 items 表。若该组件名在表中存在，则返回其序号；否则将组件接在表尾,表尾指针 item 即为该组件序号。为此我们设计了两个函数:

```
function Getnextitem (var Line):string[10];
  begin
    p←Line 中第 1 个 '_' 前的位置;
    if  p<0  then p←Line 长度;
    Getnextitem←由 Line 中第 1…第 p 个字符组成的命令字或组件名;
    删去 Line 中的第 1…第 p 个字符;
    删去 Line 串首的无用空格;
  end;{Getnextitem}
function GetitemNum(组件名串 s):byte;
  begin
    计算 items 表中不同于 s 的组件名个数 p;
    if p>item then begin item←item+1;items[item]←s;end;{then}
    GetitemNum←p;
  end; {GetitemNum}
```

显然,读入当前命令串 Line 后,由

s←Getnextitem (Line);

取得命令字,而后

i←GeitemNum(Getnxtitem(Line));

取得命令字后第 1 个组件 item1 在 items 序列中的序号。

Depend 命令字后的组件名可能不止一个。为了描述该组件被哪些组件所共享,我们设计了一个布尔矩阵:

$$\text{Depend}[i,j]=\begin{cases}\text{true} & \text{items 表中第 } i \text{ 个组件被第 } j \text{ 个组件共享}\\ \text{false} & \text{items 表中第 } i \text{ 个组件未被第 } j \text{ 个组件共享}\end{cases}$$

$$(1\leqslant i,j\leqslant \text{item})$$

显然通过

```
s←Get nextitem (Line);
i←Getitem Num(Getnxtitem(Line));
while Line<>''do Depend [i,GetitemNum(Getnextitem(Line))]←true;
```

可以确定当前 Depend 命令中 item1 组件被哪些组件所共享,即 item1 与其他组件间的依赖关系。

例如执行下列 Depend 命令组

```
Depend  telent  tcpip  netcard
Depend  tcpip   netcard
Depend  dns  tcpip  netcard
Depend  browser  tcpip  html
```

建立的 items 序列为

items[1]= 'telent'　item[2]='tcpip'　item[3]= 'netcard'

items[4]= 'dns'　item[5]= 'browser'　item[6]= 'html'

系统依赖关系矩阵 Depend 为

Depend[1,2]=Depend[1,3]=Depend[2,3]=Depend[4,2]=

Depend[4,3]=Depend[5,2]=Depend[5,6]=true

其他元素值为 false。所有组件的 Install 值为 0(所有组件未安装)。

我们可以通过一张系统图来描述 Depend 命令组执行后的结果。

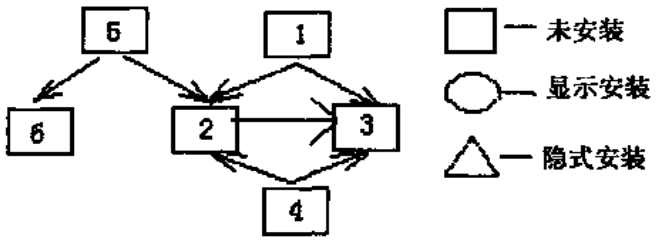


图 2.1-1

图中的顶点为组件,有向边为依赖关系，'组件 i →组件 j' 表示组件 j 共享组件 i,组件 i 支持组件 j。

二、Install 命令的处理

Depend 命令组之后是 Install 命令组,在明确组件间依赖关系的基础上进行系统安装。Install 命令的格式是:

```
Install  item1
```

命令字后仅一个组件。显然读入命令串 Line 后，通过

s←Getnextitem(Line)

取得命令字 'Install'，而后

i←GetitemNum(Getnextitem(Line))

取得 item1 在 items 序列中的序号,即安装 itens 表中的第 i 个组件(简称组件 i)及其他所依赖的所有组件。

安装分两种形式:

1. 显式安装

如果组件 i 未被安装的话,则该组件被 Install 命令显式地安装。

2. 隐式安装

如果组件 i 被组件 j 所需要(Depend[i,j]=true)而组件 j 又未被安装,则组件 j 被 Install 命令隐式地安装;同样,另一个未被安装的组件 k 需要组件 j(Depend[j,k]=true),则组件 k 也被 Install 命令隐式地安装,这一传递关系是递归的。

由此可见,一个组件分“未安装、显式安装、隐式安装”三种情况。设

$$\text{Install}[i]=\begin{cases}0 & \text{items 序列中的第 } i \text{ 个组件未被安装}\\ 1 & \text{items 序列中的第 } i \text{ 个组件被隐式安装}\\ 2 & \text{items 序列中的第 } i \text{ 个组件被显式安装}\end{cases}$$

我们设计一个安装过程 installitem(k,b)。其中 k 为被安装组件在 items 序列中的序号;b 为显式安装标志,如果组件 k 是被依赖组件的话,b=false。

```
procedure   installitem(k,b)
 begin
   for i:=1 to item do
     if (Depend[k,i]) and (Install[i]=0)   then installitem (i,false);
     打印 items 序列中的第 k 个组件正在安装;
     if b then install[k]←2
         else install[k]←1;
 end; {installitem}
```

显然,读入一条 Install 命令并取得被安装组件在 items 序列中的序号之后,通过执行下列程序:

```
if Install [i]<>0 then 输出 items[i]组件不能被安装
                  else installitem(i,true);
```

便可安装该组件及其所有依赖的组件。

例如对图 2.1-1 中的系统图执行下列 Install 命令组:

```
Install   net card
Install   telnet
Install   foo
Install   browser
Install   dns
```

由于所增一个组件 foo,使得 items[7]=‘foo’。组件的安装矩阵 Install 为

Install[1]=Install[3]=Install[4]=Install[5]=Install[7]=2

Install[2]=Install[6]=1

对应的系统图如下:

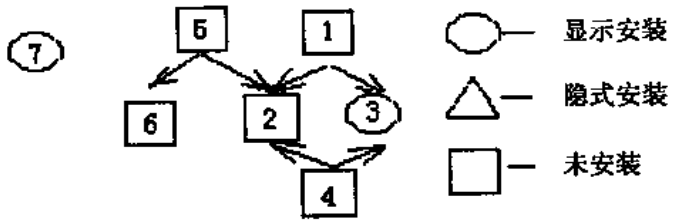


图 2.1-2

三、Remove 命令的处理

系统中安装的组件可以通过 Remove 命令卸载。Remove 命令的格式是

Remove item1

命令字后仅一个组件,显然读入命令串 Line 后通过

s←Getnextitem(Line);

取得命令字‘Remove’,而后由

i←GetitemNum(Getnextitem(Line)),

取得 item1 在 items 序列中的序号 i (简称组件 i),即卸载组件 i 及其他所依赖的组件。卸载的前提是待卸载的对象处于安装状态并且系统现安装的组件中未有支持它的组件,换句话讲,如果有一个已安装的组件 i 待卸载(Install[i]<>0),而 items 序列中存在一个满足下列条件的组件 j

(Depend[j,i]=true)and(install[j]<>0)　　($1 \leqslant j \leqslant$item)

则组件 i 无法从系统中卸载下来。

如果组件 i 处于安装状态并且系统现安装的组件中未有支持它的组件,则该组件被 Remove 命令显式地卸载;同样,如果被组件 i 所依赖的隐式安装的组件 (满足(Depend[i,p]=true) and (Install[p]=1)的所有组件 p ($1 \leqslant p \leqslant$item)) 中存在满足上述条件的组件,则该组件也被 Remove 命令隐式地卸载,这个传递关系显然是递归的。

我们设计一个卸载过程 Removeitem(k,b),其中 k 为待卸载组件在 items 序列中的序号,b 为显式卸载标志。如果 k 是一个被依赖的隐式安装的组件,则 b=false。

```
procedure  Removeitem(k,b);
 begin
计算 items 序列中是否有支持 k 的安装组件 i (满足(Depend[i,k]=true)and(Install[i]<>0));
  if items 序列中没有这样的组件
     then begin
          打印 item[k]正在卸载;
          Install[k]←0;
          end {then}
     else if b then 打印 items[k]不能卸载信息;
   for i:=1 to item do
```

```
    if (Depend[k,i]=true) and (Install[i]=1) then Removeitem (i,false);
  end; {Removeitem}
```

显然,在取得待卸载组件在 items 序列中的序号 i 后,通过执行下列程序

```
  if Install[i]=0 then 输出 items[i]组件不能卸载;
                  else Removeitem(i,true);
```

便可卸载该组件及其他所依赖的组件(如果可能)。

例如对图 2.1-2 的系统图执行下列 Remove 命令组:

```
Remove  netcard
Remove  telnet
Remove  netcard
Remove  dns
Remove  tcpip
Remove  browser
```

使得 Install[3]=Install[7]=2, Install[1]=Install[2]=Install[4]=Install[5]=Install[6]=0 对应的系统图为:

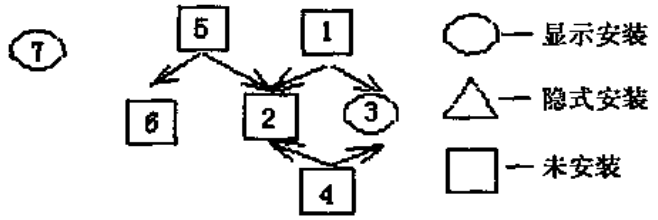


图 2.1-3

四、List 命令的处理

List 命令的格式为: List

命令字后无任何组件。该命令将列出 items 序列中 Install 值<>0 的所有组件。即

```
for i:=1 to item do
  if Install[i]<>0 then 输出 items[i]组件;
```

例如对图 2.1-3 执行 List 命令后,显示系统当前安装的组件为:

```
net card
foo
```

五、算法流程

有了上述基础,不难列出算法的大致流程:

```
repeat
  读当前命令行 Line;
```

```
s←Getnextitem(Line);  {取当前命令字}
if s[1]∉ { 'L'  'E' }{若命令非'List'和'end',则取第 1 个组件}
  then i←GetitemNum(Getnextitem(Line));
    case  s[1]  of
      'D' :  处理 Depend 命令;
      'T' :  处理 Install 命令;
      'R' :  处理 Remove 命令;
      'L' :  处理 List 命令;
      'E:' :  break;
    end; {case}
Until  false;
```

程 序 分 析

```
program Depend;
const
    Max_item     = 100;    { 最多组件数 }

type
    TItem     = string[10];   { 组件名 }

var
    Line : string;              { 当前命令行 }
    Item : byte;                { 当前组件数 }
    Items     : array [1..MAX_ITEM] of TItem; { 组件名序列 }
    Depnd     : array [1..MAX_ITEM, 1..MAX_ITEM] of boolean;
                { 组件的依赖矩阵。Depnd[i,j]=true 表示组件 i 依赖组件 j }
    Install   : array [1..MAX_ITEM] of byte;
                {
                                          0  组件 i 未安装
                存贮安装信息 install{i}=  1  组件 i 隐式安装
                                          2  组件 i 显式安装
                }
    i     : byte; { 辅助变量 }
    s     : TItem;

function GetNextItem(var Line: string): TItem;
{返回 Line 串中第 1 个空格前的子串 GetNextItem 和截去该子串后的剩余子串 line }
```

```
var
    p    : integer;
begin
    p := pos(' ', Line) - 1;              { 计算第 1 个空格位置 }
    if p < 0 then                         { 若 line 为最后一个组件名,则指向串尾 }
        p := length(Line);
    GetNextItem := copy(Line, 1, p); { 返回第 1 个空格前的子串}
    delete(Line, 1, p);                   { 截去该子串 }
    while (length(Line) > 0) and (Line[1] = ' ') do   { 删去串首的无用空格 }
        delete(Line, 1, 1);
end;   {GetNextitem}

function GetItemNum(ItemName: TItem): byte;
{ 返回以 ItemName 为名的组件在 Items 数组的下标 }
var
    p    : byte;
begin
    p := 1;              { 计算 itemName 组件在组件序列的下标 }
    while (p <= Item) and (Items[p] <> ItemName) do
        inc(p);
    if p > Item then   { 若该组件是一个新组件,则接在组件序列尾 }
    begin
        inc(Item);
        Items[Item] := ItemName;
    end;   {then}
    GetItemNum := p;
end;   {GetItemNum}

procedure InstallItem(ItemNum: byte; ClearInstall: boolean);
{安装组件 ItemNum 及其所依赖的未安装组件,其中 ClearInstall 为显式安装标志 }
var
    i    : byte;
begin
    for i := 1 to Item do   {搜索每一个组件,对所有依赖 I
                            temNum 且未安装的组件进行隐式安装}
        if Depnd[ItemNum, i] and (Install[i] = 0) then InstallItem(i, false);
    writeln('    Installing ', Items[ItemNum]);
    if ClearInstall {若 itemNum 组件被显式安装,则设置该组件显式安装标志}
      then Install[ItemNum] := 2
```

```
        else Install[ItemNum] := 1;{否则设置该组件隐式安装标志 }
end; {InstallItem}

procedure RemoveItem(ItemNum: byte; ShowMsg: boolean);
{卸载组件 ItemNum 及其所依赖的已不再需要的隐式安装组件,其中 ShowMsg 为提示错误
信息标志 }
 var
    i     : byte;
begin
  i := 1;          { 检查在所有支持 itemNum 的组件中是否有安装组件 }
  while (i <= Item) and not (Depnd[i, ItemNum] and (Install[i] <> 0)) do inc(i);
    if i > Item { 若无,则组件 ItemNum 可被卸载 }
    then begin
         writeln('     Removing ', Items[ItemNum]);
         Install[ItemNum] := 0;
         end{then}
    else       { 若有,则提示错误信息 }
         if ShowMsg then writeln('     ', Items[ItemNum], ' is still needed.');
    { 卸载其所依赖的隐式安装的组件 }
    for i := 1 to Item do
         if Depnd[ItemNum, i] and (Install[i] = 1) then RemoveItem(i, false);
end; {RemoveItem}

begin
    assign(input, 'depend.in');      { 输入文件名串与文件变量连接 }
    reset(input);                    { 输入文件读准备 }
    assign(output, 'depend.out');    { 输出文件名串与文件变量连接 }
    rewrite(output);                 { 输出文件写准备 }
    fillchar(Depnd, sizeof(Depnd), false); { 变量初始化 }
    fillchar(Install, sizeof(Install), 0);
    repeat
         readln(Line);               { 读取下一个命令行 }
         writeln(Line);
         s := GetNextItem(Line);     { 取命令字 }
         if not (s[1] in ['L', 'E'])      {若非'list'和'end'命令,则计算命令字后第 1 个组件序号}
              then i := GetItemNum(GetNextItem(Line));
         case s[1] of
              'D': { 确定组件 1 与其他组件的依赖关系 }
                   while Line <> '' do Depnd[i, GetItemNum(GetNextItem(Line))] := true;
```

```
            'I':  { 安装组件 i }
                  if Install[i] <> 0 then writeln('     ', Items[i], ' is already installed.')
                                     else InstallItem(i, true);
            'R':  { 卸载组件 i }
                  if Install[i] = 0 then writeln('     ', Items[i], ' is not installed.')
                                    else RemoveItem(i, true);
            'L':  { 显示已安装的组件 }
                  for i := 1 to Item do
                      if Install[i] <> 0 then writeln('     ', Items[i]);
            'E':  { 处理 END 命令 }
                  break;
        end;  {case}
    until false;
    close(input); { 关闭输入输出文件 }
    close(output);
end.  {main}
```

§2.2 吉尔的又一个骑车问题

试　题

【英文原稿】

Input file： jill.in

Jill likes to ride her bicycle, but since the pretty city of Greenhills where she lives has grown, Jill often uses the excellent public bus system for part of her journey. She has a folding bicycle which she carries with her when she uses the bus for the first part of her trip. When the bus reaches some pleasant part of the city, Jill gets off and rides her bicycle. She follows the bus route until she reaches her destination or she comes to a part of the city she does not like. In the latter event she will board the bus to finish her trip.

Through years of experience, Jill has rated each road on an integer scale of "niceness". Positive niceness values indicate roads Jill likes; negative values are used for roads she does not like. Jill plans where to leave the bus and start bicycling, as well as where to stop bicycling and re-join the bus, so that the sum of niceness values of the roads she bicycles on is maximized. This means that she will sometimes cycle along a road she does not like, provided that it joins up two

other parts of her journey involving roads she likes enough to compensate. It may be that no part of the route is suitable for cycling so that Jill takes the bus for its entire route. Conversely, it may be that the whole route is so nice Jill will not use the bus at all.

Since there are many different bus routes, each with several stops at which Jill could leave or enter the bus, she feels that a computer program could help her identify the best part to cycle for each bus route.

Input

The input file contains information on several bus routes. The first line of the file is a single integer b representing the number of route descriptions in the file. The identifier for each route (r) is the sequence number within the data file, $1 \leqslant r \leqslant b$. Each route description begins with the number of stops on the route: an integer s, $2 \leqslant s \leqslant 20{,}000$ on a line by itself. The number of stops is followed by $s-1$ lines, each line i ($1 \leqslant i < s$) is an integer n_i representing Jill's assessment of the niceness of the road between the two stops i and $i+1$.

Output

For each route r in the input file, your program should identify the beginning bus stop i and the ending bus stop j that identify the segment of the route which yields the maximal sum of niceness, $m = n_i + n_{i+1} + \ldots + n_{j-1}$. If more than one segment is maximally nice, choose the one with the longest cycle ride (largest $j-i$). To break ties in longest maximal segments, choose the segment that begins with the earliest stop(lowest i). For each route r in the input file, print a line in the form:

The nicest part of route r is between stops i and j.

However, if the maximal sum is not positive, your program should print:

Route r has no nice parts.

Input sample

```
3
3
  -1
   6
10
   4
  -5
   4
  -3
   4
   4
```

```
-4
 4
-5
4
-2
-3
-4
```

Output sample

```
The nicest part of route 1 is between stops 2 and 3
The nicest part of route 2 is between stops 3 and 9
Route 3 has no nice parts
```

【中文译稿】

输入文件名：jill.in

吉尔很喜欢骑自行车，但自从她所居住的城市 Greenhill 发展起来之后，吉尔一般都利用完善的公共汽车系统完成她旅途的一部分。她会带一辆折叠式的自行车去乘车。当公共汽车开到城市的一些美丽地方的时候，吉尔就会下车并开始骑她的自行车。吉尔会沿着公共汽车的线路行进，直到到达她的目的地，或者到达一个她不喜欢的地方，而这时候她会再一次乘上公共汽车来完成她余下的路程。

经过多年的经历，吉尔将每段路的“美好程度”量化为一个整型数。正整数意味着吉尔喜欢这条路，而负整数则意味她不喜欢。吉尔想要作这样一项计划：在何处下公共汽车来骑自行车，而在何处又回到公共汽车上，能够使得她用自行车骑过的路程的“美好程度”之和达到最大值。这意味着她有时也会在一段她不喜欢的路上骑车，因为这段路连接了两段她足够喜欢的路段。也许在线路上没有一个地方是吉尔喜欢的，那她只能整条线路都乘车。相反，也许线路上所有地方吉尔都喜欢，那么吉尔就不需要乘车了。

这里有许多不同的公共汽车线路，每条线路都有数个车站，吉尔可以在这些车站上下车。她希望能有一个计算机程序来帮她决定对于每条线路她应该在哪些路段上骑车最好。

输入

输入文件包含了数条公共汽车线路的信息。文件的第一行是一个整型数 b，表明文件中有几条公共汽车线路的信息。而在数据文件中每条线路的标识(r)则是一个序列数，$1 \leqslant r \leqslant b$。每条线路的信息由一个表明线路上有几个车站的整型数 $s(2 \leqslant s \leqslant 20{,}000)$开始。在这个数字后面有 s-1 行，这 s-1 行表明在两个站点之间的“美丽程度”，其中第 i 行中的数字 n_i的含义是在车站 i 与 i+1 之间的“美丽程度”。

输出

对于输入文件中的每条路线 r，程序应该给出骑车路段的起始车站 i 和终止车站 j，以

及相关的“美好程度”之和的最大值 m，$m=n_i+n_{i+1}+\ldots+n_{j-1}$。如果达到这个最大值有多种方案，那么挑选一种使骑车路线最长（使$(j-i)$最大）的方案。若这样还是有多个方案，那么再挑选 i 最小的那个方案。对每条路线 r，都应按以下方式打印一行：

The nicest part of route r is between stops i and j.

并且，如果这个最大值不是正数，那么你的程序应该打印：

Route r has no nice parts.

输入范例

```
3
3
  -1
   6
10
   4
  -5
   4
  -3
   4
   4
  -4
   4
  -5
4
  -2
  -3
  -4
```

输出范例

The nicest part of route 1 is between stops 2 and 3

The nicest part of route 2 is between stops 3 and 9

Route 3 has no nice parts

算 法 分 析

对于一条线路来说,吉尔选择“美好程度”之和最大的一段路程骑车。当然,如果这个“美好程度”之和的最大值是一个负数的话,表明吉尔不喜欢这条线路,她将放弃骑车的想法。但是要注意的是,如果“美好程度”之和的最大值为一个正数并且对应了多条可选择的骑车路线的话,吉尔首先选择经过站点数最多的一个方案。若这样还是有多个方案,那么她

再选择骑车出发站距线路首站最近的一个方案。

帮助吉尔选择骑车路程的算法并不复杂。如果不考虑优化,则可以通过双重循环,枚举所有可能的路线(外循环枚举起点 i,内循环枚举终点 j),从中选择一个最佳方案:

```
设   n 为当前线路的站点数;
     len 为骑车路线中的线段数  (1≤len≤n-1);
     start、over 为骑车路线的起讫点;
     max 为骑阵路线的“美好程度”之和;
      max←-1;    start←n;    len←1;
        for i:=1 to n-1 do
         for j:=i+1 to n do
           begin
             if ((i 站至 j 站的“美好程度”之和)≥max)
               then begin
                 if (i 站至 j 站的“美好程度”之和)>max
                   then begin
                         start←i;  over←j;
                         max←i 站至 j 站的“美好程度”之和;
                         len←j-i;
                         end   {then}
                    else if (j-i)≥len then begin len  ←j-i; start←i; over←j; end {then}
                         end   {then}
           end;   {for}
       if   max<0   then   输出该条线路没有美好的路段
                    else   输出 start 站和 over 站之间是最美好的一段路线;
```

上述算法的时间复杂度 $W(n)=n^2$。站数 n 每增加一个 k,程序的执行时间将以 k 平方的速率增加,n 值愈大,算法的时效愈低。更何况,试题要求 n 在 2…20000 之间,对于这么大的问题规模,该算法显然不堪承受。下面,我们给出一个优化的算法:

从站 1 出发逐站地延伸线路,直至 $n-1$ 条线段分析完为止。当分析至第 i 条线段时($1\leqslant i\leqslant n-1$),设

sum:站 1 至站 $i+1$ 条线段的美好程度和;

w:延伸过程中 sum 的最小值,即 $w=\min_{2\leqslant k\leqslant i+1}$ {站 1 至站 k 的美丽程度和}。若确定站 1 至站 p ($2\leqslant p\leqslant i+1$)的美好程度和为 w,sum>$w$,则在站 1 和站 $i+1$ 之间站 p 至站 $i+1$ 路线的美好程度和是最大的,其值为 sum-w。

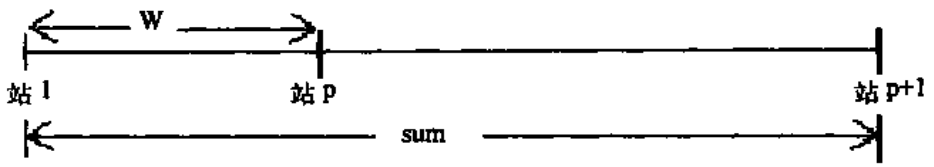


图 2.2-1

设 max 为站 1 至站 2、站 1 至站 3、…站 1 至站 i 的 i-1 条线路中美好程度之和的最大值,即

max= $\max_{2\le k\le i}$ {站 1 至站 k 美好程度和的最大值}

如果出现下述两种情况之一

① sum-w>max

②(sum-w=max) and (over-start<i+1-p)

则记下目前最佳方案

max←sum-w; start←p; over←i+1;

如果 sum<w,则说明 sum 成为延伸过程中得出的美好程度之和的最小值(w←sum)。由于站 1 至站 i+1 间美好程度之和的最大值(sum-w)为负值,因此吉尔将放弃在该路线上骑车的计划,选择另一条路线的起点应设为 i+1 站(p←i+1)。

由此得出算法:

```
max←-1;  sum←0;  w←0;  p←1;
  for i:=1  to n-1 do
    begin
     计算前 i 条线段的美好程度和 sum;
     if ((sum-w)>max) or ((sum-w)=max) and (over-start<i+1-p)
        then begin
           max←sum-w;  start←p;  over←i+1;
                end;  {then}
     if sum<w  then begin w←sum;  p←i+1; end;  {then}
    end; {for}
  if  max<0  then 输出该条线路没有美好的路段
              else 输出 start 站和 over 站之间是最美好的一段路线;
```

上述算法的时间复杂度为 w(n)=n-1,其时间效率比第一种算法有显著提高。

程 序 分 析

```
program Jill;
 var
   Route    : word;   { 线路号 }
```

```
    Stop : word;          { 车站数 }
    Nice : longint;       { 当前路段的美好程度和 }
    Start,
    Over: word;           { 当前最佳方案的起讫站 }
    Nicest    : longint;  { 当前最佳方案的美好程度和 }
    i, o  : word;         { 当前方案的起讫站 }
    Sum,                  { 站 1 至当前站的美好程度和 }
    Worst     : longint;  { min { 站 1 至站 k 的美好程度和 }}
                          2≤k≤当前站

begin
    assign(input, 'jill.in');      { 输入文件名串与文件变量连接 }
    reset(input);                  { 输入文件读准备 }
    assign(output, 'jill.out');    { 输出文件名串与文件变量连接 }
    rewrite(output);               { 输出文件写准备 }
    readln(Route);                 { 读线路数 }
    for Route := 1 to Route do { 分析每一条线路 }
     begin
       Nicest := -1;               { 最佳方案的美好程度和初始化 }
       Sum := 0;                   { 当前方案的美好程度和初始化为 0 }
       Worst := 0;
       o := 1;                     { 当前方案由站 1 开始 }
       readln(Stop);               { 读当前线路的站数 }
       for i := 1 to Stop - 1 do   { 分析每一个路段 }
        begin
        readln(Nice);              { 读当前路段的美好程度,累计线路的美好程度和 }
        inc(Sum, Nice);
        if (Sum - Worst > Nicest) or ((Sum - Worst = Nicest) and (Over - Start < i + 1 - o))
        {若当前方案的美好程度和最大,或者虽然与最佳方案相等但经过的路段数最多,
        则当前方案作为最佳方案记下  }
            then
              begin
               Nicest := Sum - Worst;
               Start := o;
               Over := i + 1;
              end;  {then}
        if Sum < Worst then
  { 若当前方案的美好程度和为负值，则废弃当前方案，从下一路段开始重新计算当前
方案 }
```

```
          begin
            Worst := Sum;   o := i + 1;
          end;   {then}
        end;   {for}
        if Nicest < 0 then   { 若美好程度和的最大值为负,则输出没有美好路段 }
            writeln('Route ', Route, ' has no nice parts.')
        else   { 否则选择由 start 站至 over 站的路段骑车 }
            writeln('The nicest part of route ', Route, ' is between stops ', Start, ' and ', Over);
    end;   {for}
    close(input);   { 关闭输入文件和输出文件 }
    close(output);
end.   {main}
```

§2.3 莫尔斯编码

试　题

【英文原稿】

Input file: morse.in

Samuel F. B. Morse is best known for the coding scheme that carries his name. Morse code is still used in international radio communication. The coding of text using Morse code is straightforward. Each character (case is insignificant) is translated to a predefined sequence of *dits* and *dahs* (the elements of Morse code). Dits are represented as periods(".") and dahs are represented as hyphens or minus sighs ("-"). Each element is transmitted by sending a signal for some period of time. A dit is rather short, and a dah is, in perfectly formed code, three times as long as a dit. A short silent space appears between elements, with a longer space between characters. A still longer space separates words. This dependence on the spacing and timing of elements means that Morse code operators sometimes do not send perfect code. This results in difficulties for the receiving operator, but frequently the message can be decoded depending on the context.

In this problem we consider reception of words in Morse code without spacing between letters. Without the spacing, it is possible for multiple words to be coded the same. For example, if the message "dit dit dit" were received, it could be interpreted as "EEE", "EI", "IE" or "S" based on the coding scheme shown in the sample input. To decide between these multiple interpretations, we assume a particular context by expecting each received word to appear in a

dictionary.

For this problem your program will read a table giving the encoding of letters and digits into Morse code, a list of expected words (*context*), and a sequence of words encoded in Morse code (*morse*). These *morse* words may be flawed. For each *morse* word, your program is to determine the matching word from *context*, if any. If multiple words from *context* match *morse*, or if no word matches perfectly, your program will display the best matching word and a mismatch indicator.

If a single word from *context* matches *morse* perfectly, it will be displayed on a single line, by itself. If multiple *context* words match *morse* perfectly, then select the matching word with the fewest characters. If this still results in an ambiguous match, any of these matches may be displayed. If multiple *context* words exist for a given *morse*, the matching word will be displayed followed by an exclamation point ("!").

We assume only a simple case of errors in transmission in which elements may be either truncated from the end of a *morse* word or added to the end of a *morse* word. When no perfect matches for *morse* are found, display the word from *context* that matches the longest prefix of morse, or has the fewest extra elements beyond those in *morse*. If multiple words in context match using these rules, any of these matches may be displayed. Words that do not match perfectly are displayed with a question mark ("?") suffixed.

The input data will only contain cases that fall within the preceding rules.

Input

The Morse code table will appear first and consists of lines each containing an uppercase letter or a digit C, zero or more blanks, and a sequence of no more than six periods and hyphens giving the Morse code for C. Blanks may precede or follow the items on the line. A line containing a single asterisk("*"), possibly preceded or followed by blanks, terminates the Morse code table. You may assume that there will be Morse code given for every character that appears in the *context* section.

The *context* section appears next, with one word per line, possibly preceded and followed by blanks. Each word in context will contain no more than ten characters. No characters other than upper case letters and digits will appear. There will be at most 100 *context* words. A line containing only a single asterisk ("*"), possibly preceded or followed by blanks, terminates the *context* section.

The remainder of the input contains morse words separated by blanks or end-of-line characters. A line containing only a single asterisk("*"), possibly preceded or followed by blanks, terminates the input. No *morse* word will have more than eighty (80) elements.

Output

For each input *morse* word, display the appropriate matching word from *context* followed by

an exclamation mark ("!") or question mark ("?") if appropriate. Each word is to appear on a separate line starting in column one.

Input sample

```
A .-
B -...
C -.-.
D -..
E .
F ..-.
G --.
H ....
I ..
J .---
K -.-
L .-..
M --
N -.
O ---
P .--.
Q --.-
R .-.
S ...
T -
U ..-
V ...-
W .--
X -..-
Y -.--
Z --..
0 -----
1.----
2..---
3...--
4....-
5.....
6-....
7--...
```

8—––..
9————.
*
AN
EARTHQUAKE
EAT
GOD
MATH
IM
READY
TO
WHAT
WROTH
*
.—.....— .....—....
—.————.. .—.—.————..
.—.....— .—.
..—.—.—....—.—..—.—.—.
..— .—...—..—.—
——— ..—
*

Output sample

WHAT
HATH
GOD
WROTH?
WHAT
AN
EARTHQUAKE
IM!
READY
TO
IM?

【中文译稿】

输入文件名：morse.in

Samuel F.B. Morse 由于以他名字命名的编码方案而闻名。莫尔斯编码现在仍被用在国

际无线电报通讯中。使用莫尔斯编码对文本进行编码是非常简单明确的。每个字符(大小写不敏感)被翻译成一序列预先定义好的短音和长音(莫尔斯编码的基本元素)。短音用点(".")表示，而长音则用连字号或减号("-")表示。这两个元素的传输是通过传送一个维持不同时间长短的信号来实现的。短音是一个短信号，而长音则是一个为短信号三倍长的信号，在元素之间用一个短间隔标识，相对较长的传输间隔表示字符的间隔，而一个更长的传输间隔则表示单词的间隔。莫尔斯编码的传输元素基于时间长短的特性意味着莫尔斯编码发报人员有时候会发送一些不太标准的编码。这就使得接收人员有时候必须根据上下文来解码。

在本题中我们考虑这样一个情况，我们使用莫尔斯编码编码，但在字符之间将不加入间隔。而没有这样的间隔，不同的单词的编码就有可能相同。例如，如果发来的讯息是"短、短、短"，则根据由输入例子定义的编码模式，可以被解释成"EEE"，"EI"，"IE"或者"S"。为了从这些解释中挑选一个最可能的单词，我们假设此时的上下文条件为每个所期望接收到的单词应该在字典中出现。

本题中，你的程序将依序读入字符莫尔斯编码方案，所期望接受的单词列表(*context*)，和一个用莫尔斯编码编码的单词序列(*morse*)。在 *morse* 中单词是有一些问题的，对于每个这样的单词，你的程序需要决定与 *context* 中的哪个单词匹配。如果在 *context* 中有多个单词与之匹配，则应挑选一个最佳匹配，若没有一个单词匹配，则应给出一个未匹配的标识。

如果只有一个匹配到的单词，则将它输出在一行中。如果有多个匹配，则挑选一个最短的单词，如果仍有多个，则在一行中显示其中任意的一个,并在其输出后加一个叹号("!")。

我们假设在传输中只可能发生一种简单的错误，即在单词编码的末尾可能被截去或添加了一些元素(短音或长音)。当在 *morse* 中有无法匹配单词时，则匹配该 *morse* 单词尽可能长的前缀，或匹配在末尾添加了尽可能少的元素的该 *morse* 单词。如果按此原则有多个匹配到的单词，则在一行中显示其中任意的一个。这些非完美匹配的单词后面要加上一个问号("?")作为后缀。输入数据将仅仅包含前面这些情况。

输入

莫尔斯编码表将首先出现在输入文件中，这个表的格式为每行包含一个大写英文字符或数字 C 和一个不超过 6 个莫尔斯元素组成的 C 的莫尔斯编码，两者之间用多个空格隔开。在行首或行末可能会有一些空格。然后文件中将有一个由星号("*")组成的行，该行同样在行首或行末可能会有一些空格。这一行用于结束莫尔斯编码表。你可以假设在下面的 *context* 中出现的所有字符在莫尔斯编码表中都已经提供其莫尔斯编码。

然后是 *context* 部分，每行一个单词，在行首或行末可能会有一些空格。除了大写字符和数字将不会出现其他字符。*context* 中的每个单词的长度不超过 10 个字符，最多可以有 100 个单词。同样由一个星号("*")组成的行用于结束 *context*，该行在行首或行末可能会有一些空格。

余下部分是 *morse* 中的编码后的单词，这些单词用数个空格或行末符分隔。用一个星号("*")组成的行结束输入，该行在行首或行末可能会有一些空格。所有 *morse* 中单词将不会超过 80 个元素。

输出

对 *morse* 部分中的每个编码后的单词，按照前面所阐述的规则显示从 *context* 中匹配到的最合适的单词，有必要的话加上叹号("!")或问号("?")。每个单词应在一个独立行的行首显示。

输入范例

A .-
B -...
C -.-.
D -..
E .
F ..-.
G --.
H ....
I ..
J .---
K -.-
L .-..
M --
N -.
O ---
P .--.
Q --.-
R .-.
S ...
T -
U ..-
V ...-
W .--
X -..-
Y -.--
Z --..
0 -----
1.----
2..---
3...--
4....-

5.....
6-....
7--...
8---..
9----.
*
AN
EARTHQUAKE
EAT
GOD
MATH
IM
READY
TO
WHAT
WROTH
*
.--.....-- .....--....
--.----.. .--.-.----..
.--.....-- .--.
..-.-.-....--.-..-.--.-.
..-- .-...--..-.--
---- ..--
*

输出范例

WHAT
HATH
GOD
WROTH?
WHAT
AN
EARTHQUAKE
IM!
READY
TO
IM?

算 法 分 析

一、输入的同时建立几张表

1．建立字符的 morse 码表 codes

设 codes[*c*]为字符 *c* 的 morse 码。一个字符的 morse 码信息由一行字串 Line 组成：‘_字符 c_morse 码序列_’。字符和对应的 morse 码序列间可用多个空格隔开,而行首行尾也可能含一些空格,这些空格都是无用的。因此,可以通过

```
while Line[1]= ‘’ do delete(Line,1,1);
while Line[2]= ‘’ do delete(Line,2,1);
while Line[length[line]]= ‘’ do delete(line,length(line),1);
```

将之删除,使得 line 变为‘字符 c morse 码序列’。最后通过

```
codes[line[1]]←copy(line,2,255);
```

将字符 c 的 morse 码存入 codes[字符 c]。

在输入了每个 morse 码序列后,开始输入单词表。同样,输入每个单词的同时必须建立单词的 morse 码表：

2．建立单词表(contents)和单词的 morse 码表(morses)

设 *n* 为单词数；contents[*i*]为第 *i* 个单词串。由于输入单词串时行首行尾可能含一些无用空格,因此必须通过上述方法将之删除后存入 contents 表；morses[*i*]为第 *i* 个单词的 morse 码串($1\leqslant i\leqslant n$)。我们将单词中的每一个字符转换成对应的 morse 码后存入,即

```
morses[i]←’’;
for j:=1 to 单词长度 do morses[i]←morses[i]+codes[单词的第 j 个字符];
```

二、匹配计算

设当前待匹配单词的 morse 码序列为 *m* 。那么对于 morses 表中的每一个单词来说能与 *m* 匹配到的最长前缀是多少呢?

设 list[*j*]为 morses 表中第 *j* 个单词能与 *m* 匹配到的最长前缀的长度

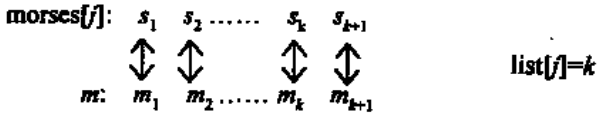


图 2.3-1

可通过下述算法计算 list 表:

```
for i:=1 to m 的长度 do
 for j:=1 to n do if(list[j]=i-1)and (morses[j][i]=mi) then list[j]←list[j]+1;
```

显然 list 表中所有满足((list[*j*]=*m* 的长度) and (list[*j*]=morses[*j*]的长度))的单词 *j* 都为匹配到的单词。如果这样的单词有多个,则挑选其中一个最短的单词。设这个匹配到的单词序号为 *i*:

i←0;

for j:=1 to n do

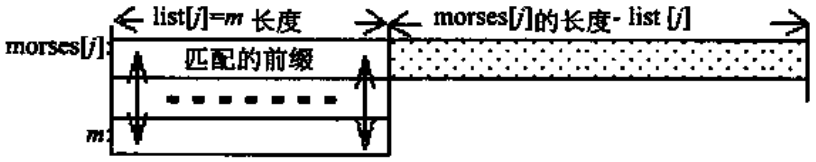
if (list[j]=m 长度) and (list[j]=morses[j]的长度) and ((i=0) or(单词 j 的长度<单词 i 的长度))

then i←j;

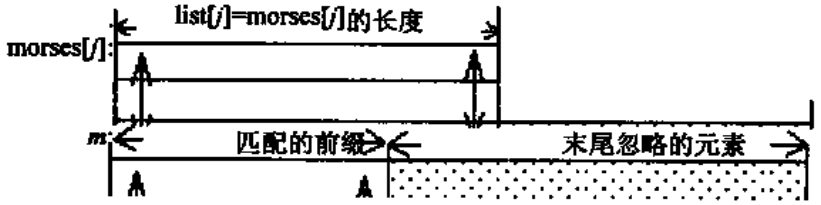
经上述运算后有 *i*=0、*i*<>0 两种结果,由此引出对这两种结果的分析:

① 找出匹配到最长前缀(或末尾忽略元素最少)的单词。

若 *i*=0,则说明单词序列表 contents 中未有与 *m* 完全匹配的单词。此时应在表中寻找一个能够匹配最长前缀的单词。前缀有两种情况:



(1)



(2)

图 2.3-2

由图 2.3-2 可见,*m* 与某单词匹配时,*m* 或者该单词末尾被忽略的元素数愈少,则匹配到的前缀愈长。我们将每个单词的 List 值与 *m* 的长度和单词的 morse 码长度比较,计算末尾被忽略的元素数,从中找出忽略后缀最短的一个单词 *i*, 该单词即为一个能与 *m* 匹配到最长前缀的单词:

设 di 为被忽略后缀的最小长度; dj 为当前被忽略的后缀长度:

```
di←∞
for j:=1 to n do
  begin
    if list [j]=m 长度  then dj←morses[j]的长度-List[j]
                        else if morses[j]的长度=List[j]
                                then dj←m 的长度-List[j]
```

```
                                    else dj←∞;
                if  dj<di  then begin di←dj; i←j; end;{then}
              end;  {for}
```

输出未完全匹配的单词 contents[i],该单词后加后缀‘?’。

② 找出匹配到的单词

若 $i<>0$，则说明在单词表中找到了一个与 m 完全匹配且字符数最小的单词 i，问题是单词表中是否还有这样的单词。显然,如果单词表中的其他单词都不能与 m 完全匹配,则单词 i 为唯一可完全匹配到的单词;否则单词 i 为多个匹配中的一个,输出时应加后缀‘!’：

设 j 为单词表中未与 m 完全匹配的单词个数+1：

```
j←1;
while(j≤n)and not((list[j]=m 长度)and(list[j]=morses[j]的长度)and(j<>i))do j←j+1;
if j>n  then 输出单词 i
       else  输出单词 i 和后缀‘!’;
```

程 序 分 析

```
program Morse;

const
    MaxContent  = 100;                    { 最多单词数 }

var
    Codes       : array ['0'..'Z'] of string[6];{ 莫尔斯编码表。codes[c]为字符 c 的 morse 码 }
    Content     : byte;                   { 单词数 }
    Contents  : array [1..MaxContent] of string[10];{ 单词序列表。Contents[i]第 i 个单词串 }
    Morses      : array [1..MaxContent] of string[60];
                  {单词的 morses 码表,morses[i]为单词 i 的 morse 码串 }
    Lens, Chars  : array [1..MaxContent] of byte;
    { Lens[i]=length(Morses[i]),  Chars[i]=length(Contents[i]) }

procedure Open(name: string);                 { 打开输入输出文件 }
begin
    assign(input, name + '.in');              { 输入文件名串与文件变量连接 }
    assign(output, name + '.out');            { 输出文件名串与文件变量连接 }
    reset(input);                             { 输入文件读准备 }
    rewrite(output);                          { 输出文件写准备 }
end;  {Open}

procedure Load;        { 输入字符的 morse 表,单词表,建立单词的 morse 编码表 }
```

```
var
    i    : byte;          { 辅助变量 }
    line : string;        { 当前输入行 }
begin
    readln(line);                                  { 读一行莫尔斯编码信息 }
    while pos('*', line) = 0 do                    { 若莫尔斯编码表未读完 }
    begin
        while line[1] = ' ' do delete(line, 1, 1);   { 删去行首的无用空格 }
        while line[2] = ' ' do delete(line, 2, 1);   { 删去字符与莫尔斯编码间的无用空格 }
        while line[length(line)] = ' ' do delete(line, length(line), 1); { 删去行尾的无用空格 }
        Codes[line[1]] := copy(line, 2, 255);    { 字符的莫尔斯编码存入 codes 编码表 }
        readln(line);                              {读下一行信息}
    end; {while}
    Content := 0;                                  { 单词数初始化 }
    readln(line);                                  { 读一个单词 }
    while pos('*', line) = 0 do                    { 若单词序列表未读完 }
    begin
        while line[1] = ' ' do delete(line, 1, 1);   { 删去串首、串尾的无用空格 }
        while line[length(line)] = ' ' do delete(line, length(line), 1);
        inc(Content);                              { 累计单词数 }
        Contents[Content] := line;                 { 单词存入序列表 contents }
        Morses[Content] := '';
        for i := 1 to length(line) do              { 对单词进行 morse 编码后存入 morses 表}
            Morses[Content] := Morses[Content] + Codes[line[i]];
        Chars[Content] := length(line);
        Lens[Content] := length(Morses[Content]);
        readln(line);
    end;  {for}
end;  {load}

procedure Process;                                 { 对输入单词的 morse 编码进行匹配 }
var
    c          : char;                             { 当前 morse 字符或分隔符 }
    m          : string;                           { 当前待匹配单词的 morse 码序列 }
 list: array[1..MaxContent]of byte;{list[j]为 morses 表中第 j 个单词能与 m 匹配到最长前缀的
长度}
    i, j, di, dj : byte;                           { 辅助变量 }
begin
  m := '';
```

```
repeat
  read(c);              { 读入下一个 morse 字符 }
  if c in ['.', '-']
    then m := m + c  { 求待匹配单词的 morse 码序列 m }
    else begin
      if m <> '' then
      begin          { 计算 list 表 }
       fillchar(list, sizeof(list), 0);
       for i := 1 to length(m) do
         for j := 1 to Content do
           if (list[j] = i - 1) and (Morses[j][i] = m[i]) then inc(list[j]);
{计算 morses 表中能与 m 完全匹配到的单词 i。若这样的单词有多个,则选择其中最短的一个词}
       i := 0;
       for j := 1 to Content do
         if (list[j] = length(m)) and (list[j] = Lens[j]) and ((i = 0) or (Chars[j] < Chars[i]))
           then i := j;
       if i = 0 then
{若 morses 表中未有与 m 完全匹配的单词,则找出匹配到最长前缀或末尾忽略元素最少的单词}
         begin
          di := $FF;                    { 被忽略后缀的最小长度初始化 }
          for j := 1 to Content do     { 分析每一个单词 }
           begin
            if list[j] = length(m)      { 计算第 j 个单词被忽略的后缀长度 }
              then dj := Lens[j] - list[j]
              else  if Lens[j] = list[j] then dj := length(m) - list[j]
                                         else dj := $FF;
            if dj < di then             { 若被忽略的后缀长度目前最短,则记下 }
              begin i := j; di := dj; end;   {then}
           end;   {for}
          writeln(Contents[i], '?');   { 显示能匹配到最长前缀的单词 i }
         end   {then}
          else begin
            j := 1;          {计算单词表中除单词 i 外不能与 m 完全匹配的单词个数}
            while (j <= Content) and not ((list[j] = length(m))
                  and (list[j] = Lens[j]) and (j <> i)) do   inc(j);
            if j > Content    {若单词 i 外的其他单词都不能与 m 完全匹配,则输出单词}
              then writeln(Contents[i])
```

```
                    else writeln(Contents[i], '!');  { 否则输出单词 i 为多个匹配中的一个 }
                  end;  {else}
              end;  {then}
            m := '';
          end; {else}
    until c = '*';{直至 morse 表输入完毕}
end;  {process}

begin
    Open('morse');      { 打开文件 }
    Load;               { 输入字符的 morse 表、单词表,建立单词的 morse 编码表}
    Process;            { 匹配处理 }
    close(input);       { 关闭输入文件和输出文件 }
    close(output);
end.  {main}
```

§2.4 RAID 技术

试　题

【英文原稿】

Input file： raid

RAID (Redundant Array of Inexpensive Disks) is a technique which uses multiple disks to store data. By storing the data on more than one disk, RAID is more fault tolerant than storing data on a single disk. If there is a problem with one of the disks, the system can still recover the original data provided that the remaining disks do not have corresponding problems.

One approach to RAID breaks data into blocks and stores these blocks on all but one of the disks. The remaining disk is used to store the parity information for the data blocks. This scheme uses *vertical parity* in which bits in a given position in data blocks are exclusive ORed to form the corresponding parity bit. The parity block moves between the disks, starting at the first disk, and moving to the next one in order. For instance, if there were five disks and 28 data blocks were stored on them, they would be arranged as follows:

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
	Data Block1	Data Block2	Data Block3	DataBlock4
DataBlock5		DataBlock6	DataBlock7	DataBlock8
DataBlock9	DataBlock10		DataBlock11	DataBlock12
DataBlock13	DataBlock14	DataBlock15		DataBlock16
DataBlock17	DataBlock18	DataBlock19	DataBlock20	
	DataBlock21	DataBlock22	DataBlock23	DataBlock24
DataBlock25		DataBlock26	DataBlock27	DataBlock28

With this arrangement of disks, a block size of two bits and even parity, the hexadecimal sample data 6C7A79EDFC(0110110001111010011110011110110111111100 in binary) would be stored as:

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
	01	10	11	00
01		11	10	10
01	11		10	01
11	10	11		01
11	11	11	00	

If a block becomes unavailable, its information can still be retrieved using the information on the other disks. For example, if the first bit of the first block of disk 3 becomes unavailable, it can be reconstructed using the corresponding parity and data bits from the other four disks. We know that our sample system uses even parity:

$0 \oplus 0 \oplus ? \oplus 1 \oplus 0 = 0$

So the missing bit must be 1.

An arrangement of disks is invalid if a parity error is detected, or if any data block cannot be reconstructed because two or more disks are unavailable for that block.

Write a program to report errors and recover information from RAID disks.

Input

The input consists of several disk sets.

Each disk set has 3 parts. The first part of the disk set contains three integers on one line: the first integer d, $2 \leqslant d \leqslant 6$,is the number of disks, the second integer s, $1 \leqslant s \leqslant 64$, is the size of each block in bits, and the third integer b, $1 \leqslant b \leqslant 100$, is the total number of data and parity blocks on each disk. The second part of the disk set is a single letter on a line, either "E" signifying even parity or "O" signifying odd parity. The third part of the disk set contains d lines, one for each

disk, each holding $s \times b$ characters representing the bits on the disk, with the most significant bits first. Each bit will be specified as "0" or "1" if it holds valid data, or "x" if that bit is unavailable. The end of input will be a disk set with d=0. There will be no other data for this set which should not be processed.

Output

For each disk set in the input, display the number of the set and whether the set is valid or invalid. If the set is valid, display the recovered data bits in hexadecimal. If necessary, add extra "0" bits at the end of the recovered data so the number of bits is always a multiple of 4. All output shall be appropriately labeled.

Input sample

```
5 2 5
E
0001011111
0110111011
1011011111
1110101100
0010010111
3 2 5
E
0001111111
0111111011
xx11011111
3 5 1
O
11111
11xxx
x1111
0
```

Output sample

```
Disk set 1 is valid, contents are: 6C7A79EDFC
Disk set 2 is invalid.
Disk set 3 is valid, contents are: FFC
```

【中文译稿】

输入文件名：raid

RAID(非昂贵磁盘冗余阵列)是一个使用多个磁盘存贮数据的技术。通过在多个磁盘上存贮数据，RAID 方式比在一个盘上存贮数据的方式表现更为出色。如果这些磁盘中有一个出现问题，系统仍然可以从余下未发生问题的磁盘中恢复原始数据。

RAID 组织数据的一种方式是将一组数据分成若干数据块，分别存贮在所有（而不是一个）磁盘上，额外的一个磁盘则用于存贮原始数据块的校验信息。该模式使用了垂直校验的方法，将各数据块中相应位置上的一序列 bit 作异或运算求出其相应的校验码。校验块按序分散存放在不同的磁盘上，首先是 1 号盘，然后是 2 号盘，依次类推。例如在一个有 5 个磁盘和 28 个数据块的情况下，它们将被组织如下：

磁盘 1	磁盘 2	磁盘 3	磁盘 4	磁盘 5
	数据块 1	数据块 2	数据块 3	数据块 4
数据块 5		数据块 6	数据块 7	数据块 8
数据块 9	数据块 10		数据块 11	数据块 12
数据块 13	数据块 14	数据块 15		数据块 16
数据块 17	数据块 18	数据块 19	数据块 20	
	数据块 21	数据块 22	数据块 23	数据块 24
数据块 25		数据块 26	数据块 27	数据块 28

图 2.4-1

假设每个磁盘块存贮 2 个 bit，并且采取偶校验，那么 16 进制数据 6C7A79EDFC(二进制为 01101100 01111010 01111001 11101101 11111100)将按如下存贮：

磁盘 1	磁盘 2	磁盘 3	磁盘 4	磁盘 5
	01	10	11	00
01		11	10	10
01	11		10	01
11	10	11		01
11	11	11	00	

图 2.4-2

如果有一块出现了问题，其上存贮的信息就可用其他磁盘上的信息来恢复。例如，3 号盘上第一块的第一个 bit 出现错误，它就可以被其存贮在其他 4 个盘上相关的校验块和数据块来恢复。我们知道本例中的系统采取的校验方式为偶校验：

0 ⊕0⊕?⊕1⊕0=0

所以丢失的 bit 应该是 1。

如果有校验错误，或者因为有多个磁盘上的相关数据出错而导致数据块无法重建，我们就说此时磁盘组织为非法。

请写一个程序从 RAID 磁盘中报告错误并恢复信息。

输入

输入文件包含了多个磁盘集。

每个磁盘集有 3 个部分，第一部分为一个有 3 个整型数的数据行：第 1 个 d，$2 \leqslant d \leqslant 6$，是磁盘的数量；第 2 个 s，$1 \leqslant s \leqslant 64$，是每块的 bit 数；第 3 个 b，$1 \leqslant b \leqslant 100$，是每个盘上所有的数据块和校验块数量的总和。第二部分是由一个字符组成的数据行，“E”表明使用偶校验，而“O”表明使用奇校验。第三部分有 d 行，每行上有 $s \times b$ 个表示盘上 bit 的字符。当相应字符为“0”、“1”表明该 bit 数据合法且为 0 或 1，而“x”表明该 bit 有错待确定。输入的结尾用一个以 d=0 的磁盘集结束。这里也不会提供额外无用的数据。

输出

对每个输入的磁盘集，输出磁盘集相应的序号及该磁盘集是否合法。若合法，以 16 进制输出相应的数据，若有必要，应在数据之后补 0 以保证 bit 的数量是 4 的倍数。所有的输出应被适当地表示。

输入范例

```
5 2 5
E
0001011111
0110111011
1011011111
1110101100
0010010111
3 2 5
E
0001111111
0111111011
xx11011111
3 5 1
O
11111
11xxx
x1111
0
```

输出范例

```
Disk set 1 is valid, contents are: 6C7A79EDFC
```

Disk set 2 is invalid.

Disk set 3 is valid, contents are: FFC

算 法 分 析

一、raid 技术中的磁盘组织

raid 技术使用 d+1 张盘存贮数据,其中 d 张盘作为原始盘,存贮原始数据块和校验块的信息,一张盘作校验盘,备份校验块中的校验码。原始盘中有一张出现问题,系统可以通过余下未发生问题的原始盘和校验盘恢复数据。

设 d 为原始盘数;

s 为每个块的 bit 数;b 为原始盘上的块数;

a_{ij} 为第 i 张原始盘上第 j 块的信息($1 \leqslant i \leqslant d$, $1 \leqslant j \leqslant b$),若 $(j-1) \bmod d = i-1$,则 a_{ij} 为第 i 张原始盘上的校验块。

磁盘组织的结构如下:

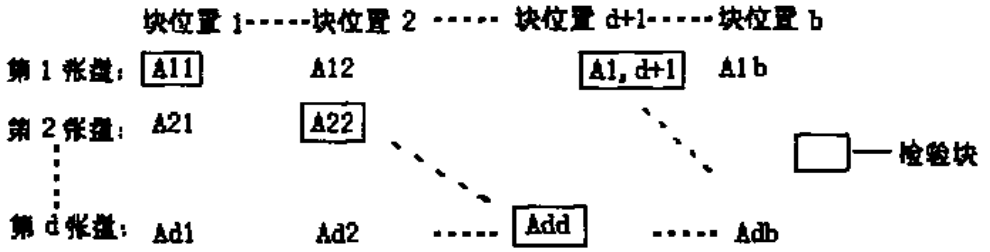


图 2.4-3

为了使垂直校验尽可能多地查出问题,设定磁盘组织的每一列含一个校验块,j 列上的校验块为 $a_{(j-1) \bmod d+1, j}$,校验块的信息为 j 列上 d-1 个数据块垂直异或的结果:

$$a_{(j-1) \bmod d+1, j} = \cdots \oplus a_{(j-1) \bmod d, j} \oplus a_{(j-1) \bmod d+2, j} \cdots \oplus a_{d, j} \qquad (1 \leqslant j \leqslant b)$$

所谓磁盘组织合法,是指在偶校验的情况下:

s 个 '0'

$$a_{1j} \oplus a_{2j} \oplus \cdots \oplus a_{dj} = \{\ '0'\ , \cdots\cdots, '0'\ \}$$

在奇校验的情况下:

s 个 '1'

$$a_{1j} \oplus a_{2j} \oplus \cdots\cdots \oplus a_{dj} = \{\ '1'\ \cdots\cdots, '1'\ \} \qquad \{1 \leqslant j \leqslant b\}$$

并且 d 张盘同块的对应位上不允许出现多个 'x' (未确定的 bit 信息)。因为这些失去

的信息无法通过垂直异或运算恢复。例如有 3 张原始盘，每张盘 1 块，每块 5 个 bit，采用奇校验。

第 1 张盘：	1	1	1	1	1
第 2 张盘：	1	1	X	X	X
第 3 张盘：	X	1	X	1	1

图 2.4-4

上述磁盘组织非法,因为三张盘的第 3 位上出现了两个‘x’。

又如 5 张原始盘,每盘 5 个块,每块 2 个二进制位,采用偶校验。

	第 1 块	第 2 块	第 3 块	第 4 块	第 5 块
第 1 张盘：	00	01	01	11	11
第 2 张盘：	01	10	11	10	11
第 3 张盘：	10	11	01	11	11
第 4 张盘：	11	10	10	11	00
第 5 张盘：	00	10	01	01	11
垂直异式结果：	00	00	00	00	00

□ 校验块

图 2.4-5

显然磁盘组织合法。

如果在一个合法的磁盘组织中,某盘上的 1 块(例如 a_{ij})出现了问题,则可以通过下述运算

$$a_{1j} \oplus a_{2j} \oplus \cdots\cdots \oplus a_{ij} \oplus \cdots\cdots \oplus a_{dj} = \text{第 } j \text{ 块垂直异或的要求} \quad (1 \leqslant i \leqslant d,\ 1 \leqslant j \leqslant b)$$

恢复失去的信息。

例如图 2.4-4 中第 3 张盘的第 3 位改为‘1’,磁盘组织合法。

第 1 张盘：	1	1	1	1	1
第 2 张盘：	1	1	X	X	X
第 3 张盘：	X	1	1	1	1
垂直异式的要求：	1	1	1	1	1

图 2.4-6

显然,在采用奇校验方式的情况下垂直异或的要求为{'1''1''1''1''1'}。由此可得出第 2、第 3 张盘的信息为 {'1''1''1''1''1'} 即

第 1 张盘：	1	1	1	1	1
第 2 张盘：	1	1	1	1	1
第 3 张盘：	1	1	1	1	1
垂直异式的要求：	1	1	1	1	1

图 2.4-7

d 张原始盘的数据块按块号递增顺序组合起来,转换为对应的 16 进制数(若 bit 位不是 4 的倍数,则在数尾添 0 补足),便组成了磁盘组织的数据。例如(图 2.4-5)的磁盘组织数据为 6C7A79EDFC(即$(0110110001111010011110011110110111111100)_2$的十六进制数)。

二、垂直异或运算

由上可知,垂直异或运算是指 d 张盘上同块的对应 bit 位顺序进行位异或。由于输入数据通常采用字串类型,而每个 bit 位的 '0' 或 '1' 通过关系运算后与布尔值 true、false 对应(true、false 的序数值为 0、1),因此可以转换为布尔值的异或运算。即通过

p←false;

for k:=1 to d do p←p xor (a_{ki} 中第 j 个 bit 位= '1');

计算出 d 张盘上 i 块第 j 个 bit 位($1\leqslant i\leqslant b$, $1\leqslant j\leqslant s$)相异或的结果 p,这个 p 由布尔量表示。

如果 p<>(校验方式=奇校验),则说明磁盘组织非法。如果磁盘组织合法,但 a_{ki} 中第 j 个 bit 位为 'x' 。则该位的值确定为 chr(48+ord (p xor (校验方式=奇校验)));

```
for i:=1 to b do
  for j:=1 to s do      {简称 i 块第 j 个 bit 位为对应位置}
   begin
     p←false;
     for k:=1 to d do
       if k 盘的对应位置= 'x'
         then  if 其他盘的对应位置出现过 'x'
                 then  返回磁盘组织非法标志
                 else  记下盘号 k
         else  p←p xor (k 盘的对应位置= '1' );
     if  (d 张盘的对应位置上未出现一个 'x' )and (p<>(校验方式=奇校验))
         then  返回磁盘组织非法标志;
     if  k 盘的对应位置出现 'x'
         then  k 盘对应位置←chr(48+ord (p xor (校验方式=奇校验)));
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

end; {for}
返回磁盘组织合法标志;

三、输出结果

我们顺序将 d 张盘中的第 1 个数据块、d 张盘中的第 2 个数据块、……、d 张盘中的第 b 个数据块组合起来,转换成对应的十六进制数输出。值得提醒的是,对于 a_{ij} 块来说,如果 $(j-1) \bmod d = i-1$,则说明 a_{ij} 为校验块,不应作为磁盘组织中的数据输出。即

```
code= '' ;
for  j:=1 to b do
 for  i:=1  to d do
   if (j-1)mod d <>i-1 then code←code+aij;
```

将二进制数串 code 以四位一组转换成对应的十六进制数输出。尾部不足四位添‘0’;

程 序 分 析

```
program Raid;

const
    MaxDisk=6;              {盘数最大值}
    MaxLen =64;             {块中 bit 位数的最大值}
    MaxBlock     = 100;     {每张盘中块数的最大值}

type                        {块的数据类型}
    TBlock   = string[MaxLen];

var
    DiskSet   : word;       {磁盘集序号}
    Disk,                   {磁盘数}
    Len,                    {每个数据块或校验块的 bit 数}
    Block     : byte;       {每张磁盘的块数}
    Mode      : char;       {奇偶校验标志}
    Data : array [1..MaxDisk, 1..MaxBlock]of TBlock;{data[i,j]为第 i 张磁盘上 j 块的信息}
    i, j, k       : byte;

function Valid: boolean;    {若磁盘组织合法,返回 true;否则返回 false}
var
    i, j, k, l    : byte;
    b     : boolean;
begin
```

● 责任编辑 孙未来 严顺燕
● 封面设计 吴璐丹

Keys and Notes to ACM International Collegiate Programming Contest

定价:25.00元

ISBN 7-309-02141-X
9 787309 021417 >