2001 年 IOI 試題與參考解答

Mobile phones

題目 (PROBLEM)

假設第四代行動電話在 Tampere 地區的基地台運作方式如下。這個區域被分割成方格。方格構成一個 SxS 的矩陣，而每個方格由橫列座標（row）和縱行座標（column）來索引，座標值範圍由 0 到 S-1。每一個方格有一個基地台。每個方格內，使用中的行動電話數目會更動，其原因為一個行動電話會從一個方格移動到另一個方格、或者行動電話關機或開機。每隔一段時間，每一個基地台會向主基地台報告使用中行動電話的數目，以及該基地台的橫列座標與縱行座標值。

請寫一個程式，接收來自這些基地台的報告，並回答查詢，如目前某一矩形區域內使用中的行動電話總數。

輸入與輸出 (INPUT AND OUTPUT)

輸入請由標準輸入讀取整數。而回答查詢請輸出整數到標準輸出。輸入的格式如下。每一筆輸入由一橫列的整數構成。第一個整數為指令，依據該指令，其後可能跟隨數個參數（皆為整數)。參數的格式如下表。

指令	參數	意義
0	S	初始化地區成為 SxS 方形.，且皆為零。 這個指令為輸入的第一個指令，而且只會被給一次。
1	X Y A	加 A 到方形（X,Y）表格中之使用中行動電話的數目。A 可以是正數或負數
2	L B R T	查詢目前在矩形區域內的行動電話總數。該總數由每個方形（X,Y)， L≦X≦R,B≦Y≦T. 的行動電話數目加總而成。
3		結束程式執行。這個指令為輸入的最後一個指令而且只會被給一次。

輸入所給的數值將在合法的數值範圍內，所以不用檢查輸入的正確性。如 A 為負值，則方格表中小於零者將不再遞減。座標值由 0 起始，例如一個表若有大小為 4x4, 座標值為 0≦X≦3 和 0≦Y≦3。

若輸入不是指令 2 時，你的程式不應該做不必要的回答。假設輸入的指令為 2 時，你的程式應回答該查詢，把答案（一個整數）輸出到標準輸出。

程式規範 (PROGRAMMING INSTRUCTIONS)

在下面的例子，整數 last 是一列中被讀取的最後一個整數，而 answer 是含有你的答案的整數變數

假設你的程式用 C++撰寫並使用 iostreams，你應該使用下面的方法從標準輸入讀取與寫到標準輸出。

```
cin>> last ;
cout<< answer<< endl<< flush;
```

假設你的程式用 C 或 C++撰寫，並使用 scanf 以及 printf，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
scanf( "%d" ,&last);
printf( "%d\n" ,answer); fflush (stdout);
```

假使你的程式用 Pascal 撰寫，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
Read(last);..... Readln;
Writeln(answer);
```

範例 (EXAMPLE)

標準輸入	標準輸出	解釋
0 4		初始化表格成 4x4。
1 1 2 3		更改方格(1,2),令其值+3。
2 0 0 2 2		查詢矩形區域 0≦X≦2,0≦Y≦2 中行動電話的總數。
	3	回答查詢。
1 1 1 2		更改方格(1,1),令其值+2。
1 1 2 -1		更改方格(1,2),令其值 -1。
2 1 1 2 3		查詢矩形區域 0≦X≦2,0≦Y≦2 中行動電話的總數。
	4	回答查詢。
3		結束程式。

數值限制 (CONSTRAINTS)

表格大小	S ×S	1×1≦S×S≦1024×1024
方格內的值	V	$0 \leqq V \leqq 2^{15}-1(=32767)$
更改值	A	$-2^{15} \leqq A \leqq 2^{15}-1(=32767)$
輸入指令的數目	U	3≦U≦60002
在整個表格中最大的可能行動電話數目	M	$M=2^{30}$

在 20 組的輸入中，其中 16 組的表格大小最多為 512x512。

注意：網站中測試的功能會把你的測試檔餵到你程式的標準輸入

Mobiles - example solution (Timo Tossavainen)

```
/*
PROB: mobiles
LANG: c
*/
/* IOI 2001 mobiles problem: 2-dimensional binary indexed tree solution */
#include <stdio.h>
#define MAX_SIZE 1024
void init (int size);
void update (int x, int y, int amount);
int sum (int x1, int y1, int x2, int y2);
int main()
{
int cmd, a1, a2, a3, a4;
do
{
scanf ("%d", &cmd);
switch (cmd)
{
case 0:
scanf ("%d", &a1);
init (a1);
break;
case 1:
scanf ("%d %d %d", &a1, &a2, &a3);
update (a1, a2, a3);
break;
case 2:
scanf ("%d %d %d %d", &a1, &a2, &a3, &a4);
printf ("%d\n", sum(a1, a2, a3, a4));
fflush (stdout);
break;
default:
}
} while(cmd != 3);
return 0;
}
#define LOW_BIT(x) ((x) & ((x) ^ ((x) - 1)))
int size = 0;
int table [MAX_SIZE][MAX_SIZE];
void init (int sz)
{
for (size = 1; size < sz; size <<= 1)
;
}
```

```
int sum (int x1, int y1, int x2, int y2)
{
int res, ix1, ix2, iy1, iy2;
res = 0;
for(iy2 = y2+1; iy2 > y1; iy2 -= LOW_BIT(iy2))
{
for (ix2 = x2+1; ix2 > x1; ix2 -= LOW_BIT(ix2))
res += table[ix2-1][iy2-1];
for (ix1 = x1; ix1 > ix2; ix1 -= LOW_BIT(ix1))
res -= table[ix1-1][iy2-1];
}
for(iy1 = y1; iy1 > iy2; iy1 -= LOW_BIT(iy1))
{
for (ix2 = x2+1; ix2 > x1; ix2 -= LOW_BIT(ix2))
res -= table[ix2-1][iy1-1];
for (ix1 = x1; ix1 > ix2; ix1 -= LOW_BIT(ix1))
res += table[ix1-1][iy1-1];
}
return res;
}
void update (int x, int y, int amount)
{
int ix;
x++; y++;
for(; y <= size; y += LOW_BIT(y))
for(ix = x; ix <= size; ix += LOW_BIT(ix))
table[ix-1][y-1] += amount;
}
```

Ioiwari Game

題目（PROBLEM）

曼卡拉類型的彈珠（beads）與坑洞（pits）遊戲是最古老的娛樂遊戲，本題目是專為 IOI 設計的新遊戲版本。這種兩人玩的遊戲是在一個圓盤邊緣的 7 個坑洞進行。此外，每一個人有自己的彈珠放置區（bank)。遊戲開始時先將 20 顆彈珠隨意分配至圓盤上的坑洞裡，且每個坑洞有至少 2 顆、至多 4 顆彈珠。兩位遊戲者依以下規則輪流移動圓盤上的彈珠。首先選擇一個非空的坑洞，將該坑洞的所有彈珠取出放在手上。只要手上擁有彈珠，則依順時鐘方向，自取出彈珠的下一個坑洞開始執行下列動作：

- 手上超過一顆彈珠的情形：如果現在的坑洞裡有 5 顆彈珠，則自坑洞裡移出一顆彈珠放至自己的彈珠放置區，否則則從手上取出一顆彈珠放至坑洞裡。
- 手上只有一顆彈珠的情形：如果現在的坑洞裡有至少一顆、至多 4 顆彈珠，則將該坑洞裡的所有彈珠及手上的一顆彈珠移至自己的彈珠放置區，否則（亦即坑洞裡有 0 或 5 顆彈珠)，則將手上的彈珠放至對手的彈珠放置區。

當所有的坑洞都沒有彈珠時，則遊戲結束，而以彈珠放置區裡彈珠最多者獲勝。

在本遊戲中先玩者（starting player）一定有一個穩贏策略。請寫一個程式，擔任 IOI 遊戲的先玩者，並要獲勝。你的程式的對手會盡力取勝，一旦有機會，它就會打敗你的程式取得勝利。

輸入與輸出（INPUT AND OUTPUT）

你的程式應從標準輸入(standard input)讀取資料，並將輸出寫至標準輸出(standard output）。你的程式是先玩者（player 1），對方是後玩者（player 2）。你的程式一開始必須先讀入在同一行上的 7 個整數，p1, …, p7：即坑洞 1 至 7 的起始彈珠數。坑洞依順時鐘方向標示為 1 至 7。

遊戲開始時，彈珠放置區是空的，而遊戲開始後，你的程式依下列規則進行輸入或輸出：

- 假如輪到你的程式玩，將選定的坑洞標示寫至標準輸出。
- 假如輪到你的對手玩，則自標準輸入讀入對手所選定坑洞的標示。

工具（TOOLS）

你會被給予一個程式（在 Linux 上為 ioiwari2，在 Windows 上為 ioiwari2.exe），扮演後玩者（Player 2）的角色，這個程式首先會輸出一行資料，為坑洞 1 至 7 的起始彈珠數，如

4 3 2 4 2 3 2

然後開始進行遊戲，即自標準輸入讀入先玩者（Player 1）的玩法，並將它自己的玩法寫至標準輸出。你可以同時在兩個視窗執行你的程式及 ioiwari2，並手動地模擬兩個程式相的交談情形。

程式規範（PROGRAMMING INSTRUCTIONS）

在下面的例子，變數 last 是最後讀入的整數，而變數 mymove 含有你的玩法（move）。

假設你的程式用 C++撰寫並使用 iostreams，你應該使用下面的方法從標準輸入讀取與寫到標準輸出。

```
count<<mymove<<endl<<flush;
cin>>last;
```

假設你的程式用 C 或 C++撰寫，並使用 scanf 以及 printf，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
printf( "%d\n", mymove); fflush (stdout);
scanf( "%d",&last);
```

假使你的程式用 Pascal 撰寫，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
Writeln (mymove);
Readln (last);
```

範例（EXAMPLE）

底下是一個包含 6 個步驟的完整遊戲。

玩者/坑洞標示	1	2	3	4	5	6	7	先玩者彈珠放置區	後玩者彈珠放置區
起始狀態	4	3	2	4	2	3	2	0	0
先玩者：2	4	0	3	5	0	3	2	3	0
後玩者：3	4	0	0	4	1	4	0	3	4
先玩者：5	4	0	0	4	0	0	0	8	4
後玩者：4	0	0	0	0	1	1	1	8	9
先玩者：5	0	0	0	0	0	0	1	10	9
後玩者：7	0	0	0	0	0	0	0	11	9

給分方式（SCORING）

如果你的程式贏得勝利，則給予 4 分；平手則給予 2 分；否則 0 分。

Ioiwari - example solutions (Gyula Horvath)

```
/*
PROB: ioiwari
LANG: C++
*/
#include <stdio.h>
#include <iostream.h>
#define TSize 7
#define Total 20
#define Inf (Total * 2 + 1)
#define MaxN 279935L
#define true 1
#define false 0
typedef struct Board
{
char Who;
unsigned char Pit[TSize];
unsigned char Bank[2];
} Board;
static unsigned char T[2][MaxN + 1];
static unsigned char OM1[MaxN + 1];
static Board B;
static long B6N(Board *B)
{
short i;
long a;
a = 0;
for (i = 0; i < TSize; i++)
a = a * 6 + B->Pit[i];
return a;
}
static void Move(Board *B, int i, Board *BB)
{
int S;
int j;
int W0, W;
S = B->Pit[i - 1];
W0 = B->Who;
W = !W0;
*BB = *B;
BB->Who = W;
BB->Pit[i - 1] = 0;
j = i;
while (S > 1) {
j++;
```

```
if (j > TSize)
j = 1;
if (BB->Pit[j - 1] == 5) {
BB->Pit[j - 1]--;
BB->Bank[W0]++;
} else {
BB->Pit[j - 1]++;
S--;
}
}
j++;
if (j > TSize)
j = 1;
if (BB->Pit[j - 1] >= 1 && BB->Pit[j - 1] <= 4) {
BB->Bank[W0] += BB->Pit[j - 1] + 1;
BB->Pit[j - 1] = 0;
} else
BB->Bank[W]++;
}
static short MinMax(Board *B)
{
char i;
Board BB;
short Diffn, Diffs;
long a;
a = B6N(B);
if (T[B->Who][a] != Inf)
return (T[B->Who][a]);
if (B->Who) {
Diffn = -Inf;
for (i = 1; i <= TSize; i++) {
if (B->Pit[i - 1] > 0) {
Move(B, i, &BB);
Diffs = MinMax(&BB) + BB.Bank[true] - BB.Bank[0] - B->Bank[true] +
B->Bank[0];
if (Diffs > Diffn) {
Diffn = Diffs;
OM1[a] = i;
}
}
}
} else {
Diffn = Inf;
for (i = 1; i <= TSize; i++) {
if (B->Pit[i - 1] > 0) {
```

```
Move(B, i, &BB);
Diffs = MinMax(&BB) + BB.Bank[true] - BB.Bank[0] - B->Bank[true] +
B->Bank[0];
if (Diffs < Diffn) {
Diffn = Diffs;
/*OM2[a] = i;*/
}
} /*for i*/
}
}
T[B->Who][a] = Diffn;
return Diffn;
}
static void Start(void)
{
int bead, i, Diff;
for (i = 0; i < TSize; i++) {
scanf("%d", &bead);
B.Pit[i] = bead;
}
B.Bank[true] = 0;
B.Bank[0] = 0;
B.Who = true;
for (i = 1; i <= MaxN; i++) {
T[true][i] = Inf;
T[0][i] = Inf;
}
T[true][0] = Total;
T[0][0] = Total;
OM1[0] = 0;
Diff = MinMax(&B);
}
void Play(void)
{
long a;
int i, ii;
for (;;) {
a = B6N(&B);
i = OM1[a];
cout<<i<<endl<<flush;
Move(&B, i, &B);
if (B.Bank[1] + B.Bank[0] == Total)
exit(0);
cin>>ii;
Move(&B, ii, &B);
```

```
if (B.Bank[1] + B.Bank[0] == Total)
exit(0);
}
}
int main(void)
{
Start();
Play();
}
--------
#include <stdio.h>
#define TSize 7
#define Total 20
#define Inf (Total * 2 + 1)
#define MaxN 279935L
#define true 1
#define false 0
typedef struct Board
{
char Who;
unsigned char Pit[TSize];
unsigned char Bank[2];
} Board;
static unsigned char T[2][MaxN + 1];
static unsigned char OM1[MaxN + 1];
static Board B;
static long B6N(Board *B)
{
short i;
long a;
a = 0;
for (i = 0; i < TSize; i++)
a = a * 6 + B->Pit[i];
return a;
}
static void Move(Board *B, int i, Board *BB)
{
int S;
int j;
int W0, W;
S = B->Pit[i - 1];
W0 = B->Who;
W = !W0;
*BB = *B;
BB->Who = W;
```

```
BB->Pit[i - 1] = 0;
j = i;
while (S > 1) {
j++;
if (j > TSize)
j = 1;
if (BB->Pit[j - 1] == 5) {
BB->Pit[j - 1]--;
BB->Bank[W0]++;
} else {
BB->Pit[j - 1]++;
S--;
}
}
j++;
if (j > TSize)
j = 1;
if (BB->Pit[j - 1] >= 1 && BB->Pit[j - 1] <= 4) {
BB->Bank[W0] += BB->Pit[j - 1] + 1;
BB->Pit[j - 1] = 0;
} else
BB->Bank[W]++;
}
static short MinMax(Board *B)
{
char i;
Board BB;
short Diffn, Diffs;
long a;
a = B6N(B);
if (T[B->Who][a] != Inf)
return (T[B->Who][a]);
if (B->Who) {
Diffn = -Inf;
for (i = 1; i <= TSize; i++) {
if (B->Pit[i - 1] > 0) {
Move(B, i, &BB);
Diffs = MinMax(&BB) + BB.Bank[true] - BB.Bank[0] - B->Bank[true] +
B->Bank[0];
if (Diffs > Diffn) {
Diffn = Diffs;
OM1[a] = i;
}
}
}
```

```
} else {
Diffn = Inf;
for (i = 1; i <= TSize; i++) {
if (B->Pit[i - 1] > 0) {
Move(B, i, &BB);
Diffs = MinMax(&BB) + BB.Bank[true] - BB.Bank[0] - B->Bank[true] +
B->Bank[0];
if (Diffs < Diffn) {
Diffn = Diffs;
/*OM2[a] = i;*/
}
} /*for i*/
}
}
T[B->Who][a] = Diffn;
return Diffn;
}
static void Start(void)
{
int bead, i, Diff;
for (i = 0; i < TSize; i++) {
scanf("%d", &bead);
B.Pit[i] = bead;
}
B.Bank[true] = 0;
B.Bank[0] = 0;
B.Who = true;
for (i = 1; i <= MaxN; i++) {
T[true][i] = Inf;
T[0][i] = Inf;
}
T[true][0] = Total;
T[0][0] = Total;
OM1[0] = 0;
Diff = MinMax(&B); }
void Play(void)
{
long a;
int i, ii;
for (;;) {
a = B6N(&B);
i = OM1[a];
printf("%d\n", i); fflush(stdout);
Move(&B, i, &B);
if (B.Bank[1] + B.Bank[0] == Total)
```

```
exit(0);
scanf("%d", &ii);
Move(&B, ii, &B);
if (B.Bank[1] + B.Bank[0] == Total)
exit(0);
}
}
int main(void)
{
Start();
Play();
}
--------
Program Owari;
Const
TSize=7; {# pits}
Total=20; {total number of beads in pits}
Inf =2*Total+1; {infinity value for minimax}
MaxN =6*6*6*6*6*6*6-1; {max. base-6 number=279936}
Type
Board=Record
Who:Boolean; {=True: first player moves next}
Pit:Array[1..TSize] of Byte; {Pit contents }
Bank:Array[Boolean] of Integer;{Banks for players}
End;
Var
T:Array[Boolean,0..MaxN] of Byte;
{T[w,b] is the best score difference for player w, that can be obtained
from game position whose base-6 number is b }
OM1: Array[0..MaxN] of Byte; {winning moves for the first player}
B:Board; {current game board}
i:Longint;
Function B6N(Var B:Board):Longint;
{Base-6 number id of board B}
Const
P6:Array[1..TSize] of
Longint=(6*6*6*6*6*6,6*6*6*6*6,6*6*6*6,6*6*6,6*6,6,1);
Var i:Integer;
a:Longint;
{Remark: Rotational equivalent of B that gives the least B6N should be taken
first. This would reduce the required memory to 121305 from 279936 }
Begin{B6N};
a:=0;
For i:=1 To TSize Do Begin
a:=a+B.Pit[i]*P6[i];
```

```
End;
B6N:=a;
End{B6N};
Procedure Move(Var B:Board; i:Byte; Var BB:Board);
Var S:Integer;
j:Byte;
W0,W:Boolean;
Begin
S:=B.Pit[i];
W0:=B.Who; {current player}
W:=Not W0; {opposite player}
BB:=B;
BB.Who:=W;
BB.Pit[i]:=0;
j:=i;
While S>1 Do Begin
Inc(j); If j>Tsize Then j:=1;
If BB.Pit[j]=5 Then Begin
Dec(BB.Pit[j]);
Inc(BB.Bank[W0]);
End Else Begin
Inc(BB.Pit[j]);
Dec(S);
End;
End{while};
Inc(j); If j>Tsize Then j:=1;
If (BB.Pit[j]>=1)And(BB.Pit[j]<=4) Then Begin
Inc(BB.Bank[W0],BB.Pit[j]+1);
BB.Pit[j]:=0;
End Else Begin
Inc(BB.Bank[W]);
End;
End{Move};
Function MinMax(Var B:Board):Integer;
{Returns the optimal score difference, and computes the optimal move for the
first player. Computation is by recursion with memoization. }
Var
i:Byte;
BB:Board;
Diffn,Diffs:Integer;
a:Longint;
Begin{MinMax}
a:=B6N(B); {base-6 number for B}
If (T[B.Who,a])<>Inf Then Begin {already computed, }
MinMax:=T[B.Who, a]; {read the value from T}
```

```
Exit;
End;
If B.Who Then Begin {first player moves}
Diffn:=-Inf;
For i:=1 To TSize Do
If (B.Pit[i]>0) Then Begin
Move(B,i,BB);
Diffs:=MinMax(BB)+(BB.Bank[True]-BB.Bank[False])-
(B.Bank[True]-B.Bank[False]);
If Diffs>Diffn Then Begin
Diffn:=Diffs;
OM1[a]:=i;
End;
End;
;{for i}
End Else Begin {second player moves}
Diffn:=Inf;
For i:=1 To TSize Do
If (B.Pit[i]>0) Then Begin
Move(B,i,BB);
Diffs:=MinMax(BB)+(BB.Bank[True]-BB.Bank[False])-
(B.Bank[True]-B.Bank[False]);
If Diffs<Diffn Then Begin
Diffn:=Diffs;
End;
End;{for i}
End;
T[B.Who, a]:=Diffn; {memoise}
MinMax:=Diffn;
End{MinMax};
Procedure Play(Var B:Board);
{Play the game}
Var i,ii:Integer;
a:Longint;
Begin
While True Do Begin
a:=B6N(B);
i:=OM1[a];
Move(B,i, B);
WriteLn(i);
If B.Bank[true]+B.Bank[false]=Total Then halt;
ReadLn(ii);
Move(B, ii, B);
If B.Bank[true]+B.Bank[false]=Total Then halt;
End{while};
```

```
End{Play};
Begin{Prog}
For i:=1 To TSize Do {read the initial game position}
ReadLn(B.Pit[i]);
B.Bank[True]:=0; {init bank contents}
B.Bank[False]:=0;
B.Who:=True; {first player moves first}
For i:=1 To MaxN Do Begin {init for minimax computation}
T[True, i]:=Inf;
T[False,i]:=Inf;
End;
T[True, 0]:=Total;
T[False,0]:=Total;
OM1[0]:=0;
MinMax(B); {compute optimal moves for the first player}
Play(B); {play the game}
End.
```

Twofive

題目 (PROBLEM)

聖誕老人和他的小幫手們間之秘密訊息，係以 25 個字母的語言編碼，此 25 個字母與一般英文字母相同，而唯一的差別是沒有 Z。換句話說，這些字母為 A 到 Y 的 25 個字母，順序亦與英文字母一致。在這 25 個字母的語言中，每一個語言字串 (word) 恰好為 25 個不同字母，每一個字串 (word) 可以寫在 5x5 的表格中；以一橫列 (row)一橫列 (row) 的方式填寫，舉下列的字串為例：一個字串 ADJPTBEKQUCGLRVFINSWHMOXY 可寫在下列表格中：

A	D	J	P	T
B	E	K	Q	U
C	G	L	R	V
F		N	S	W
	I			
H	M	O	X	Y

在這 25 個字母的語言中，任何一個正確的字串，每縱行 (column) 中的字母順序為遞增 (in ascending order)，每橫列中的字母順序也為遞增，例如，ADJPTBEKQUCGLRVFINSWHMOXY 為正確字串，而 ADJPTBEGQUCKLRVFINSWHMOXY 為不正確字串 (其中字母順序於第 2 縱行 及第 3 縱行並非一直遞增)。

聖誕老人有一本字典，在這本字典中，所有正確字串 (word) 皆以字母順序來排序，並且給予一個號碼，號碼從 1 開始。舉例說明：字串 ABCDEFGHIJKLMNOPQRSTUVWXY 的號碼為 1，字串 ABCDEFGHIJKLMNOPQRSUTVWXY 的號碼為 2，第一個字串與第二個字串不同點為 U 與 T 的順序。

很不幸的，聖誕老人的字典有非常多字串，請寫一個程式，當給予一個字串時，程式則印出它相對應的號碼；而當給予一個號碼時，程式則印出它相對應的字串。字典中因不會有超過 2^{31} 個字串。

輸入 (INPUT)

輸入檔名為 twofive.in，輸入檔有兩行 (two lines)。第一行 (first line) 含一個為 'W' 或 'N' 之單一字母的字串。如果第一行 (first line) 包含 'W'，那麼第二行 (second line) 為一個正確字串 (word)。如果第一行 (first line) 包含 'N'，那麼第二行 (second line)為一個號碼，此號碼必然對應於一個正確字串 (word)。

輸出 (OUTPUT)

輸出檔名為 twofive.out，輸出檔只有一行 (one line)。如果輸入檔第二行 (second line) 為一個字串 (word)，輸出此字串所相對應的號碼。如果輸入檔第二行 (second line) 為一個號碼，輸出此號碼所相對應的字串。

輸入與輸出範例 (EXAMPLE INPUTS AND OUTPUTS)

twofive.in

```
W
ABCDEFGHIJKLMNOPQRSTUVWXY
```

twofive.out

```
1
```

twofive.in

```
N
1
```

twofive.out

```
ABCDEFGHIJKLMNOPQRSTUVWXY
```

Twofive - example solution (Tero Karras)

```
{
prog: twofive
lang: pascal
}
const
states=6*6*6*6*6;
var
reclev:longint;
snum:array[0..states-1] of longint;
state:array[0..5] of longint;
known,kncol:array[1..25] of longint;
function calcstate:longint;
var i,a,b,c:longint;
begin
inc(reclev);
a:=0;
for i:=1 to 5 do
a:=a*6+state[i];
if snum[a]<0 then
begin
b:=0;
c:=known[reclev];
if c<0 then
begin
for i:=1 to 5 do
if state[i-1]>state[i] then
begin
inc(state[i]);
inc(b,calcstate());
```

```
dec(state[i]);
end;
end
else
if (state[c-1]>state[c]) and (state[c]+1=kncol[reclev]) then
begin
inc(state[c]);
inc(b,calcstate());
dec(state[c]);
end;
snum[a]:=b;
end;
calcstate:=snum[a];
dec(reclev);
end;
function numconts:longint;
var i : longint;
begin
state[0]:=5;
for i:=1 to 5 do
state[i]:=0;
for i:=0 to states-2 do
snum[i]:=-1;
snum[states-1]:=1;
reclev:=0;
numconts:=calcstate;
end;
procedure display;
var i,j : longint;
tbl : array[1..5,1..5] of longint;
begin
for i:=1 to 5 do
for j:=1 to 5 do
tbl[i,j]:=0;
for i:=1 to 25 do
if known[i]>0 then
tbl[known[i],kncol[i]]:=i;
for i:=1 to 5 do
begin
for j:=1 to 5 do
write(char(tbl[i,j]+64));
writeln;
end;
end;
procedure clearfixed;
```

```
var
i : longint;
begin
for i:=1 to 25 do
known[i]:=-1;
end;
function wordtonum(str : string):longint;
var
i,j,k,cnum,cchr:longint;
begin
clearfixed;
cnum:=1;
for j:=1 to 5 do
for i:=1 to 5 do
begin
cchr:=longint(str[i+(j-1)*5])-64;
for k:=1 to cchr-1 do
if known[k]<0 then
begin
known[k]:=j;
kncol[k]:=i;
inc(cnum,numconts);
known[k]:=-1;
end;
known[cchr]:=j;
kncol[cchr]:=i;
end;
wordtonum:=cnum;
end;
function numtoword(cnum : longint):string;
var
i,j,k,a:longint;
str:string[30];
begin
clearfixed;
for j:=1 to 5 do
for i:=1 to 5 do
for k:=1 to 25 do
if known[k]<0 then
begin
known[k]:=j;
kncol[k]:=i;
str[i+(j-1)*5]:=char(k+64);
a:=numconts;
if cnum-a<1 then break;
```

```
dec(cnum,a);
known[k]:=-1;
end;
str[0]:=char(25);
numtoword:=str;
end;
var
i,j,k,l,a:longint;
mode:char;
str:string[30];
fi,fo:text;
begin
assign(fi,'twofive.in');
assign(fo,'twofive.out');
reset(fi);
rewrite(fo);
readln(fi,mode);
if mode='W' then
begin
readln(fi,str);
writeln(fo,wordtonum(str));
end
else
begin
read(fi,a);
writeln(fo,numtoword(a));
end;
close(fi);
close(fo);
end.
```

Score

題目（PROBLEM）

Score 是一個雙人對奕的的遊戲。這個遊戲有一組位置(positions)與箭頭(arrows)。每個箭頭從一個位置指到另外一個位置。每一個位置被一個玩家(player)所擁有，該玩家稱之為該位置的擁有者(owner)。另外，每個位置包含了一個正整數值，所有的值都不相同。所有位置中有一個位置是起始位置。遊戲未開始前，每個玩家所擁有的分數都是 0 分。

這個遊戲的玩法如下。我們用 C 來標示目前位置。開始時，C 標示在起始位置上。遊戲的每一步會做以下的操作：

1. 假設 C 所標示的目前位置所含的值比該位置的擁有者的分數還大，那麼 C 標示的位置的值變成該位置的擁有者的分數。反之，則 C 所標示的位置的擁有者的分數保持不變。此外，對手的分數維持不變。
2. 在上一個步驟後，C 位置的擁有者可以選擇一個從目前位置出發的箭頭到另外一個位置。該位置變成新的目前位置。

當 C 所標示的目前位置又回到起始位置時，這個遊戲結束。遊戲的贏家是結束時分數較高的玩家。

箭頭的安排會符合以下的條件：

- 永遠能選一個箭頭離開目前位置。
- 每一個位置 P 都有路徑可從起始位置到達；也就是說，有一連串的箭頭可以從起始位置到 P。
- 這個遊戲保證會結束。

請寫一個程式來玩贏這個遊戲。評分用之測試資料，你的程式皆有機會可贏，不管你是先下還是後下。你的程式的對手永遠選擇最佳的下法，也就是說，一旦你給對手機會，你的對手將贏得該遊戲，而你的程式將會落敗。

輸入與輸出(INPUT AND OUTPUT)

你的程式從標準輸入讀取輸入並將輸出寫到標準輸出。你的程式是玩家一號 (player 1) 而你的對手是玩家二號 (player 2)。當你的程式開始時，必須先從標準輸入讀入下面的輸入資料。

第一行 (line) 包含一個整數，這個整數為位置的數目 N，$1 \leq N \leq 1000$。 位置用 1 到 N 來編號。
接下來的 N 行，每一行包含 N 個整數，這些整數是用來說明箭頭的資訊。假如有個箭頭從位置 i 到位置 j，那麼第 i 行的第 j 個整數的值為 1，否則為 0。

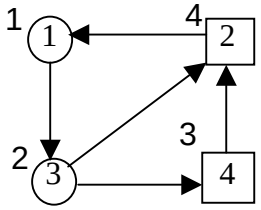
再下一行包含 N 個整數，這些整數說明每個位置的擁有者是誰。假如位置 I 為玩家 1 所擁有，那麼第 i 個整數是 1，否則第 i 個整數為 2。

再下一行包含 N 個整數，說明每個位置所含有的正整數值。假如第 i 個整數是 j，那麼位置 i 所含的正整數值為 j。 j 的可能值介於 1≦j≦N 而且所有位置的值都不相同。

在這些輸入之後，這個遊戲的起始位置為 1。你的程式應該運作如下：

- 假設輪到你的程式下下一步，你的程式應該寫出下一個位置的編號 P，1≦P≦N，到標準輸出。
- 假設輪到你的對手下下一步，你的程式應該從標準輸入讀得下一個位置的編號 P，1≦P≦N。

考慮以下的例子。這個例子用下圖來表示。每個位置用圓或方塊來表示。當一個位置用圓表示時，表示該位置屬於玩家 1，反之若用方塊表示，則表示該位置屬於玩家 2。每一個位置所含的正整數值寫在圓或方塊內。位置的編號則寫在圓或方塊的外緣。



圖中的遊戲的一個可能的對奕過程如下

標準輸入	標準輸出	解釋
4	N	
0 1 0 0		箭頭資料
0 0 1 1		箭頭資料
0 0 0 1		箭頭資料
1 0 0 0		箭頭資料
1 1 2 2		位置的擁有者
1 3 4 2		位置含有的數值
	2	玩家 1 選擇移動到編號 2 位置
	4	玩家 1 選擇移動到編號 4 位置
1		玩家 2 選擇移動到起始位置 – 遊戲結束

遊戲結束後，玩家 1 的分數為 3，而玩家 2 的分數為 2，玩家 1 贏得遊戲。

程式規範 (PROGRAMMING INSTRUCTIONS)

在下面的例子，target 是位置的整數變數。
假設你的程式用 C++撰寫並使用 iostreams，你應該使用下面的方法從標準輸入讀取與寫到標準輸出。

```
cin>>target;
cout<< target<< endl<< flush;
```

假設你的程式用 C 或 C++撰寫，並使用 scanf 以及 printf，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
scanf( “%d” ,&target);
printf( “%d\n” ,target); fflush (stdout);
```

假使你的程式用 Pascal 撰寫，你應該用下列的方法從標準輸入讀取資料與寫資料到標準輸出。

```
Read(target);..... Readln;
Writeln(target);
```

工具 (TOOLS)

一個程式（score2 on Linux, score2.exe on Windows）已提供給你。該程式將從 score.in 讀入遊戲的資訊然後把它的輸出寫到你的程式的輸入。接著該程式採取隨機的策略，從標準輸入讀取的你程式的玩法，然後把它自己的玩法寫到標準輸出。

給分及評分方式 (SCORING AND EVALUATION)

對每一個測試的例子，假設你贏了該遊戲，你將得到滿分，否則你將得 0 分。在評量的過程中，你的程式的第一次執行會和另外一個程式對戰，時間限制為比問題時間限制多一秒。你的程式的輸入與輸出將會被紀錄下來。然後你的程式會被執行第二次，輸入採用第一次執行所記錄成的檔案，然後正式的執行時間才被紀錄。你的程式應該要產生與第一次執行的相同的輸出。

Score - example solution (Timo Poranen)

```
/*
PROB: score
LANG: C++
*/
#include <iostream.h>
const int MAX=1000;
int start_position_passed;
int number_of_positions;
int start_position;
int current_position;
int owner_of_the_position[MAX];
int value_of_the_position[MAX];
int adjacency_matrix[MAX][MAX];
int player_strategy[MAX];
int dfs_number[MAX];
int dfs_counter;
int player_I_max[MAX];
int player_II_max[MAX];
void search(int position) {
dfs_number[position]=dfs_counter;
dfs_counter++;
int player_I_wins[MAX];
int player_II_wins[MAX];
for (int i=0;i<number_of_positions;i++){
player_I_wins[i]=-1;
player_II_wins[i]=-1;
}
for (int i=0;i<number_of_positions;i++) {
if (adjacency_matrix[position][i] !=0) {
if (dfs_number[i]==-1) {
search(i);
if (player_I_max[i]>player_II_max[i]) {
player_I_wins[i]=player_I_max[i];
}
else if (player_I_max[i]<player_II_max[i]) {
player_II_wins[i]=player_II_max[i];
}
else {
}
}
else {
if (player_I_max[i]>player_II_max[i]) {
player_I_wins[i]=player_I_max[i];
}
```

```
else if (player_I_max[i]<player_II_max[i]) {
player_II_wins[i]=player_II_max[i];
}
else {
}
}
}
}
int j=-1;
int max_score=-1;
int opponent_min_score=-1;
if (owner_of_the_position[position]==1) {
for (int i=0;i<number_of_positions;i++) {
if (player_I_wins[i]>-1) {
if (player_I_wins[i]>max_score) {
max_score=player_I_wins[i];
j=i;
}
}
}
if (j != -1) {
player_strategy[position]=j;
player_I_max[position]=max_score;
player_II_max[position]=player_II_max[j];
if (value_of_the_position[position]>max_score) {
player_I_max[position]=value_of_the_position[position];
}
}
else {
for (int i=0;i<number_of_positions;i++) {
if (player_II_wins[i]>-1 && opponent_min_score==-1) {
opponent_min_score=player_II_wins[i];
j=i;
}
else if (player_II_wins[i]>-1 && player_II_wins[i]<opponent_min_score) {
opponent_min_score=player_II_wins[i];
j=i;
}
else {
}
}
}
player_strategy[position]=j;
player_I_max[position]=player_I_max[j];
player_II_max[position]=opponent_min_score;
```

```
if (value_of_the_position[position]>player_I_max[position]) {
player_I_max[position]=value_of_the_position[position];
}
}
else {
for (int i=0;i<number_of_positions;i++) {
if (player_II_wins[i]>-1) {
if (player_II_wins[i]>max_score) {
max_score=player_II_wins[i];
j=i;
}
}
}
if (j != -1) {
player_strategy[position]=j;
player_II_max[position]=max_score;
player_I_max[position]=player_I_max[j];
if (value_of_the_position[position]>max_score) {
player_II_max[position]=value_of_the_position[position];
}
}
else {
for (int i=0;i<number_of_positions;i++) {
if (player_I_wins[i]>-1 && opponent_min_score==-1) {
opponent_min_score=player_I_wins[i];
j=i;
}
else if (player_I_wins[i]>-1 && player_I_wins[i]<opponent_min_score) {
opponent_min_score=player_I_wins[i];
j=i;
}
else {
}
}
player_strategy[position]=j;
player_II_max[position]=player_II_max[j];
player_I_max[position]=opponent_min_score;
if (value_of_the_position[position]>player_II_max[position]) {
player_II_max[position]=value_of_the_position[position];
}
}
}
}
void construct_strategy() {
dfs_counter=0;
```

```
for (int i=0;i<number_of_positions;i++) {
if (owner_of_the_position[i]==1) {
player_I_max[i]=value_of_the_position[i];
player_II_max[i]=0;
}
else if (owner_of_the_position[i]==2) {
player_I_max[i]=0;
player_II_max[i]=value_of_the_position[i];
}
else {
}
dfs_number[i]=-1;
}
search(0);
}
void p1_move(int move)
{
if (move==start_position)
start_position_passed++;
current_position=move;
}
void p2_move(int target)
{
if (target==start_position)
start_position_passed++;
current_position=target;
}
void readInput() {
int i;
int j;
cin>>number_of_positions;
for (i=0;i<number_of_positions;i++) {
for (j=0;j<number_of_positions;j++) {
cin>>adjacency_matrix[i][j];
}
}
for (i=0;i<number_of_positions;i++) {
cin>>owner_of_the_position[i];
}
for (i=0;i<number_of_positions;i++) {
cin>> value_of_the_position[i];
}
}
int main () {
start_position=1;
```

```
start_position_passed=0;
current_position=start_position;
readInput();
construct_strategy();
for (;;) {
if (owner_of_the_position[current_position-1]==1)
{
int move=player_strategy[current_position-1];
move++;
cout<<(move)<<endl<<flush;
p1_move(move);
}
if (current_position==start_position && start_position_passed==1)
{
exit(0);
}
if (owner_of_the_position[current_position-1]==2)
{
int ans;
cin>>ans;
p2_move(ans);
}
if (current_position==start_position && start_position_passed==1)
{
exit(0);
}
}
exit(0);
}
```

Double Crypt

題目（PROBLEM）

高等編碼標準（*AES*）是使用一個新的編碼演算法，它的原理與三個由 128 位元（bits）所組成的字串（blocks）有關。給定一個訊息字串（message block）*p (plaintext)*，鑰匙（key）字串 *k (key)*，編碼函數 *E* 將會傳回一個編碼後的字串 *c (ciphertext)*：

$c = E(p, k)$

反編碼函數 *D* 則是編碼函數 *E* 的反轉 ，如

D（E（p，k），k）= p ，　E（D（c，k），k）= c。

而高等二次編碼標準（*Double AES*）則利用鑰匙字串 k_1 及 k_2 連續編碼二次，先 k_1 再 k_2：

$c_2 = E(E(p, k_1), k_2)$

在此題目中另外給定一個整數 *s*。每個鑰匙字串中只有最左邊的 $4 * s$ 個位元是有用的，其餘的位元（最右邊的 $128 - 4 * s$）皆為 0。

給定訊息及二次編碼後字串，請找出編碼用之鑰匙字串。輸入包含訊息字串 *p*，二次編碼後字串 c_2，及編碼鑰匙結構 *s*。

高等編碼標準中所用之編碼及反編碼演算法可從提供之函式庫中取得。**你應該繳交找到之編碼鑰匙字串，而非你的程式。**

輸入（INPUT）

給予十個測試檔案，檔案名稱為 double1.in 至 double10.in，每一個測試檔案包含三行，第一行為整數 *s*，第二行為訊息字串 *p*，而第三行為 *p* 經過二次編碼後字串 c_2。兩個字串都是由 32 個 16 進位（’0’..’9’,’A’..’F’）數字表示，所提供的函式庫含有將其轉換至字串（block）之函數。所有的測式檔案皆有解。

輸出（OUTPUT）

你應繳交十個對應到測試檔案的輸出檔，每一個輸出檔包含三行。第一行含

#FILE double I

I 是測試檔案號碼。第二行含第一個鑰匙字串 k_1，第三行含第二個鑰匙字串 k_2，也就是

$c_2 = E(E(p, k_1), k_2)$。

這兩個鑰匙字串皆必須用 32 個 16 進位（'0'..'9','A'..'F'）數字表示之，所提供的函式庫含有將其轉換至字串（block）之函數。若有多組解，你只需繳交其中一組。

範例（EXAMPLE）

在此範例我們假設測試檔案號碼為 0。

double0.in

```
1
00112233445566778899AABBCCDDEEFF
6323B4A5BC16C479ED6D94F5B58FF
0C2
```

A possible output file

```
#FILE double 0
A0000000000000000000000000000000
70000000000000000000000000000000
```

函式庫（LIBRARY）

FreePascal library (aeslibp.*):

```
type
  HexStr = String [ 32 ]; { only '0'..'9', 'A'..'F' }
  Block  = array [ 0..15 ] of Byte; { 128 bits }

procedure HexStrToBlock ( const hs: HexStr; var b: Block );
procedure BlockToHexStr ( const b: Block; var hs: HexStr );
procedure Encrypt ( const p, k: Block; var c: Block );
  { c = E(p,k) }
procedure Decrypt ( const c, k: Block; var p: Block );
  { p = D(c,k) }
```

You are given a program aestoolp, which illustrates how to use the FreePascal library.

GNU C/C++ library (Linux and Windows: aeslibc.h, aeslibc.o):

```
typedef char HexStr[33]; /* '0'..'9', 'A'..'F', '\0'-terminated */
typedef unsigned char Block[16];  /* 128 bits */

void hexstr2block ( const HexStr hs, /* out-param */ Block b );
void block2hexstr ( const Block b, /* out-param */ HexStr hs );
void encrypt ( const Block p, const Block k, /* out-param */ Block c );
  /* c = E(p,k) */
```

```
void decrypt ( const Block c, const Block k, /* out-param */ Block p );
  /* p = D(c,k) */
```

aestoolc.c 示範如何使用 GNU C / C++ 函式庫。

限制條件（CONSTRAINTS）

$1 \le s \le 5$

提示（HINT）

一個好的程式應可在 10 秒內找出所需之鑰匙字串。

Double - example solution (Tom Verhoeff)

```
program Double;
{ Copyright (c) 2001, Tom Verhoeff (TUE) }
{ A good solution for task DOUBLE of IOI2001 Competition }
uses AESlibP;
{$B-,I+,Q+,R+,S+}
const
TaskName = 'double';
CaseID: String = ''; { actually a variable, value can be overruled }
InpExtension = '.in';
OutExtension = '.out';
OutHeader = '#FILE'; { required start of header on first line of output file }
HexBits = 4; { # bits per hex }
MaxRelevant = 5; { maximum number of relevant hexits in keys }
var
s : Integer; { key size, i.e. number relevant hexits in keys (task input) }
p : HexStr; { plaintext message (task input) }
c2 : HexStr; { double-encrypted ciphertext message (task input) }
k1 : HexStr; { first recovered key of pair (task output) }
k2 : HexStr; { second recovered key of pair (task output) }
{ derived values }
sBytes: Integer; { # key bytes possibly nonzero }
sPadBits: Integer; { # 0-padding hexits in highest used byte, 0 or 1 }
sMaxLast: Byte; { max value of last byte, $ff=255 or $f0=240 }
sStepLast: Byte; { min non-zero value of last byte, 1 or $10=16 }
pt : Block; { corresponds to p }
ct : Block; { corresponds to c2 }
ck1: Block; { corresponds to k1 }
ck2: Block; { corresponds to k2 }
procedure ReadInput;
var inp: Text;
begin
Assign ( inp, TaskName + CaseID + InpExtension )
; reset ( inp )
; readln ( inp, s )
; readln ( inp, p )
; readln ( inp, c2 )
; Close ( inp )
; sBytes := (s+1) div 2
; sPadBits := 4 * ( 1 - (s-1) mod 2 )
; sMaxLast := ( $ff SHL sPadBits ) AND $ff
; sStepLast := 1 SHL sPadBits
; HexStrToBlock ( p, pt )
; HexStrToBlock ( c2, ct )
end; { ReadInput }
```

```
procedure WriteOutput;
var out: Text;
begin
BlockToHexStr ( ck1, k1 )
; BlockToHexStr ( ck2, k2 )
; Assign ( out, TaskName + CaseID + OutExtension )
; rewrite ( out )
; writeln ( out, OutHeader, ' ', TaskName, ' ', CaseID )
; writeln ( out, k1 )
; writeln ( out, k2 )
; Close ( out )
end; { WriteOutput }
function EqualBlock ( const b1, b2: Block ): Boolean;
var i, j: Integer;
begin
i := 0 ; j := BlockLen
; while i <> j do
if b1[i] = b2[i] then Inc ( i )
else j := i
; EqualBlock := ( i = BlockLen )
end; { EqualBlock }
const
NKeys = LongInt ( 1 ) SHL ( HexBits * MaxRelevant ); { max # keys }
Unoccupied = -1 {NKeys}; { special value for type CompressedKey }
HashModulus = 2 * NKeys;
MaxHash = HashModulus - 1; { power of 2 minus 1 }
type
HashValue = 0 .. MaxHash;
CompressedKey = LongInt {0 .. Unoccupied};
Table = array [ HashValue ] of record
cck: CompressedKey;
msg: Block;
end;
function CompressKey ( const ck: Block ): CompressedKey;
var i: Integer; cck: CompressedKey;
begin
cck := ck [ 0 ]
; for i := 1 to 3 do
cck := ( cck SHL 8 ) OR ck [ i ]
; CompressKey := cck
end; { CompressKey }
procedure UncompressKey ( cck: CompressedKey; var ck: Block );
{ pre: ck[s..BlockLen-1] = 0 }
var i: Integer;
begin
```

```
for i := 3 downto 1 do begin
ck [ i ] := cck AND $ff
; cck := cck SHR 8
end { for i }
; ck [ 0 ] := cck
end; { UncompressKey }
function HashMessage ( const mt: Block ): HashValue;
var i: Integer; hv: Cardinal;
begin
hv := mt [ 0 ]
; for i := 1 to 3 do
hv := ( hv SHL 8 ) OR mt [ i ]
; HashMessage := hv AND $1fffff
end; { HashMessage }
procedure EmptyTable ( var t: Table );
var h: HashValue;
begin
for h := 0 to MaxHash do
t [ h ] . cck := Unoccupied
end; { EmptyTable }
procedure Store ( var t: Table; const mt, ck: Block );
var hv: HashValue;
begin
hv := HashMessage ( mt )
; while t [ hv ] . cck <> Unoccupied do { occupied }
hv := ( hv + 1 ) mod HashModulus { linear hashing }
; with t [ hv ] do begin
cck := CompressKey ( ck )
; msg := mt
end { with }
end; { Store }
procedure Retrieve ( const t: Table; const mt: Block;
var found: Boolean; var ck: Block );
{ pre: ck[s..BlockLen-1]=0
post: if found then mt occurs with key ck else mt does not occur }
var hv: HashValue;
begin
hv := HashMessage ( mt )
; found := False
; repeat
with t [ hv ] do
if cck <> Unoccupied then begin
found := EqualBlock ( mt, msg )
; if found then
UncompressKey ( cck, ck )
```

```
else { linear hashing }
hv := ( hv + 1 ) mod HashModulus
end { if }
until found or ( t [ hv ] . cck = Unoccupied )
end; { Retrieve }
procedure FirstKey ( var k: Block );
var i: Integer;
begin
for i := 0 to BlockLen - 1 do
k [ i ] := 0
end; { FirstKey }
function NextKey ( var k: Block ): Boolean;
{ ret: whether k is indeed next key }
var i, j: Integer; max, step: Byte;
begin
i := -1 ; j := sBytes - 1
; max := sMaxLast
; step := sStepLast
; while i <> j do
if k [ j ] = max then begin
k [ j ] := 0
; max := 255
; step := 1
; Dec ( j )
end { then }
else { k[j] <> max } begin
Inc ( k[j], step )
; i := j
end { else }
; NextKey := 0 <= j
end; { NextKey }
var
tbl: Table; { auxiliary }
procedure EncryptPhase;
var mt1: Block;
begin
FirstKey ( ck1 )
; repeat
Encrypt ( pt, ck1, mt1 )
; Store ( tbl, mt1, ck1 )
until not NextKey ( ck1 )
end; { EncryptPhase }
procedure DecryptPhase;
var mt1: Block; found: Boolean; ck1candidate, ck2candidate: Block;
begin
```

```
FirstKey ( ck1candidate ) { for Retrieve/UncompressKey }
; FirstKey ( ck2candidate )
; repeat
Decrypt ( ct, ck2candidate, mt1 )
; Retrieve ( tbl, mt1, found, ck1candidate )
; if found then begin
; ck1 := ck1candidate
; ck2 := ck2candidate
end { if }
until found or not NextKey ( ck2candidate )
; if not found then writeln ( 'No solution found!?' )
end; { DecryptPhase }
procedure ComputeAnswer;
begin
EmptyTable ( tbl )
; EncryptPhase
; DecryptPhase
end; { ComputeAnswer }
begin
writeln ( 'Solver for Double' )
; writeln ( 'HashModulus = ', HashModulus : 10 )
; writeln ( 'SizeOf ( Table ) = ', SizeOf ( Table ) : 10 )
; if ParamCount > 0 then CaseID := ParamStr ( 1 )
; ReadInput
; ComputeAnswer
; WriteOutput
end.
```

Depot

題目 (PROBLEM)

一個芬蘭高科技公司有一個很大的長方形儲藏處 (depot)，此儲藏處有一位職員和一位經理。儲藏處的四個邊分別稱為左、上、右、下。儲藏處的空間以橫列 (row)及縱行 (column) 切割為正方形之儲存格，橫列 (row) 由上而下以 1, 2, … 編號之，縱行 (column) 由左而右也以 1, 2, … 編號之。

儲藏處提供盒子 (containers) 來存放昂貴的科技儀器，每個盒子都有一個唯一的識別號碼，一個儲存格可放置一個盒子，因為儲藏處很大，任何同時存在的盒子總數小於橫列 (row) 數目，並小於縱行 (column) 數目。盒子一旦到了儲藏處 (depot)，就會一直待在儲藏處，每當有新的盒子時，儲藏處最左上角的儲存格將是第一個被嘗試放置的儲存格。

儲藏處職員集中盒子到儲藏處的左上角，使得未來可以依盒子的識別號碼來找到想要的盒子：

假設下一個新盒子的識別號碼為 k (簡稱『k 號盒子』)，儲藏處職員首先試著由左邊開始察看第一個橫列 (row) 上所有的儲存格中的盒子，如果第一個橫列 (row) 上所有盒子的識別號碼都小於 k，則『k 號盒子』就緊靠著放在第一個橫列 (row) 上最右邊盒子的右邊儲存格，如果『l 號盒子』是第一個橫列 (row) 上，由左至右第一個識別號碼大於 k 的盒子，那麼『k 號盒子』就放在『l 號盒子』的儲存格，儲藏處職員接著以上述方法放置『l 號盒子』在下一個 (也就是第二個) 橫列 (row)上，如果儲藏處職員遇到一個空的橫列 (row)，盒子將放在此橫列 (row) 最左邊儲存格。

假設到達儲藏處之盒子的識別號碼依序為 3, 4, 6, 2, 5, 1，那麼盒子在儲藏處的放置將如下：
1 4 5
2 6
3

儲藏處經理與職員的某一段對話如下：
儲藏處經理：『5 號盒子』是否比『4 號盒子』更早到達儲藏處？
儲藏處職員：不可能。
儲藏處經理：那麼儲藏處的放置可以決定盒子到達儲藏處順序？
儲藏處職員：一般來說並不是，譬如說，以目前盒子在儲藏處的放置，盒子到達儲藏處的識別號碼可以為 3, 2, 1, 4, 6, 5 或 3, 2, 1, 6, 4, 5 或是其它 14 種順序之一。

因為儲藏處經理不希望職員看起來比較聰明，經理決定離開。請寫一個程式來幫助儲藏處經理，依照儲藏處的放置來表列出所有盒子到達儲藏處的可能順序。

輸入（INPUT）

輸入檔名為depot.in，第一行為一個整數R：放置盒子的橫列（row）數目，接下來的R行提供儲存處R個橫列（row）的資訊，由上而下：每一輸入行（line）的第一個整數K表示所對應橫列（row）的盒子數目，接下來為由左至右之盒子的識別號碼，所有盒子的識別號碼I為1至50的整數，盒子的數目N為1至13的整數。

輸出 （OUTPUT）

輸出檔名為 depot.out，輸出檔列出所有盒子到達儲藏處的識別號碼順序，每一輸出行（line）代表一個盒子到達儲藏處的識別號碼可能順序，每一輸出行（line）有 N 個整數，代表盒子到達儲藏處的識別號碼順序，盒子到達順序不應重複表列，。

輸入與輸出範例 （EXAMPLE INPUT AND OUTPUT）

Example 1: depot.in

```
3
3 1 4 5
2 2 6
```

depot.out

```
3 2 1 4 6 5
3 2 1 6 4 5
3 4 2 1 6 5
3 2 4 1 6 5
3 2 6 1 4 5
3 6 2 1 4 5
3 4 2 6 1 5
3 4 6 2 1 5
3 2 4 6 1 5
3 2 6 4 1 5
3 6 2 4 1 5
3 4 2 6 5 1
3 4 6 2 5 1
3 2 4 6 5 1
3 2 6 4 5 1
3 6 2 4 5 1
```

Example 2: depot.in

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

depot.out

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

得分(SCORING)

如果輸出檔保含不可能之盒子到達順序或沒有盒子到達順序，則該測試檔得分為 0。否則得分如下：輸出檔列出所有盒子到達儲藏處的識別號碼順序，且沒有重複表列，則該測試檔得分為 4。輸出檔列出超過一半盒子到達儲藏處的識別號碼順序，且沒有重複表列，則該測試檔得分為 2。輸出檔列出不到一半盒子到達儲藏處的識別號碼順序，或有重複表列，則該測試檔得分為 1。

Depot - example solution

```
{
prog: depot
lang: pascal
}
const
maxN=11;
type
carr=array[1..maxN] of longint;
var
i,j,k,r,n:longint;
tbl:array[1..maxN,1..maxN] of longint;
ntbl,order:array[1..maxN+1] of longint;
fi,fo:text;
procedure display;
var i,j:longint;
begin
for i:=1 to r do
begin
for j:=1 to ntbl[i] do
write(tbl[i,j],' ');
writeln;
end;
end;
procedure recurse(nd:longint);
var
i,j,k,l,v,t:longint;
str:string;
strp:^string;
c:carr;
begin
if nd=n then
begin
str:='';
for i:=n downto 1 do
begin
if i<n then write(fo,' ');
write(fo,order[i]);
end;
writeln(fo);
exit;
end;
for i:=1 to r do
begin
if ntbl[i]=0 then break;
```

```
if ntbl[i]>ntbl[i+1] then
begin
v:=tbl[i,ntbl[i]];
dec(ntbl[i]);
for j:=i-1 downto 1 do
for k:=ntbl[j] downto 1 do
if v>tbl[j,k] then
begin
t:=v;
v:=tbl[j,k];
tbl[j,k]:=t;
c[j]:=k;
break;
end;
order[nd+1]:=v;
recurse(nd+1);
for j:=1 to i-1 do
begin
t:=v;
v:=tbl[j,c[j]];
tbl[j,c[j]]:=t;
end;
inc(ntbl[i]);
tbl[i,ntbl[i]]:=v;
end;
end;
end;
begin
assign(fi,'depot.in');
assign(fo,'depot.out');
reset(fi);
read(fi,r);
n:=0;
for i:=1 to r do
begin
read(fi,ntbl[i]);
inc(n,ntbl[i]);
for j:=1 to ntbl[i] do
read(fi,tbl[i,j]);
end;
close(fi);
rewrite(fo);
ntbl[r+1]:=0;
recurse(0);
close(fo);
```

```
end.
```