论文附录

【附录 1】论文中引用的原题

〖问题 3〗千足虫

暗的千足虫（ishongololo）

祖鲁族称千足虫为"ishongololo"，它身长，色黑而亮，是一种多脚的节肢动物．千足虫会吃光它所经过的路上的一切果实．我们以这种事实为基础来理解本题．让我们考虑一个长为 K 宽为 W，高为 H 的各面相互垂直的固体．

请你编程，在题目限定的条件下，使千足虫尽可能多地吃掉小立方块 block．
程序的输出是千足虫吃掉每个 Block 所经路线和动作．千足虫从果实之外开始，吃的第一个 block 必须是(1,1,1)，然后必须再爬到这个 block 上直到无路可走或无 block 可吃时为止。

限定条件：
1．千足虫严格地占据 1 个空的小立方块 block。
2．千足虫每次吃完一个小立方块 block。
3．千足虫不能进入以前自己进入过的小立方块 block。
（此即不能往回走，也不能跨越自己已经走过的路线）．
4．千足虫不能进入未吃过的小立方块 block，也不能爬到果实之外。
5．千足虫只能吃掉或只能爬入相邻的 block，即该 block 与千足虫所在的 block 共面。此外该 block 还必须没有别的面暴露于已被吃光的 block。

输入
你的程序将接收到 3 个整数即长度 L，宽度 W，和
高度 H．L，W，H 是三个整数，每个数占一行，且
它们的取值范围在 1 到 32 之间（含 1 和 32）．

输出
输出数据由若干行组成．每行以"E"（表示吃 Eat)或 M（表示移动 Move）打头，后跟三个整，这三个整数表示千足虫“吃掉”或“移入其内”的小立方块(block)．

评分标准
如果千足虫违反了约束条件，那么你的答案只得零分．
所得总分为吃掉的小立方块的总数和已知最优解之比．
所得最高分不会超过 1OO%

〖问题 4〗01 串

01 串

给定 7 个整数 $N,A_0,B_0,L_0,A_1,B_1,L_1$，要求设计一个 01 串 $S=s_1s_2\ldots s_i\ldots s_N$，满足：

1. $s_i=0$ 或 $s_i=1$，$1<=i<=N$；
2. 对于 S 的任何连续的长度为 L_0 的子串 $s_js_{j+1}\ldots s_{j+L0-1}(1<=j<=N-L_0+1)$，0 的个数大于等于 A_0 且小于等于 B_0；
3. 对于 S 的任何连续的长度为 L_1 的子串 $s_js_{j+1}\ldots s_{j+L1-1}(1<=j<=N-L_1+1)$，1 的个数大于等于 A_1 且小于等于 B_1；

例如，$N=6,A_0=1,B_0=2,L_0=3,A_1=1,B_1=1,L_1=2$，则存在一个满足上述所有条件的 01 串 S=010101。

输入

仅一行，有 7 个整数，依次表示 $N,A_0,B_0,L_0,A_1,B_1,L_1$（$3<=N<=1000$，$1<=A_0<=B_0<=L_0<=N$，$1<=A_1<=B_1<=L_1<=N$），相邻两个整数之间用一个空格分隔。

输出

仅一行，若不存在满足所有条件的 01 串，则输出一个整数-1，否则输出一个满足所有条件的 01 串。

样例输入

6 1 2 3 1 1 2

样例输出

010101

【附录 2】跳棋问题的实现(Tiao.pas)

键盘输入棋盘大小 n，结果输入到屏幕

```
uses crt;
const
   ch        :   array[0..1]of char = ('.','*');
var
   a         :   array[0..100,0..100]of byte; {棋盘}
   n,i,j,k,k1,num :   integer;
procedure Print; {打印棋盘}
 var ii,jj : integer;
 begin
 clrscr;
 inc(num); writeln('No.',num);
 for ii:=0 to n+1 do
  begin
  for jj:=0 to n+1 do
    write(ch[a[ii,jj]],' ');
```

```
    writeln;
    end;
  write('Press <ENTER>...');
  readln;
  end;
procedure l(t1,t2:integer);  {基本跳法 C}
  begin
  a[t1,t2-1]:=1; a[t1,t2]:=0; a[t1,t2+1]:=0;
  print;
  a[t1+1,t2]:=0; a[t1+2,t2]:=0; a[t1,t2]:=1;
  print;
  a[t1,t2-1]:=0; a[t1,t2]:=0; a[t1,t2+1]:=1;
  print;
  end;
procedure h(t1,t2:integer);  {基本跳法 B}
  begin
  a[t1,t2]:=0;a[t1+1,t2]:=0;a[t1-1,t2]:=1;
  print;
  a[t1,t2+1]:=0;a[t1,t2+2]:=0;a[t1,t2]:=1;
  print;
  a[t1-1,t2]:=0;a[t1,t2]:=0;a[t1+1,t2]:=1;
  print;
  end;
procedure Mo2(t1,t2:integer);  {另一种基本跳法}
  begin
  a[t1,t2]:=0;a[t1+1,t2]:=0;a[t1+2,t2]:=1;print;
  a[t1,t2+1]:=0;a[t1+1,t2+1]:=0;a[t1+2,t2+1]:=1;print;
  a[t1+2,t2]:=0;a[t1+2,t2+1]:=0;a[t1+2,t2+2]:=1;print;
  a[t1,t2+2]:=0;a[t1,t2+3]:=0;a[t1,t2+1]:=1;print;
  a[t1+2,t2+2]:=0;a[t1+1,t2+2]:=0;a[t1,t2+2]:=1;print;
  a[t1,t2+1]:=0;a[t1,t2+2]:=0;a[t1,t2+3]:=1;print;
  end;
begin
  clrscr;
  write('N='); readln(n);
  if n mod 3 = 0 then begin write('No Way!');readln;halt;end;  {无解}
  num:=0;
  fillchar(a,sizeof(a),0);
  for i:=1 to n do
    for j:=1 to n do
      a[i,j]:=1;
  print;
  k1:=n div 3;
  if n mod 3 = 1 then dec(k1);
```

```
  for i:=1 to k1 do
   for j:=1 to n-3 do
    l((i-1)*3+1,j);
  for i:=1 to k1*3 do h(i,n-2);
  case n mod 3 of {分类解决}
   1:begin
    for i:=1 to k1+1 do h(3*k1+1,(i-1)*3+1);
    for i:=1 to n-2 do l(3*k1+2,i);
    a[n,n]:=0; a[n,n-1]:=0; a[n,n-2]:=1; print;
    a[n-1,n]:=0; a[n-1,n-1]:=0; a[n-1,n-2]:=1; print;
    a[n,n-2]:=0; a[n-1,n-2]:=0; a[n-2,n-2]:=1; print;
    a[n-3,n]:=0; a[n-2,n]:=0; a[n-1,n]:=1; print;
    a[n-2,n-2]:=0; a[n-2,n-1]:=0; a[n-2,n]:=1; print;
    a[n-1,n]:=0; a[n-2,n]:=0; a[n,n]:=1; print;
    end;
   2:begin
    for i:=1 to k1 do
     mo2(3*k1+1,(i-1)*3+1);
    a[3*k1+1,3*k1+1]:=0;a[3*k1+2,3*k1+1]:=0;a[3*k1,3*k1+1]:=1;print;
    a[3*k1+1,3*k1+2]:=0;a[3*k1+2,3*k1+2]:=0;a[3*k1,3*k1+2]:=1;print;
    a[3*k1,3*k1+2]:=0;a[3*k1,3*k1+2]:=0;a[3*k1,3*k1]:=1;print;
    end;
  end;
end.
```

【附录 3】01 串问题的实现(Sequence.pas)

输入输出格式见原题。

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
{$M 64000,0,655360}
program Sequence;
const
   fin      =     'sequence.in';                    {输入文件}
   fon      =     'sequence.out';                   {输出文件}
   maxN     =     1000;                  {01 串长度的最大值}
var
   N        :     integer;                          {01 串长度}
   A        :     array[1..2,0..2]of integer;              {限定数值}
   D        :     array[0..maxN] of integer;               {最长路径}
   V        :     array[0..maxN]of integer;   {v[I]表示从点 I 出发的有向边条数}
     E      :     array[0..maxN,1..6,0..1]of integer;
       {E[I,J,0]表示从点 I 出发的第 J 条有向边指向的顶点，E[I,J,1]表示这条边的权值}
   i,j,k,l  :     integer;
```

```
procedure Input; {输入数据}
  var f:text;
  begin
  assign(f,fin); reset(f);
  read(f,n);
  for i:=1 to 2 do
    read(f,a[i,1],a[i,2],a[i,0]);
  j:=a[1,1];
  a[1,1]:=a[1,0]-a[1,2];
  a[1,2]:=a[1,0]-j;
  close(f);
  end;
procedure Construct; {构建图的模型}
  begin
  fillchar(v,sizeof(v),0);
  for i:=1 to n do
    begin
    inc(v[i-1]); inc(v[i]);
    e[i-1,v[i-1],0]:=i;
    e[i-1,v[i-1],1]:=0;
    e[i,v[i],0]:=i-1;
    e[i,v[i],1]:=-1;
    end;
  for i:=1 to 2 do
    for j:=0 to n-a[i,0] do
      begin
      k:=j+a[i,0];
      inc(v[j]); inc(v[k]);
      e[j,v[j],0]:=k;
      e[j,v[j],1]:=a[i,1];
      e[k,v[k],0]:=j;
      e[k,v[k],1]:=-a[i,2];
      end;
  end;
procedure Done; {输出}
  var f:text;
  begin
  assign(f,fon); rewrite(f);
  if l> n
  then write(f,-1) else
    for i:=1 to n do
      write(f,d[i]-d[i-1]);
  writeln(f); close(f);
  end;
```

```
begin
  Input;
  Construct;
  fillchar(d,sizeof(d),0);
  l:=0; {迭代次数}
  repeat {求最长路径}
    inc(l); k:=0;
    for i:=0 to n do
      for j:=1 to v[i] do
        if d[i]+e[i,j,1] > d[e[i,j,0]] then
          begin
          d[e[i,j,0]]:=d[i]+e[i,j,1];
          k:=1;
          end;
  until (k=0) or (l>n);
  Done;
end.
```

【附录 4】旅馆问题的实现(Hotel.pas)

输入格式：第一行是暑期的总天数 n 和客房总数。第 2–n 行是一个邻接矩阵的上三角（不包括对角线），表示预定日期为第 I 天开房，第 J 天退房的订单预定的房间总数。

输出格式：第一行是总收益天数 ◦接下来的一个邻接矩阵的上三角（不包括对角线），表示对于预定日期为第 I 天开房，第 J 天退房的订单，接受的房间总数。

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
{$M 64000,0,655360}
program Hotel;
const
   fin       =     'hotel.in';                    {输入文件}
   fon       =     'hotel.out';                   {输出文件}
   maxN      =     100;                           {最大总天数}
var
      N, M      :     integer;                    {总天数和客房总数}
         C         :     longint;                       {最优解}
         a         :     array[1..maxn,1..maxn]of integer;          {容量上限}
         {由于构造的流网络可能是一个多重图（连结两点不只一条边），按其表示的意义不同分为两类}
      f         :     array[1..maxn,1..maxn]of integer;      {第一类边流量}
         f1        :     array[1..maxn]of integer;          {第二类边流量}
         d, p      :     array[1..maxn]of integer;              {最大费用}
         i,j,k,l   :     integer;

      procedure Input; {输入}
  var fi:text;
```

```
  begin
  assign(fi,fin); reset(fi);
  readln(fi,n,m);
  fillchar(a,sizeof(a),0);
  for i:=1 to n-1 do
    begin
    for j:=i+1 to n do
      read(fi,a[i,j]);
    readln(fi);
    end;
  close(fi);
  end;

procedure Out; {输出}
  var fo:text;
  begin
  assign(fo,fon); rewrite(fo);
  writeln(fo,C);
  for i:=1 to n-1 do
    begin
    for j:=i+1 to n-1 do
      write(fo,f[i,j],' ');
    writeln(fo,f[i,n]);
    end;
  close(fo);
  end;

begin
  input;
  for k:=1 to m do  {m 次增流}
    begin
    for j:=2 to n do
      begin d[j]:=-1; p[j]:=0; end;
    p[1]:=1; d[1]:=0;
    repeat {寻找最大费用路径}
      l:=0;
      for i:=1 to n do
        for j:=1 to n do
         if (p[i]>0) and (f[i,j]<a[i,j]) and (d[i]+(j-i)>d[j]) then
          begin
          d[j]:=d[i]+(j-i);
          p[j]:=i; l:=1;
          end;
      for i:=1 to n-1 do
```

```
      if (p[i]>0) and (d[i]>d[i+1]) then
        begin
        d[i+1]:=d[i];
        p[i+1]:=i; l:=1;
        end;
    for i:=1 to n-1 do
      if (p[i+1]>0) and (f1[i]>0) and (d[i+1]>d[i]) then
        begin
        d[i]:=d[i+1];
        p[i]:=i+1; l:=1;
        end;
  until (l=0);
  if p[n] > 0 then  {若存在增流路径}
    begin
    i:=p[n]; j:=n;
    c:=c+d[n];
    repeat
      if abs(d[i]-d[j])=0 then
        if i<j
        then f1[i]:=f1[i]+1
        else f1[j]:=f1[j]-1
      else
        begin
        f[i,j]:=f[i,j]+1;
        f[j,i]:=-f[i,j];
        end;
      j:=i; i:=p[i];
    until j=1;
    end;
  end;
 out;
end.
```

【附录 4】千足虫问题的实现

输入输出格式见原题。

注：本题附有千足虫的动作模拟程序（Tox_Simu.pas）和搜索算法的实验程序（Toxic_s.pas）。

```
program Toxic_Game;
const
 name1      =     'toxic.in';  {输入文件}
 name2      =     'toxic.out'; {输出文件}
 go         :     array[1..4, 1..2] of shortint {平面上 4 个方向的坐标增量}
            =     ((0, 1), (1, 0), (0, -1), (-1, 0));
```

```
  six          :      array[1..6, 1..3] of shortint {立方体 6 个相邻面的坐标增量}
               =      ((1, 0, 0), (0, 1, 0), (0, 0, 1),
                      (-1, 0, 0), (0, -1, 0), (0, 0, -1));
type
  Tsize        =      array[1..3] of integer;
  Tblocks      =      array[0..33, 0..33, 0..33] of shortint;
var
  size         :      Tsize;  {长方体的长、宽、高}
  StoE, EtoS,                  {StoE[i]由于坐标变换}
  BStoE        :      Tsize;  {BStoE 表示 Best Size to Experiment，记录下最优的坐
标变换}
  mx, my, mz   :      integer; {maxX、maxY、maxZ}
  ans, sum     :      integer; {ans 存放当前最优解，sum 存放当前吃掉的立方体数目}
  blocks       :      Tblocks; {存放当前路径的信息}
       {blocks[x,y,z]:  -1 表示走过该格  -2 表示已经吃掉该格  大等于 0 表示当前暴露的
面的数目}
  b2           :      ^TBlocks; {记录路径，为“拾遗”做准备}
  outf         :      text;    {输出文件}
  print        :      boolean;  {寻找最优解答和打印解用同一个过程，print 表示是否打
印}

procedure init; {文件初始化}
  var f : text;
  begin
  assign(f, name1);
  reset(f);
  readln(f, size[1], size[2], size[3]);
  close(f);

  assign(outf, name2);
  rewrite(outf)
  end;

procedure initblocks; {初始化 blocks}
  var x, y, z : integer;
  begin
  fillchar(blocks, sizeof(blocks), 255);
  for x := 1 to mx do
    for y := 1 to my do
      for z := 1 to mz do
        blocks[x, y, z] := 0
  end;

procedure say(ch : char; x, y, z : integer); {当 print=TRUE 时输出一个命令}
```

```
  var o : Tsize;
  begin
  if not print then exit;
  o[ EtoS[1] ] := x;
  o[ EtoS[2] ] := y;
  o[ EtoS[3] ] := z;
  writeln(outf, ch, ' ', o[1], ' ', o[2], ' ', o[3])
  end;

function eat_block(x, y, z : integer) : boolean; {吃掉立方体(x,y,z)，返回是否成功}
  var i, xx, yy, zz : integer;
  begin
  if (x = 0) or (x > mx) or
     (y = 0) or (y > my) or
     (z = 0) or (z > mz) or
     (blocks[x, y, z] <> 1)
    then begin eat_block := false; exit end;
  eat_block := true;
  blocks[x, y, z] := -2;
  say('E', x, y, z);
  inc(sum);
  for i := 1 to 6 do
    begin
    xx := x + six[i, 1];
    yy := y + six[i, 2];
    zz := z + six[i, 3];
    if blocks[xx, yy, zz] >=0
      then inc(blocks[xx, yy, zz])
    end
  end;

procedure pick_block(x, y, z : integer);
{对(x,y,z)拾遗，如果立方体(x,y,z)只有一个面与路径接触，则吃掉(x,y,z)不会影响路径}
  var i, j, xx, yy, zz : integer;
  begin
  if (x = 0) or (x > mx) or
     (y = 0) or (y > my) or
     (z = 0) or (z > mz) or
     (blocks[x, y, z] <> 1)
    then exit;
  j := 0;
  for i := 1 to 6 do
    begin
    xx := x + six[i, 1];
```

```
    yy := y + six[i, 2];
    zz := z + six[i, 3];
    if (xx >= 1) and (xx <= mx) and
       (yy >= 1) and (yy <= my) and
       (zz >= 1) and (zz <= mz) and (b2^[xx, yy, zz] = -1)
      then inc(j)
    end;
  if j = 1 then eat_block(x, y, z)
  end;

procedure pick_6(x, y, z : integer); {对 6 个方向拾遗}
  var i : integer;
  begin
  if (z > 0) and print
    then for i := 1 to 6 do
           pick_block(x + six[i, 1], y + six[i, 2], z + six[i, 3])
  end;

procedure make_way; {按照构造的方法路径}
  var nx, ny, nz, ns : integer;
{(nx, ny, nz)表示当前坐标，ns 表示当前状态，ns=1 表示在主线上移动，ns>1 表示在
相邻主线的衔接处}
      ford, upd      : integer;

  procedure odd_plane; {奇数层的移动和吃食}
  var t, i, h : integer;
  begin
  eat_block(nx, ny, nz);
  pick_6(nx, ny, nz-1);
  blocks[nx, ny, nz-1] := -1;
  say('M', nx, ny, nz);
  if ns = 2
    then ns := 4
    else ns := 1;
  if ny = 1
    then upd := 1
    else upd := 3;

  repeat
    case ns of
    1 : begin
        if nx = 1
          then ford := 2
          else ford := 4;
```

```
      h := 1;
      while eat_block(nx + go[ford, 1], ny, nz) do
        begin
        inc(h);
        eat_block(nx, ny, nz-1);
        if nz+1 = mz then eat_block(nx, ny, nz+1);
        if (upd <> 1) or (h < mx-1) then eat_block(nx, ny+1, nz);
        if (upd <> 3) or (h < mx-1) then eat_block(nx, ny-1, nz);
        pick_6(nx, ny, nz);
        blocks[nx, ny, nz] := -1;
        nx := nx + go[ford, 1];
        say('M', nx, ny, nz)
        end;
      ns := 2
      end;
2..4 : begin
      if eat_block(nx, ny + go[upd, 2], nz)
        then begin
            eat_block(nx, ny, nz-1);
            if nz+1 = mz then eat_block(nx, ny, nz+1);
            pick_6(nx, ny, nz);
            blocks[nx, ny, nz] := -1;
            ny := ny + go[upd, 2];
            say('M', nx, ny, nz);
            ns := ns mod 4+1
            end
        else break
      end
  end
until false
end;

procedure even_plane; {偶数层的移动和吃食}
begin
if not eat_block(nx, ny, nz) then exit;
pick_6(nx, ny, nz-1);
blocks[nx, ny, nz-1] := -1;
say('M', nx, ny, nz)
end;

begin
{对坐标进行变换}
sum := 0;
EtoS[ StoE[1] ] := 1;
```

```
EtoS[ StoE[2] ] := 2;
EtoS[ StoE[3] ] := 3;
mx := Size[ EtoS[1] ];
my := Size[ EtoS[2] ];
mz := Size[ EtoS[3] ];
{开始构造}
initblocks;
blocks[1, 1, 1] := 1;
nx := 1; ny := 1; nz := 0; ns := 3;

for nz := 1 to mz do
  if odd(nz)
    then odd_plane
    else even_plane;

{更新当前最优解}
if sum > ans
  then begin ans := sum; b2^ := blocks; BStoE := StoE end
end;

{主程序}
begin
new(b2);
init;
ans := -1; min := maxint;
print := false;

for StoE[1] := 1 to 3 do
  for StoE[2] := 1 to 3 do
    for StoE[3] := 1 to 3 do
      if [ StoE[1], StoE[2], StoE[3] ] = [1..3]
        then make_way;

StoE := BStoE;
print := true;
make_way;

close(outf);
writeln('Eat = ', sum);
writeln('Rate = ', sum /mx/my/mz :0 :2)
end.
```