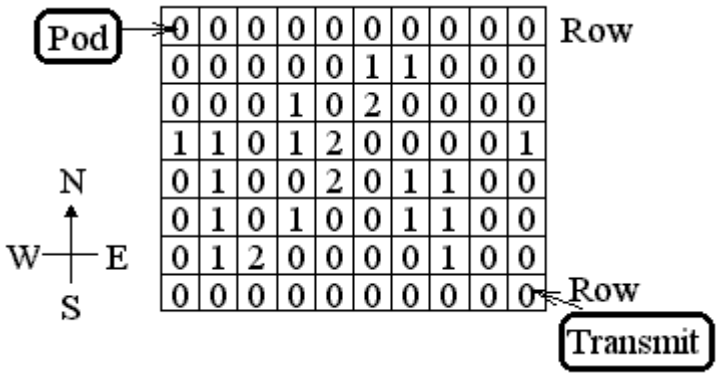
1997 年 IOI 試題與參考解答(第一天之解答由陳康本試做；第二天由龔律全試做)

第一天第一題－MARS 火星探測

未來火星探測執行任務方法如下，首先在表面某個叫 POD 的定位放下一些火星探測車(MEV)，每台 MEV 由 POD 出發，朝表面另一個叫 TRANSMITTER 的點行進，探測路途中必須採集沿路經過的岩石標本，同一岩石標本只可被一探測車採集一次，岩石標本被採集後，某他 MEV 能通過原來岩石標本所在位置。

探測車(MEV)不可經過某些不良表面(ROUGH TERRAIN)。

探測車(MEV)只能往南垂直或往東水平沿方格線(GRID PATTERN)由 POD 往 TRANSMITTER 行進，同一時間同一地方可以有兩部 MEV。



警告 若 MEV 無法由 POD 到達 TRANSMITTER，則其目前所採集所有岩石標本完全無效，且其他 MEV 也不可以再採集這些岩石標本。

TASK

根據題最後的計分公式(以採集到 TRANSMITTER 的岩石標本個數，到達 TRANSMITTER 的 MEV 數套入公式計分)找出 MEV 的行進次序，使得該函數計分最大

INPUT

火星上以 PxQ 方格表示的可行進表面，每個方格點上可以

可走方格點：以 0 代表

不良表面方格點：以 1 代表

岩石標本：以 2 代表

輸入 file 格式：

MEV 探測車車輛數目

P

Q

(x1,y1) (x2,y1) ..... (xp,y1)

(x1,y2) (x2,y2) ..... (xp,y2)

.

.

(x1,yq) (x2,yq) ..... (xp,yq)

P 和 Q 為火星表面方格長寬大小

車輛數<1000

Q 代表方格列數，每一列有 P 格

P 和 Q 不會超過 128
範例輸入

MARS.DAT	解釋
2	車輛數
P	
Q	
.	.
.	.
.	.

OUTPUT
一行一行，每行代表 MEV 車往 TRANSMITTER 的行進次序，每一行含一車輛編號和一個 0 或 1 的數字，0 代表往南，1 代表往東走一格
範例輸出

MARD.OUT	解釋
1 1	1 號車往東
1 0	1 號車往南
2 1	2 號車往東
2 0	2 號車往南

上例中，TRANSMITTER 共到達 2 輛 MEV，共收集了 3 個岩石標本，依下列公式得 5/5 = 100% 分
計分公式
公式以 TRANSMITTER 共到達 MEV 輛數及收集岩石標本總數為依據：
若輸出不合法的 MOVE，則得 0 分，不合法的 MOVE 代表移動到 ROUGH TERRAIN 或起出方格點
分數 =（TRANSMITTER 收集到的岩石標本數 + 到達 MEV 車數 − 未到車數）除以（上述可以最高數字）的百分比數
分數最高 100%，最低 0%

參考解答：
/*我所用的演算法為 Greedy Method（貪心），每次我都會找一條路能採集最多岩石，但此演算法在沒有障礙物時才能得到最佳解，在有障礙物時，得出來的解並不一定是最佳解，但我相信是一組不錯的解答，於是我採用這個方法，在五組測試後，得到 99%，有的同學和我採用一樣的演算法卻得到 100%，是因為他們考慮「東、南」的順序和我不同，我先考慮南再考慮東，他們是先考慮東再考慮南，所以得到 100%，這是 depend on test data，所以基本上，大多數的人都採用這個演算法，而這題也是得分最容易的一題*/

```
#define NDEBUG
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define Max_car 1000
#define Max 128
```

```
#define Max_size 16384
#define Rock 2
#define Gotten 4
#define Bad 1
#define Left 1
#define Up 2
#define NoDirec 4
#define Max_clock 330

clock_t start ;
int ncar, nrow, ncol, rowrock[Max], pathlen ;
char map[Max][Max], path[Max_size] ;

void Getdata(void)
{
    FILE *fi ;
    int r, c, k ;
    fi = fopen("mars.dat", "r") ;
    assert(fi) ;
    fscanf(fi, "%d", &ncar) ;
    fscanf(fi, "%d", &ncol) ;
    fscanf(fi, "%d", &nrow) ;

    for(r=0; r<nrow; r++)
        for(c=0; c<ncol; c++) {
            fscanf(fi, "%d", &k) ;
            map[r][c] = (char)k ;
        }

    fclose(fi) ;
}

void OutputPath(int car)
{
    int i, direc ;
    static FILE *fo=NULL ;

    if(!fo) {
        fo = fopen("mars.out", "w") ;
        assert(fo) ;
    }

    if(car==0) return ;
```

```
    for(i=pathlen-1; i>=0; i--) {
        if(path[i] == Left) direc = 1 ;
        else if(path[i] == Up) direc = 0 ;
        else assert(0) ;

        fprintf(fo, "%d %d\n", car, direc) ;
    }
}

void Solve(void)
{
    int getnum[Max][Max], r, c, car, nextr, nextc ;
    int mr[] = {0, 0, -1} ;
    int mc[] = {0, -1, 0} ;
    char direc[Max][Max] ;

// 每部車都跑一次
    for(car=1; car<=ncar; car++) {
        for(r=0; r<nrow; r++) {
            memset(&getnum[r][0], 0, sizeof(int)*Max) ;
            memset(&direc[r][0], 0, sizeof(char)*Max) ;
        }
        direc[0][0] = NoDirec ;
// 用兩輛迴圈算出一條可拿到最多岩石的 path，用的是 Dynamic Programming 演算法
        for(r=0; r<nrow; r++) {
            for(c=0; c<ncol; c++) {
                if(map[r][c] == Bad) continue ;
                getnum[r][c] = 0 ;
                if(r > 0) {
                    if(map[r-1][c] != Bad && direc[r-1][c]) {
                        getnum[r][c] = getnum[r-1][c] ;
                        direc[r][c] = Up ;
                    }
                }
                if(c > 0) {
                    if(getnum[r][c-1] > getnum[r][c] || !direc[r][c]) {
                        if(map[r][c-1] != Bad && direc[r][c-1]) {
                            getnum[r][c] = getnum[r][c-1] ;
                            direc[r][c] = Left ;
                        }
                    }
                }
                if(map[r][c] == Rock)
```

```
                    getnum[r][c]++ ;
            }
        }
        if(direc[nrow-1][ncol-1] != Left && direc[nrow-1][ncol-1] != Up) {
#ifndef NDEBUG
            printf("No Path\n") ;
#endif
            break ;
        }
// 回溯出這條路徑，並將它輸出到檔案裡
        pathlen=0 ;
        for(r=nrow-1, c=ncol-1; r!=0 || c!=0; ) {
            if(map[r][c] == Rock)
                map[r][c] = Gotten ;
            path[pathlen++] = direc[r][c] ;
            nextr = r+mr[direc[r][c]] ;
            nextc = c+mc[direc[r][c]] ;
            r = nextr ;
            c = nextc ;
        }
        OutputPath(car) ;
        if( (clock()-start) > Max_clock)
            exit(0) ;
    }
}

int main()
{
    start = clock() ;
    Getdata() ;
    pathlen = 0 ;
    OutputPath(0) ;
    Solve() ;
    fcloseall() ;

    return 0 ;
}
```

第一天第二題－TOXIC 有毒蟲

iShongololo 是祖魯(Zulu)族對一種長有黑色光澤的多足蟲的稱呼。
iShongololo 吃長為 L，寬為 W，高為 H 的實心立方體食物。(L,W,H 均為整數)
Task
在滿足以下限制條件(constraints)下，找出 iShongololo 所能吃最多的單位正立方體(block)個數，程式必須輸出 iShongololo 在食物內吃 blocks 及移動的過程
iShongololo 由食物外開始行動。首先吃位於(1,1,1)的 block，然後移動到此一被吃掉的空間。它在無法合法吃 blocks 且無法合法移動時停止。
Constraint
一隻 iShongololo 大小只佔一單位立方體空間。
iShongololo 一次只吃一個 block。
iShongololo 不可以移動到以前曾經所在過的任何位置(也就是說，不可以後退，也不可以移動路線相交)
iShongololo 不可以移動到實心(未吃)的部分，也不可以移動到食物(fruit)之外。
iShongololo 只可以移動或吃掉和它目前所在位置相接一個面(face)的 block 上。除此之外，被吃掉的 block 除了和目前所在位置相接的一個面外，不可以有其他面和以前吃掉過的 block 相接。
Input
程式輸入 L(長)，W(寬)，H(高)三整數
L,W,H 各佔一行。L,W,H 大小在 1 和 32 之間(可以是 32)
範例輸入：
TOXIC.DAT
2　L(長)為 2
3　W(寬)為 3
H(高)為 2
Output
一行行，每行開頭是一個英文字母。此字母為 E(代表吃)或 M(代表移動)。後面再接 3 個整數，分別代表被吃或移動到 block 所在的 L,W,H。
以下範例為上述範例輸入的一組合法解(valid solution)。請注意合法解代表在不違反限制條件下的動作過程。合法解不一定代表所吃 block 數就一定最多(optimal，即最佳解)
範例輸出(以下**也許不是**最佳解)
TOXIC.OUT
E 1 1 1 吃在(1,1,1)上的 block
M 1 1 1 移動到(1,1,1)
E 2 1 1 吃在(2,1,1)上的 block
E 1 1 2 吃在(1,1,2)上的 block
E 1 2 1

M 1 2 1
E 1 3 1
M 1 3 1
E 2 3 1
E 1 3 2
M 1 3 2

Scoring 評分方式

若 iShongololo 違反任一限制條件，則 0 分。

(總共吃的 blocks 數) 除以 (最多能吃的 blocks 數) 的百分比，為程式所得分數。

程式最多得 100 分 (亦即是 100%)。

參考解答：

```
/*我所用的演算法為 Greedy Method (貪心)，在五組測試資料中，得了 63 分，大約是中上的成績，但還有更好的方法由於時間緊迫，所以這個程式寫得並不是很好，有興趣的人，可以自己撰寫一次*/
#define NDEBUG
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#define Max 32
#define Eaten 1
#define Disable 2
// 原本想要做 k-degree 的 Greedy Method，但是有實作上的困難，於是設 Degree = 1
#define Degree 1
#define Eat_f 2
#define Move_f 1

int bx, by, bz, nowx, nowy, nowz ;
int lasteatx, lasteaty, lasteatz, final ;
char state[Max+2][Max+2][Max+2] ;
FILE *fo ;

void Getdata(void)
{
    FILE *fi ;
    fi = fopen("toxic.dat", "r") ;
    assert(fi) ;
    fscanf(fi, "%d", &bx) ;
    fscanf(fi, "%d", &by) ;
    fscanf(fi, "%d", &bz) ;
    fclose(fi) ;
}
void Eat(int x, int y, int z)
```

```
{
    fprintf(fo, "E %d %d %d\n", x, y, z) ;
    state[x][y][z] = Eaten ;
    lasteatx = x ;
    lasteaty = y ;
    lasteatz = z ;
    final = Eat_f ;
}
void Move(int x, int y, int z)
{
    fprintf(fo, "M %d %d %d\n", x, y, z) ;
    nowx = x, nowy = y, nowz = z ;
    final = Move_f ;
}
int CanEat(int x, int y, int z)
{
    int face=0 ;
    if(state[x][y][z]==Eaten) return 0 ;
    if(x>1 && state[x-1][y][z]==Eaten) face++ ;
    if(y>1 && state[x][y-1][z]==Eaten) face++;
    if(z>1 && state[x][y][z-1]==Eaten) face++;
    if(x<bx && state[x+1][y][z]==Eaten) face++;
    if(y<by && state[x][y+1][z]==Eaten) face++;
    if(z<bz && state[x][y][z+1]==Eaten) face++;
    return face <= 1 ;
}
void Eatadj(void)
{
    if(nowx > 1 && CanEat(nowx-1, nowy, nowz))
        Eat(nowx-1, nowy, nowz) ;
    if(nowy > 1 && CanEat(nowx, nowy-1, nowz))
        Eat(nowx, nowy-1, nowz) ;
    if(nowz > 1 && CanEat(nowx, nowy, nowz-1))
        Eat(nowx, nowy, nowz-1) ;
    if(nowx < bx && CanEat(nowx+1, nowy, nowz))
        Eat(nowx+1, nowy, nowz) ;
    if(nowy < by && CanEat(nowx, nowy+1, nowz))
        Eat(nowx, nowy+1, nowz) ;
    if(nowz < bz && CanEat(nowx, nowy, nowz+1))
        Eat(nowx, nowy, nowz+1) ;
}
int AdjCanEat(int x, int y, int z)
{
    int face = 0 ;
```

```
    if(x>1 && CanEat(x-1, y, z)) face++ ;
    if(y>1 && CanEat(x, y-1, z)) face++ ;
    if(z>1 && CanEat(x, y, z-1)) face++ ;
    if(x<bx && CanEat(x+1, y, z)) face++ ;
    if(y<by && CanEat(x, y+1, z)) face++ ;
    if(z<bz && CanEat(x, y, z+1)) face++ ;
    return face ;
}
int Eatnum(int x, int y, int z, int depth)
{
    int max ;
    char record = state[x][y][z] ;
    if(state[x][y][z] != Eaten) return 0 ;
    if(depth >= Degree) {
        max = AdjCanEat(x, y, z) ;
    }
    state[x][y][z] = record ;
    return max ;
}
int FindMove(int *px, int *py, int *pz)
{
    int max = 0, f ;
    if(nowx>1)
        if( (f=Eatnum(nowx-1, nowy, nowz, 1)) > max)
            max = f, *px=nowx-1, *py=nowy, *pz=nowz ;
    if(nowy>1)
        if( (f=Eatnum(nowx, nowy-1, nowz, 1)) > max)
            max = f, *px=nowx, *py=nowy-1, *pz=nowz ;
    if(nowz>1)
        if( (f=Eatnum(nowx, nowy, nowz-1, 1)) > max)
            max = f, *px=nowx, *py=nowy, *pz=nowz-1 ;
    if(nowx<bx)
        if( (f=Eatnum(nowx+1, nowy, nowz, 1)) > max)
            max = f, *px=nowx+1, *py=nowy, *pz=nowz ;
    if(nowy<by)
        if( (f=Eatnum(nowx, nowy+1, nowz, 1)) > max)
            max = f, *px=nowx, *py=nowy+1, *pz=nowz ;
    if(nowz<bz)
        if( (f=Eatnum(nowx, nowy, nowz+1, 1)) > max)
            max = f, *px=nowx, *py=nowy, *pz=nowz+1 ;
    return max > 0 ;
}
void Solve(void)
{
```

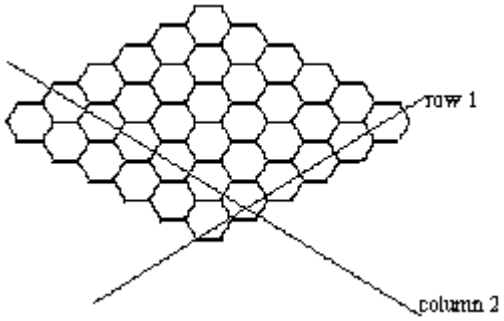
```
    int found, x, y, z ;
// initial 初始化
    fo = fopen("toxic.out", "w") ;
    nowx = nowy = nowz = 1 ;
    Eat(1, 1, 1) ;
    Move(1, 1, 1) ;
    while(1) {
// 將相鄰的全部吃掉
        Eatadj() ;
// 用 Greedy Method 找到哪一 move 是較好的
        found = FindMove(&x, &y, &z) ;
        if(!found) break ;
        Move(x, y, z) ;
    }
// 若最後一個命令是 Eat, 則再 Move 到最後一個位置
    if(final == Eat_f)
        Move(lasteatx, lasteaty, lasteatz) ;
}
int main()
{
    Getdata() ;
    Solve() ;
    fcloseall() ;
    return 0 ;
}
```

第一天第三題－Hex 六邊棋遊戲

這個遊戲的目的是第一位棋士(first player)將行 1(column1)中一個屬於他六邊棋子(Hex counter)連接到行 N(column N)中一個屬於他的六邊棋子。

六邊棋遊戲規則(Rules of Hex)

六邊棋(Hex)是兩個人在用 NxN 個六邊行(hexagons)組成的菱形(rhombus)棋盤上玩的兩人下棋遊戲。圖例中 N=6。



對奕的兩位棋士(player)分別為你的程式(your program)及評分程式集(evaluation library)。

你的程式每次都先下第一步。

兩位棋士輪流將自己的六邊棋子(Hex counters0 放在棋盤上。

每個六邊棋子(Hex counter)只能放在棋盤上空的位置上

假若兩組六邊形(hexagons)有一邊相鄰(share an edge)則這兩個六邊形相鄰(adjacent)。

屬於同一棋士的六邊棋子如果擺在相鄰的六邊形(adjacent hexagons)上，則這兩個棋子就是相連接著的(connected)。

連接性(connectivity)是 transitive 和 ommutative：換句話說：如果棋子 1 連接到棋子 2，而棋子 2 連接到棋子 3，則棋子 3 就連接到棋子 1，且棋子 1 也連接到棋子 3。

Task

你必須寫一程式來下六邊棋(Game of Hex)

先下的棋士(亦即是你的程式)的目標是將行 1(column1)中的一個屬於你的六邊棋子(hex counters)一直連到行 N(column N)中另一個屬於你的六邊棋子。

另一位棋子(亦即是評分程式集)的目標則是將列 1(row 1)中的一個屬於他的六邊棋子一直連到列 N(column N)中另一個屬於他的六邊棋子。

假如你的程式採取最佳下法，則每次都會贏

Input and Output

你的程式**不可**讀入或寫出至任何檔案。

你的程式**不可**從鍵盤輸入任何資料，也不可輸出任何結果(output)至螢幕(screen)。

你的程式的輸入是由 HexLib 程式集內的函數(function)來提供。這個程式集會自動產生一個檔案名為 HEX.OUT 的輸出檔，你可以不必理會這個檔的內容。

在遊戲剛開始時，你的程式將被給定一個盤面，上面可能會已有一些六邊棋子(Hex counters)，但先下棋者仍然可以贏棋。你的程式必須用 GetMax 及 LookAtBoard 函數來了解

棋盤盤面狀態。
在剛開始時，盤面上屬於你的程式的六滿棋子的數和評分程式集的六邊棋子的數目**相等**。
Constraint
棋盤大小一定介於 1 跟 20 之間(包括 1 和 20)
你的程式最多有可能需下到 200 步才結束。整個程式運作須在 40 秒內完成，評分程式集(evaluation library)運作的時間將一定不會超過 20 秒。
Library
你必須將 HexLib 這個程式集 link 到你的程式碼內。在題目的目錄中將有範例檔案教你如何做到。這些範例檔案分別是 TESTHEX.CPP , TEXTHEX.C , TEXTHEX.PAS , TEXTHEX.BAS
存在於 HexLib 的函數分別為 (先 Pascal, C/C++, 再 Basic)
function LookAtBoard(row, column : integer) : integer ;
int LookAtBoard(int row, int column) ;
declare function LookAtBoard cdecl(byval x as integer, byval y as integer) ;
傳回值：
- 1 若 row<1 或 row>N 或 column<1 或 column>N
若此棋盤位置上無任何六邊棋子
若此棋盤位置上有一個你的六邊棋子(player 1)
若此棋盤位置上有一個評分程式集(evaluation library)的六邊棋子(player 2)
procedure PutHex(row, column : integer) ;
void PutHex(int row, int column) ;
declare sub PutHex cdecl(byval x as integer, byval y as integer)
假若指定的棋盤位置是空的，則將你的一個六邊棋子放到指定的位置上去。
function GameIsOver: integer ;
int GameIsOver(void) ;
declare function GameIsOver cdecl()
傳回底下任一整數：
遊戲尚未結束
棋盤上每一個位置都已擺滿六邊棋子
你的程式贏了
評分程式贏了
procedure MakeLibMove ;
void MakeLibMove(void) ;
declare sub MakeLibMove cdecl()
讓評分程式集(evaluation library)計算下一步該下的位置並將它的一個六邊棋子放在該位置上。棋盤盤面的改變可以用 LookAtBoard 及其他的函數來查看。
function GetRow : integer ;
int GetRow(void) ;
declare function GetRow cdecl()

傳回最近評分程式集(evaluation library)所下的六邊棋子的列數(row)，或傳回-1 若尚未下。此函數將一直傳回相同的行數值一直到你的程式呼叫 MakeLibMove 為止。

function GetColumn : integer ;

int GetColumn (void) ;

declare function GetColumn cdecl()

傳回最近評分程式集(evaluation library)所下的六邊棋子的行數(column)，或傳回-1 若尚未下。此函數將一直傳回相同的行數值一直到你的程式呼叫 MakeLibMove 為止。

function GetMax : integer ;

int GetMax(void) ;

declare function GetMaxcdecl()

傳回棋盤大小，N。

Scoring

假如你的程式下贏的話，則得滿分。

假如你的程式下輸的話，則得滿分之 20%。

假如你的程式在遊戲結束前結束或執行時間超過時間限制，則得 0 分。

參考解答：

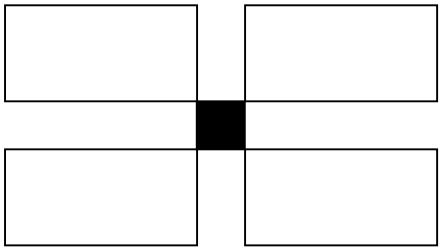
```
/*這個程式用的演算法是 Greedy Method，在五組測試後，所得分數為 68 分，是 IOI'97 中
本題最高的得分，總計是贏三盤輸兩盤我是從 column 1 找起，找到所能連接到最大的 column
值，就選擇這一步走下去，這樣可以很快的從 column 1 找到 column N*/
#define NDEBUG
#ifndef NDEBUG
#include <conio.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include "hexc.h"
#define MAX 21
#define Disable (8)
#define Human 1
#define Comp 2
int max, mr[] = {1, 1, 0, 0, -1, -1}, mc[] = {0, 1, -1, 1, -1, 0} ;
int canchoose[MAX][MAX] ;
char board[MAX][MAX] ;
/*這個函式在此程式中扮演了極重要的角色，是找出從 column 1 所能連接
到的格子，也就是下一步的「候補格子」*/
void Updatecanchoose(void)
{
    int r, c, k ;
    for(r=1; r<=max; r++)
        if(board[r][1]!=Comp) canchoose[r][1] = 1 ;
    for(r=2; r<=max; r++)
        for(c=1; c<=max; c++) {
```

```
            if(board[r][c] == Comp) continue ;
            for(k=0; k<6; k++)
                if(board[r+mr[k]][c+mc[k]]==Human &&
                    canchoose[r+mr[k]][c+mc[k]]) {
                    canchoose[r][c] = 1 ;
                    break ;
                }
        }
}
void InitBoard(void)
{
    int r, c ;
    max = GetMax() ;
    for(r=0; r<=max+1; r++)
        board[r][0] = board[r][max+1] = Disable ;
    for(c=0; c<=max+1; c++)
        board[0][c] = board[max+1][c] = Disable ;
    for(r=1; r<=max; r++)
        for(c=1; c<=max; c++)
            board[r][c] = LookAtBoard(r, c) ;
    Updatecanchoose() ;
}
void MaxcolChoose(int *prow, int *pcol)
{
    int r, c ;
    for(c=max; c>=1; c--)
        for(r=max; r>=1; r--)
            if(canchoose[r][c] && board[r][c]==NULL) {
                *prow = r ;
                *pcol = c ;
                return ;
            }
}
void FindMove(int *prow, int *pcol)
{
// 找尋 column 最大值
    MaxcolChoose(prow, pcol) ;
    board[*prow][*pcol] = Human ;
    Updatecanchoose() ;
}
#ifndef NDEBUG
void display(void)
{
    int row, col, max, l;
```

```
    max = GetMax ();
    for (row=1; row<=max; row++)
    {
        for (col=1; col<row; col++)
            cprintf (" ");
        for (col=1; col<=max; col++)
         { l = LookAtBoard (max+1-row, col);
            textcolor (l == 0 ? 15 : l*3-2);
            cprintf ("%d ",l);
         }
        textcolor (WHITE);
        cprintf ("\r\n");
    }
}
#endif
int main()
{
    int gameover, row, col ;
// 初始化
    InitBoard() ;
#ifndef NDEBUG
    display() ;
#endif
    do {
        // find a good move
        FindMove(&row, &col) ;
        PutHex(row, col) ;
        MakeLibMove() ;
        board[GetRow()][GetColumn()] = Comp ;
#ifndef NDEBUG
        display() ;
        getch() ;
#endif
        gameover = GameIsOver() ;
    } while(gameover == 0) ;
    return 0 ;
}
```

第二天第一題－Map Labelling

你是地籍測量員助理，你的工作是必須將城市的名稱寫到新的地圖上。
每一張地圖是用 1000*1000 格的矩形代表。每一個城市在地圖上佔用一格的空間，而城市名稱比須放在用格子所組成的長方形內，我們稱此長方形為標記(Labels)。



圖一：一個城市的四個可能標記的位置
標記擺放的位置必須符合下列條件：

1. 一個城市的標記必須出現在四個可能的位置上的其中一個，如圖一所示。
2. 標記不能相互重疊。
3. 標記不能覆蓋過任何城市
4. 標記必須至於整張地圖內。

每一個標記包含城市名稱的每一個字母加上一個空白。
每一個城市名稱的字母的寬度(width)及高度(height)將被輸入，而空白的大小與這些字母一致。

4		L	a	n	g	A	○								
3	●							●							
2									P	a	a	r	l	○	
1							●								
0	○	C	e	r	e	S									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

圖二
●代表城市的位置
○代表一個空白
地圖的原點格(0,0)是在左下角，在圖二的範例中，Langa 城是在(0,3)，Ceres 城在(6,1)，而 Paarl 城在(7,3)的位置上，所有的標記的擺設都是合法的，但這不是唯一合法的擺設方式。
Task
你的程式必須讀入每一個城市位置。城市名稱字母的寬高，及城市名稱，然後在不違反上述標記擺設條件下，儘可能將越多的城市標記(label)擺在地圖上。然後將這些城市標記位置輸出
Input
輸入檔第一行是城市數 N。之後每一行記錄著一個城市的資訊，包括：

城市位置：(水平位置 X，垂直位置 Y)
城市名稱字母大小：字母寬度 W，字母高度 H。
城市名稱
每個城市名稱都只有一個字(Word)，且最多有 1000 個城市。
城市名稱最長不超過 200 個字母(letters)。

Sample Input

MAPS.DAT	Explanation
3 0 3 1 1 Langa 6 1 1 1 Ceres 7 3 1 2 Paarl	N=3 X=0,Y=3,W=1,H=1

Output
你的程式須輸出 N 行，每一行有一個城市標記的左上角的水平及垂直方位。水平及垂直方位中間用一個空白分開。假若有城市無法被合法的擺設到地圖上，則輸出-1 -1。所有輸出行的順序必須跟輸入檔內城市的順序一致。

Sample Output

MAPS.OUT	Explanation
1 4 0 0 8 2	Langa's label is at (1,4) Ceres's label is at (0,0) Paarl's label is at (8,2)

Scoring
每一組測試資料：
分數是你所有合法擺設到地圖上的城市標記數除以最佳數之百分比。
最低 0%，最高 100%。
如果你有任何標記違反了任一擺設限制條件，則得 0 分。
如果標記與城市位置不符合，則得 0 分。

參考解答：

```
#include<fstream.h>
#include<assert.h>
#include<string.h>
#include<stdlib.h>
//////////////////////////////////////////////////////////
//  maps.cpp          wrote on 12/04/97 in Cape Town, SA   //
//                    comments added on 2/07/98           //
//                    by Lu-chuan Kung                    //
//  說明：                                                 //
//       本城市的目的在輸入資料中給的許多城市中，選出       //
```

```
//  最多個不相重疊的城市來標記。本程式所採用的演算法      //
//  為「貪心演算法」，先依照城市標記由小排到大，然後      //
//  一個一個城市試試看，若放在地圖上不會和之前所選的      //
//  城市或標記重疊，則選擇之。                          //
//  （此解法並不能解出最佳解，但可以求出不錯的解）      //
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
// 常數定義
#define MAX_N (1000)            // 城市數目的最大值
#define MAX_LEN (201)           // 城市名稱的最大長度
#define MAX_CORD (1000)         // 坐標的最大值

////////////////////////////////////////////////////////////
// 定義型態
typedef int (*FCMP)(const void *,const void *);
struct POINT        // 定義「點」型態為兩個整數
{
    int x,y;        // x 為 x 坐標 , y 為 y 坐標
};
struct SortNode     // 定義用來排序的資料結構
{
    int index;      // 城市的編號
    long area;      // 面積
};
struct OutNode      // 定義用在輸出時排序的資料結構
{
    int index;      // 城市的編號
    int x,y;        // 坐標
};
////////////////////////////////////////////////////////////
// 全域變數
char huge map[MAX_CORD][MAX_CORD/8];// 用 bitmap 來確定座標點是否用過
int nCity;              // 輸入資料的城市個數
SortNode citys[MAX_N];  // 用來依照面積大小排序的陣列
OutNode outCity[MAX_N]; // 用來依照名稱排序的陣列
int nameLen[MAX_N];     // 城市名稱的長度
int xOfCity[MAX_N];     // 城市標記的 x 坐標
int yOfCity[MAX_N];     // 城市標記的 y 坐標
int wOfCity[MAX_N];     // 城市標記的寬度
int hOfCity[MAX_N];     // 城市標記的高度
// read 函式傳回 座標 (x,y) 是否已經被占據
inline int Read(int x,int y)
{
```

```
    return map[x][y/8] & (1<<(y%8)) ;
}
// write 函式將 座標 (x,y) 設定為已占據
inline void Write(int x,int y)
{
    map[x][y/8] |= (1<<(y%8)) ;
}
// 依 面積 排序的函式 (給 quicksort 呼叫)
int SortFunc(const SortNode *pa,const SortNode *pb)
{
    long x = pa->area - pb->area;
    if( x < 0L )
        return -1;
    else if( x > 0L)
        return 1;
    else
        return 0;
}
// 依 編號 排序的函式 (恢復輸入時的順序)
int SortByIndex(const OutNode *pa,const OutNode *pb)
{
    return pa->index - pb->index;
}
// 由輸入檔讀入城市資料
void ReadInput(void)
{
    ifstream input("maps.dat");
    assert(input);
    char str[MAX_LEN];
    int i,w,h;
    input >> nCity;     // 讀入城市個數
    for(i=0;i<nCity;i++) {
        citys[i].index = i;
        input >> xOfCity[i] >> yOfCity[i];
        Write( xOfCity[i], yOfCity[i] ); // 設定此座標為已被占據
        input >> w >> h ;
        wOfCity[i] = w;
        hOfCity[i] = h;
        input >> str ;
        nameLen[i] = strlen(str)+1;
        // 面積為 w(寬度)xh(高度)xlength(字數)
        citys[i].area = (long)w * h * (nameLen[i]) ;
    }
    // 依面積排序
```

```
    qsort(citys,nCity,sizeof(SortNode),(FCMP)SortFunc);
}
// 此函測試標記的 4 個方向是否可放，若可以，則傳回標記的 maxX 與 maxy
// 若失敗，則傳回 -1
int CanPutAndPutDir(int cityno,int &minx,int &maxy)
{
    POINT startPoint[4];
    POINT endPoint[4];
    int i,x,y,sucess;
    // top left 左上角
    startPoint[0].x = xOfCity[cityno] - wOfCity[cityno] * nameLen[cityno];
    startPoint[0].y = yOfCity[cityno] + 1;
    endPoint[0].x   = xOfCity[cityno] - 1;
    endPoint[0].y   = yOfCity[cityno] + hOfCity[cityno];
    // bottom left 左下角
    startPoint[1].x = xOfCity[cityno] - wOfCity[cityno] * nameLen[cityno];
    startPoint[1].y = yOfCity[cityno] - hOfCity[cityno];
    endPoint[1].x   = xOfCity[cityno] - 1;
    endPoint[1].y   = yOfCity[cityno] - 1;
    // top right 右上角
    startPoint[2].x = xOfCity[cityno] + 1;
    startPoint[2].y = yOfCity[cityno] + 1;
    endPoint[2].x   = xOfCity[cityno] + wOfCity[cityno] * nameLen[cityno];
    endPoint[2].y   = yOfCity[cityno] + hOfCity[cityno];
    // bottom right 右下角
    startPoint[3].x = xOfCity[cityno] + 1;
    startPoint[3].y = yOfCity[cityno] - hOfCity[cityno];
    endPoint[3].x   = xOfCity[cityno] + wOfCity[cityno] * nameLen[cityno];
    endPoint[3].y   = yOfCity[cityno] - 1;
    // 測試四個方向
    for(i=0;i<4;i++) {
        // 若超過邊界，則試下一種方向
        if( startPoint[i].x < 0 || startPoint[i].x >= MAX_N ||
            startPoint[i].y < 0 || startPoint[i].y >= MAX_N )
            continue;
        if( endPoint[i].x < 0 || endPoint[i].x >= MAX_N ||
            endPoint[i].y < 0 || endPoint[i].y >= MAX_N )
            continue;
        sucess = 1;
        // 測試在標記內的每一個坐標是否已被占據
        for( x=startPoint[i].x ; x <=endPoint[i].x ;x++ ) {
            for(y=startPoint[i].y ; y<=endPoint[i].y ; y++) {
                if( Read(x,y) ) {   // 若已被占據，則離開 for 迴圈
                    sucess=0;
```

```
                    goto ExitFor;
                }
            }
        }
ExitFor:
        if( sucess ) {
            // mark on the map 在地圖上標記的範圍內登記為被占據
            for( x=startPoint[i].x ; x <=endPoint[i].x ;x++ ) {
                for(y=startPoint[i].y ; y<=endPoint[i].y ; y++) {
                    Write(x,y);
                }
            }
            // 傳回標記的左上角座標
            minx = startPoint[i].x;
            maxy = endPoint[i].y;
            return i;
        }
    }
    return -1;
}
// 放置地圖
void PutLabel(void)
{
    int i,j;
    int z;
    int x,y;
    // 初始化 outCity 陣列
    for(i=0;i<nCity;i++) {
        outCity[i].x = -1;
        outCity[i].y = -1;
    }
    // 依面積由小到大的順序 try try 看
    for(i=0;i<nCity;i++) {
        j = citys[i].index ;
        outCity[i].index = j;
        z = CanPutAndPutDir(j,x,y);
        if( z >= 0 ) {      // 可以放得進去
            outCity[i].x = x;
            outCity[i].y = y;
        }
    }
    // 依照輸入時的順序排序
    qsort( outCity,nCity,sizeof(OutNode),(FCMP)SortByIndex);
}
```

```
// 輸出答案
void Output(void)
{
    ofstream output("maps.out");
    int i;
    for(i=0;i<nCity;i++) {
        output << outCity[i].x << " " << outCity[i].y << endl;
    }
    return;
}
// 主函式
int main(void)
{
    ReadInput();
    PutLabel();
    Output();
    return 0;
}
```

第二天第二題－Image:Character Recognition

寫一個程式來做文字辨識
Details:
每一個字母樣本(ideal character image)是由 20 行，每行 20 個數字所組成，且每一個數字皆是'0'或'1'。圖 1a 是一個字母樣本的範例。
在 FONT.DAT 檔內有 27 個依以下順序排列的字母樣本：
○abcdefghijklmnopqrstuvwxyz
在此'○'代表空白
在 IMAGE.DAT 檔有一或更多可能扭曲的字母影像，每一個影像的可能扭曲方式如下：
1.(最多)一行重複出現(duplicated line)且緊跟在原始行之後。
2.(最多)一行被刪除(missing line)。
3.有些'0'被改成了'1'。
4.有些'1'被改成了'0'。
沒有任何影像會同時有上述第一項及第二項同時發生，且在測試資料中，每張影像最多有 30%的 0-1 對變。
如有重複出現行時(duplicated line)，則重複出現行及原始行都可能有 0-1 對變，且對變盤也可能不一樣。
Task
寫個程式來辨識一或更多的存在 IMAGE.DAT 檔的字母影像，而原始字母樣本則是存在 FONT.DAT 檔。
辨識時，請選擇需要最少的 0-1 對變扭曲方式的字母樣本當作辨識出之字母。但如果你假設兩行之間有一行是另一行之重複行時，只有 0-1 對變較少那一行須要被計算進去。所有測試資料的字母影像一定可以被一個一個獨立的辨識出來，假若你的程式寫的好的話。而且每組測試資料一定有一組最佳解。
正確的解應該會使用到 IMAGE.DAT 檔內所有資料行。
Input
兩個輸入檔的第一行是一正整數 N (19<=N<=1200)
代表之後的行數如底下之範例：
N
(digit1)(digit2)(digit3).....(digit20)
(digit1)(digit2)(digit3).....(digit20)
.....
每一行都有 20 個數字，而這些 0 與 1 數字之間沒有空白。
FONT.DAT 含有字母樣本，且一定有 541 行。
但每一組測試資料可能用不同的 FONT.DAT 檔
Output
你的程式必須產生一個命名為 IMAGE.OUT 的輸出檔。而輸出檔內只有一行辨識結果的字串(string)。輸出格式就是一行的 ASCII 文字字串。輸出檔字串內不應有任何間隔(字與字之間

不可以有間隔)，假若你的鄉辨識不出某個字母，就在正確的位置輸出'?'。
注意：上述輸出格式規定與競賽規則中所規定一"輸出需有間隔符號"不符。請遵守此新輸出格式規定。

Scoring

分數是所有正確辨識出的字母的百分比。

Sample files:

Incomplete sample showing the beginning of FONT.DAT(space and 'a')	Sample IMAGE.DAT showing an 'a' corrupted
FONT.DAT	IMAGE.DAT
00000000000000000000 00000111000000000000 00001111111011000000 00001111111001100000 00001110001100100000 00001100001100010000 00001100000100010000	19 00000000000000000000 00000000000000000000 00000000000000000000 00000111000000000000 00100111011011000000 00001111111001100000 00001110001100100000 00001100001100010000 00001100000100010000 00000100000100010000 00000010000000110000 00001111011111110000 00001111111111110000 00001111111111000000 00001000010000000000 00000000000000000000 00000000000001000000 00000000000000000000 00000000000000000000

00000100000100010000 00000010000000110000 00000001000001110000 00001111111111110000 00001111111111110000 00001111111111000000 00001000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000	
Figure 1a	Figure 1b

Sample Output

IMAGE.OUT	Explanation
a	Recognised the single character ‘a’

參考解答：

```
#include<fstream.h>
#include<stdlib.h>
#include<assert.h>
#include<string.h>
#define     NUM_CHAR    (27)
#define     N_DIGIT     (20)
#define     MAX_LINE    (1200)
#define     UNKNOWN     (-1)
#define     ERR_MAX     (10000)
enum { CORRUPT , NORMAL , DUPLICATE } ;
//-------------------------------------------------------
// global data 全域性資料
const char matchStr[] = " abcdefghijklmnopqrstuvwxyz";
//  store font data 儲存字型資料
char font[NUM_CHAR][N_DIGIT][N_DIGIT+1];
//  store input data 儲存輸入資料
char data[MAX_LINE][N_DIGIT+1];
//  store the number of lines 儲存輸入資料的行數
int nLine;
//  store the minimum error until line No.n
//  儲存到 n 行為止的最小錯誤值
int nError[MAX_LINE+1];
```

```
//  store the previous lineno to obtain the optimal value
//  CORRUPT or NORMAL or DUPLICATE , will be used to backtrace the
//  optimal answer
//  儲存到 n 行為止的最小錯誤值是從那一行走過來的，可能是
//  少一行、正常或多一行)，用來印出最後的答案
int prev[MAX_LINE+1];
// store the optimal matched char that lead to line n
// 儲存從上一行到這一行的最小錯誤值所對應的字元辨識結果
char chr[MAX_LINE+1];
//-------------------------------------------------------
// read font data into memory  此函式讀入字型
void ReadFont(void)
{
    int i,j,k;
    // open font file  開字型檔以供輸入
    ifstream input("font.dat");
    // check if file has been opened  檢查是否開檔成功
    assert(input);
    // ignore the first '540'  去掉第一行的 540
    input >> i;
    // read 27 characters  讀入 27 個字型資料
    for(i=0;i<NUM_CHAR;i++) {
        for(j=0;j<N_DIGIT;j++) {
            for(k=0;k<N_DIGIT;k++) {
                input >> font[i][j][k];
            }
        }
    }
}
//-------------------------------------------------------
// read input data into memory  此函式讀入輸入資料
void ReadData(void)
{
    int i,j;
    // open input file  開輸入檔以供輸入
    ifstream input("image.dat");
    // check if file has been opened  檢查是否開檔成功
    assert(input);
    // read the number of lines  讀入輸入資料的行數
    input >> nLine;
    // read nLine line data  讀入 nLine 行的資料
    for(i=0;i<nLine;i++) {
        for(j=0;j<N_DIGIT;j++) {
            input >> data[i][j];
```

```
        }
    }
}
// 求對某個字型的最小 corrupt error
int MinCorruptErrorOfFont(int fontno,int startLine,int maxErr)
{
    int i,j,k,err;
    int minErr = ERR_MAX;
// assume line i in font 'fontno' was corrupt 假定字型的第 i 行消失
// ignore its error value 則勿略該行的誤差值
    for(i=0;i<N_DIGIT;i++) {
        err = 0;
        for(j=0;j<N_DIGIT-1;j++) {
            if( j < i ) {
                for(k=0;k<N_DIGIT;k++)
                    if( font[fontno][j][k] != data[startLine+j][k] )
                        err++;
            }
            else {
                for(k=0;k<N_DIGIT;k++)
                    if( font[fontno][j+1][k] != data[startLine+j][k] )
                        err++;
            }
            // 如果 err 已經大於 maxErr ，則捨棄此誤差值
            if( err > maxErr )
                break;
        }
        if( err < minErr )
            minErr = err;
    }
    return minErr;
}
// 求出 corrupt 中 error 最小的字型，及其誤差值
int CorruptError(int startLine,int *pFontNo)
{
    int i,err;
    int minErr = ERR_MAX ;
    for(i=0;i<NUM_CHAR;i++) {
        err = MinCorruptErrorOfFont(i,startLine,minErr);
        if( err < minErr ) {
            minErr = err;
            *pFontNo = i;
        }
    }
```

```
    return minErr;
}
// 求沒有 corrupt 也沒有 duplicate 之下的最小誤差
int NormalError(int startLine,int *pFontNo)
{
    int i,j,k;
    int minErr = ERR_MAX ,sumErr;
    // for each font, count its error value, and return the minimum
    // 對於每一個字型，計算它的誤差值，並且傳回其中最小的
    for(i=0;i<NUM_CHAR;i++) {
        sumErr = 0;
        for(j=0;j<N_DIGIT;j++)  {
            for(k=0;k<N_DIGIT;k++) {
                if( font[i][j][k] != data[startLine+j][k] )
                    sumErr++;
            }
            // 若 誤差和 已經大於 minErr ，則捨棄之
            if( sumErr > minErr )
                break;
        }
        if( sumErr < minErr ) {
            minErr = sumErr ;
            *pFontNo = i;
        }
    }
    return minErr;
}
// 求出對於某個字型的最小 duplicate 誤差值
int MinDuplicateErrorOfFont(int fontno,int startLine,int maxErr)
{
    int i,j,k,err;
    int minErr = ERR_MAX;
    // assume line startLine + i in input data was duplicated
    // 假定 data 中第 startLine+i 行 是多餘的，捨棄其誤差值
    for(i=0;i<N_DIGIT+1;i++) {
        err = 0;
        for(j=0;j<N_DIGIT;j++) {
            if( j < i ) {
                for(k=0;k<N_DIGIT;k++)
                    if( font[fontno][j][k] != data[startLine+j][k] )
                        err++;
            }
            else {
                for(k=0;k<N_DIGIT;k++)
```

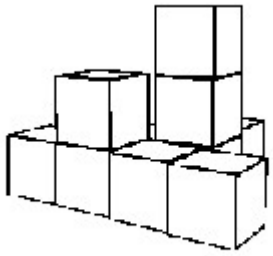
```
                    if( font[fontno][j][k] != data[startLine+j+1][k] )
                        err++;
                }
                // 假如誤差值已經超過了 maxErr ，則捨棄之
                if( err > maxErr )
                    break;
            }
            if( err < minErr ) {
                minErr = err;
            }
        }
        return minErr;
}
// 求在 duplicate 的情形下，誤差最小的字型
int DuplicateError(int startLine,int *pFontNo)
{
        int i,err;
        int minErr = ERR_MAX;
        for(i=0;i<NUM_CHAR;i++) {
            err = MinDuplicateErrorOfFont(i,startLine,minErr);
            if( err < minErr ) {
                minErr = err;
                *pFontNo = i;
            }
        }
        return minErr;
}
//-----------------------------------------------------
// compute the minimum error until lineno and fill in nError[]
// (計算到 lineno 此行為止的最小錯誤值並將之儲存於陣列 nError[]
int ComputeError(int lineno)
{
        int minErr,err,i,fontno;
        // if the minimum error has been computed, return it
        // (如果最小錯誤值已經算過了，則傳回儲存在陣列中的值)
        if( nError[lineno] != UNKNOWN ) {
            return nError[lineno];
        }
        // the minimum error of line 'lineno' can be obtain from line
        // 'lineno-19' or 'lineno-20' or 'lineno-21' add the error of
        // the font that between these lines, then pick the minimum
        // of these three as the minimum error of line 'lineno'
        //到 lineno 這一行為止的最小錯誤值可以從 lineno-19 行或 ineno-20 行
        // 或 lineno-21 行，這三者中再加上之間的那個字型的 error 中最小的
```

```
    for(minErr=ERR_MAX,i=0;i<=2;i++) {
        if( lineno - i - 19 < 0 )
            continue ;
        err = ComputeError( lineno - i - 19) ;
        // speed up
        if( err >= minErr )
            continue;
        switch(i) {
        case CORRUPT:
            err += CorruptError(lineno-19,&fontno); break;
        case NORMAL:
            err += NormalError(lineno-20,&fontno); break;
        case DUPLICATE:
            err += DuplicateError(lineno-21,&fontno); break;
        }
        if( err < minErr ) { // 選擇三者之中最小的
            minErr = err;
            prev[lineno] = lineno - i - 19;
            chr[lineno] = matchStr[fontno];  // 記錄比對的結果
        }
    }
    nError[lineno] = minErr;
    return minErr;
}
// 進行初始化的工作
void Init(void)
{
    int i;
    nError[0] = 0; // 只有第 0 行可做為起點，其起始誤差為 0
    for(i=1;i<19;i++)
        nError[i] = ERR_MAX ; // 1~18 不可做為起點，設定其誤差為 max
    for(i=19;i<=nLine;i++)
        nError[i] = UNKNOWN;  // 之後的誤差值為所欲求的
}
// 輸出答案的函式
void OutputAnswer(void)
{
    int lineno,count;
    char ansStr[MAX_LINE/(N_DIGIT-1)+1]={0};
    ofstream output("image.out");
    // 從 lineno = nLine 開始推回去 第 0 行
    for(lineno=nLine,count=0;lineno!=0;lineno=prev[lineno])
        ansStr[count++] = chr[lineno];
    // 把答案反轉
```

```
    _strrev(ansStr);
    assert(output);
    // 印出答案
    output << ansStr << endl;
}
// 主函式
int main(void)
{
    ReadFont();
    ReadData();
    Init();
    ComputeError(nLine);
    OutputAnswer();
    return 0;
}
```

第二天第三題－Stack:Stacking Containers

Neptune 貨運公司有一貨櫃儲存倉庫(depot)。倉庫接收貨櫃後，儲存一段時間後再移出。
貨櫃只在每小時整點鐘時(on the hour)移入倉庫。並在倉庫儲存正整數個小時後移出。當貨櫃移入時，貨櫃就附有文件，標明它期望(expected)被移出的時間。貨櫃被真正移出的時間和期望移出的時間可能提前或延後，但最多不超過 5 個小時。
在本題中，時間以遞增正整數表示，最大值不超過 150。
倉庫的儲貨空間(storage space)之上有一吊車(crane)。可以把貨櫃移出或移入倉庫。也可以把倉庫內的貨櫃自一點移動到另一點，吊車在儲貨空間上移動，不影響儲貨空間。
Task
寫一個好的吊車操作程式，使得使用吊車的次數為最少。
倉庫儲貨空間是矩形立方體，長為 X，寬為 Y，高為 Z。長寬高會被告知。
每個貨櫃是 1*1*1 的正立方實心體(cube)。貨櫃可以堆在儲貨空間的地板上，或是疊在一堆貨櫃的正上面。吊車只可以移動一堆貨櫃最上面的一個。



一個貨櫃若由一點用吊車移動到任何一點，不管是移入，移出，或倉庫內移動，都算成一次操作移動(move)。吊車在貨櫃來到(arrival)和移出之間操作。假設吊車操作移動不花任何時間。
當儲貨空間滿了後，你的程式必須拒絕接受移入任何貨櫃的要求。當倉庫快滿時，你的程式也許有可能變得比較沒有效率或根本無法正常操作。你的程式在任何時候都可以拒絕接受移入新貨櫃的要求。
Input
你的程式必須和一個模擬程式模組(simulation module)相連接(interact)，該模組會提出貨櫃移動要求(action)，及必需訊息(messages)。儲貨空間剛開始時完全沒有貨櫃。
你的程式寫作時提供的測試程式會傳回一小組有意義，從頭到尾的測試資料。
每個貨櫃以一唯一正整數編號。
你的程式在任何時候都可以呼叫下列函數(function)
int GetX(); 傳回儲貨空間的長度(整數值)
int GetY(); 傳回儲貨空間的長度(整數值)
int GetZ(); 傳回儲貨空間的長度(整數值)
X,Y,Z 值不超過 32
下列函數(function)傳回移動要求(action)~即要求某貨櫃移出或移入的必須資訊。貨櫃只在

每小時正點鐘會到達而要求移入。貨櫃移出則絕對不在每小時的正點鐘上。
在此，假設，每次一有新貨櫃要求移入，就代表剛好又經過一小時了。
int GetNextContainer() ;
傳回一個正整數，代表下一個要求被移入或移出的貨櫃編號，若已無任何貨櫃要求移入或移出，則傳回 0，代表程式必須正常結束，不管當時倉庫內是否還有任何貨櫃。
int GetNextAction() ;
傳回一個整數，1 代表移入新貨櫃。2 代表移出貨櫃。
int GetNextStorageTime() ;
傳回以小時計算的貨櫃預期移出的時間(由程式開始執行算起)。這個值是為了讓你的鄉預先規劃。真正貨櫃移出時間會有少許差異。但最多相差加減不超過 5 小時。當 GetNextAction 傳回 1 後，本函數才會傳回有意義的值。
上述幾個函數呼叫的次序並不重要。
接連呼叫 GetNextContainer，GetNextAction 及 GetNextStorageTime 永遠傳回同一貨櫃的移動要求資訊，直到該貨櫃被你的程式拒絕，依要求移入或移出為止。在此之後上述函數傳回下一個貨櫃移動要求的資訊。
Output
當你的程式獲得貨櫃移動要求的資訊，計算後可以使用下列函數來操作吊車
int MoveContainer(int x1,int y1, int x2, int y2) ;
把堆放在地板位置 x1,y1 那一堆貨櫃的最上面一個移到位置 x2,y2 那一堆貨櫃的最上面，傳回 1 若這個操作是合法的，傳回 0 若捆作不合法(亦即是不可能發生)。
void RefuseContainer() ;
回報不願意接收新到貨櫃。
void StoreArrivingContainer(int x, int y) ;
把新到貨櫃放到地板位置 x,y 那一堆貨櫃的最上面。
void RemoveContainer(int x, int y) ;
把放在地板位置 x,y 那一堆貨櫃的最上面一個移出倉庫。
若你的程式無法針對一個移動要求(action)執行上述任一函數時，則必須正常結束。
不合法的操作會被忽略，對模擬狀態及你的得分都不產生任何影響。
你的程式不可以產生任何輸出檔案。你呼叫的函數自動會產生記錄檔，並用以評分。
Sequencing 操作次序
你的程式應依得到的每一次貨櫃移動要求(action)決定如何操作吊車移入，移出(並有可能內部移動)貨櫃。也可以決定拒絕接收新到貨櫃。
Library
你的程式必須和一個叫 StackLib 的程式庫集相接(link)，標準 C 及 C++程式庫集已經包括那些多加的函數，只要程式加上適當的 header file 即可。
範例程式叫 TESTSTK.CPP 或 TESTSTK.C
Scoring
你的程式會被測試數個回合，每一回合會有一組最佳解。依此用下列方式評分

先計算你程式操作吊車的總次數
對每一拒絕接收移入的貨櫃，罰多加五次操作(move)
你的程式若提早正常結束則對每一未移入及移出的貨櫃，罰多加五次操作
分數計算以和最佳解相差的百分比決定。
若捆作次數是最佳解的兩倍以上，則得 0 分。
程式操作次數等於最佳解，得 100 分，在此和兩倍最佳解之間則依相差比例得皆，超過得 0 分。

參考解答：

```
#include<limits.h>
#include<iostream.h>
#include<stdlib.h>
//////////////////////////////////////////////////////////
//  stack.cpp  wrote Dec 4 1997,Cape Town, SA.          //
//             by Lu-chuan Kung                         //
//////////////////////////////////////////////////////////
#define MAX_N 33
//#define MYDEBUG
// 以下為大會所提供的程式庫
extern "C" int GetX();  // 此函式傳回倉庫的 長度
extern "C" int GetY();  // 此函式傳回倉庫的 寬度
extern "C" int GetZ();  // 此函式傳回倉庫的 高度
extern "C" int GetNextContainer();  // 傳回下一個貨櫃的編號
extern "C" int GetNextAction();     // 取得下一個要求的動作
extern "C" int GetNextStorageTime();// 取得下一個貨櫃的取出時間
extern "C" void RefuseContainer();  // 拒絕放進貨櫃
extern "C" void StoreArrivingContainer(int x, int y); // 把貨櫃放在(x,y)
extern "C" void RemoveContainer(int x, int y); // 取出(x,y)的貨櫃
// 將 (x1,y1) 的貨櫃移至 (x2,y2)
extern "C" int MoveContainer(int x1, int y1, int x2, int y2);
int nx,ny,nz;
int hour;
int huge stack[MAX_N][MAX_N][MAX_N];
int huge expTime[MAX_N][MAX_N][MAX_N];
int top[MAX_N][MAX_N];
void Init(void)
{
    nx = GetX();
    ny = GetY();
    nz = GetZ();
}
// 此函式找到倉庫中最低的(x,y) ，並傳回其高度
int FindLowest(int &x,int &y,int &minexp)
{
```

```
    int min_z = INT_MAX,i,j;
    for(i=0;i<nx;i++) {
        for(j=0;j<ny;j++) {
            if( top[i][j] < min_z ) {
                min_z = top[i][j];
                x = i;
                y = j;
                if( min_z > 0 )
                    minexp = expTime[i][j][ min_z-1 ];
            }
        }
    }
    return min_z;
}
// 傳回除了 (ex,ey) 之外的座標，最低的座標
int FindLowestElse(int &x,int &y,int ex,int ey)
{
    int i,j;
    int min_z = INT_MAX;
    for(i=0;i<nx;i++) {
        for(j=0;j<ny;j++) {
            if( i==ex && j == ey )
                continue;
            if( top[i][j] < min_z ) {
                min_z = top[i][j];
                x = i;
                y = j;
            }
        }
    }
    return min_z;
}
// 放入新的貨櫃
void GetNew(int conno,int exptime)
{
    int z,x,y,minexp;
    int cx,cy,cz;
    int i,j;
    int moveBack = 1;
    // 先找到最低的(x,y)座標
    z = FindLowest(x,y,minexp);
    if( z == nz ) { // 已經滿了
        RefuseContainer(); // 拒絕放入貨櫃
    }
```

```
    if( z == 0 ) {
#ifdef MYDEBUG
        cout << "Storing at " << x+1 << " " << y+1 << endl;
#endif
        // 把貨櫃放在 (x,y)
        StoreArrivingContainer(x+1,y+1);
        stack[x][y][ top[x][y] ] = conno;
        expTime[x][y][ top[x][y] ++ ] = exptime;
    }
    else {
        // 找到除了(x,y)之外最低的座標 (cx,cy)
        cz = FindLowestElse(cx,cy,x,y);
        // 找到 (x,y) 上最高的貨櫃，且其移出時間小於現在要放入的這個
        for(i=0;i<top[x][y];i++)
            if( expTime[x][y][i] < exptime )
                break;
        if( top[x][y] - i > nz - cz -1)
            RefuseContainer();
        // 把 第 i 個貨櫃之上的貨櫃都移到 (cx,cy)上
        for( j=top[x][y]-1; j>=i;j--) {
#ifdef MYDEBUG
            cout << "Moving " << x+1 << " " << y+1 << " to "
                            << cx+1 << " " << cy+1 << endl;
#endif
            MoveContainer(x+1,y+1,cx+1,cy+1);

        }
#ifdef MYDEBUG
        cout << "Storing at " << x+1 << " " << y+1 << endl;
#endif
        // 把進入的貨櫃放在(x,y)
        StoreArrivingContainer(x+1,y+1);
        // 把剛剛放在 (cx,cy) 上的貨櫃放回來
        for( j=top[x][y]-1; j>=i;j--) {
#ifdef MYDEBUG
            cout << "Moving " << cx+1 << " " << cy+1 << " to "
                            << x+1 << " " << y+1 << endl;
#endif
            MoveContainer(cx+1,cy+1,x+1,y+1);
}
        // 在 i 之上的每一個都往上一移一個
        for(j=top[x][y];j>i;j--) {
            stack[x][y][j] = stack[x][y][j-1];
            expTime[x][y][j] = expTime[x][y][j-1];
```

```
        }
        // 存入剛剛放入的貨櫃
        stack[x][y][i] = conno;
        expTime[x][y][i] = exptime;
        top[x][y] ++;
    }
}
// 找指定的貨櫃所在的座標
int Search(int conno,int &x,int &y)
{
    int i,j,k;
    for(i=0;i<nx;i++) {
        for(j=0;j<ny;j++) {
            for(k=0;k<top[i][j];k++)
                if( stack[i][j][k] == conno ) {
                    x = i;
                    y = j;
                    return k;
                }
        }
    }
    return -1;
}
// 移去指定的貨櫃
void Remove(int conno)
{
    int x,y,z ;
    int cx,cy,cz;
    int i;
    // 尋找指定的貨櫃
    z = Search( conno,x,y);
    if( z < 0  ) { // 找不到表示之前有 refuse 過，此時必須離開
        exit(0);
    }
    // 找除了 (x,y) 之外最低的座標
    cz = FindLowestElse(cx,cy,x,y);
    if( top[x][y] - z > nz - cz-1 ) // couldn't move
        exit(0);
    // 把 conno 之上的貨櫃都移至 (cx,cy)
    for(i=top[x][y]-1;i>z;i--) {
#ifdef MYDEBUG
            cout << "Moving " << x+1 << " " << y+1 << " to "
                                << cx+1 << " " << cy+1 << endl;
#endif
```

```
        MoveContainer(x+1,y+1,cx+1,cy+1);
    }
#ifdef MYDEBUG
    cout << "Removing container " << x+1 << " " << y+1 << endl;
#endif
    // 移出貨櫃
    RemoveContainer(x+1,y+1);
    // 搬回來
    for(i=top[x][y]-1;i>z;i--) {
#ifdef MYDEBUG
            cout << "Moving " << cx+1 << " " << cy+1 << " to "
                                << x+1 << " " << y+1 << endl;
#endif
        MoveContainer(cx+1,cy+1,x+1,y+1);
    }
    for(i=z;i<top[x][y]-1;i++) {
        stack[x][y][i] = stack[x][y][i+1];
        expTime[x][y][i] = expTime[x][y][i+1];
    }
    top[x][y] --;
}
// 主函式
int main(void)
{
    int no,action,time;
    Init();
    hour = 0;
    while(1) {
        no = GetNextContainer() ;
        if( no == 0 ) // 貨櫃程式結束
            break;
        action = GetNextAction(); // 取得下一個動作
        switch( action ) {
            case 1: // 放入一個新的貨櫃
                time = GetNextStorageTime();
                GetNew(no,time);
                hour++;
                break;
            case 2: // 移出一個指定的貨櫃
                Remove(no);
                break;
        }
    }
    return 0;}
```

十一、1996 年 IOI 試題與參考解答

IOI'96 (第一天第一題)　　任務描述

遊戲 (A Game)

這是一個兩個人玩的遊戲，遊戲盤上放有一排正整數。

兩位玩者輪流玩。當輪到時，玩者就從這一排數字的最左端或最右端的二個數字中選擇一個。被選到的數字就從遊戲盤上除去。當所有數字都被選取除去後，遊戲就結束。如果第一個遊戲者所選取的數字總和跟第二位遊戲者所選取的數字總和一樣大或更大時，則第一個遊戲者就贏得勝利。

第二位遊戲者一定做全局考量最好的選擇。

此遊戲由第一位遊戲者開始先選取。

假如此遊戲盤上一開始就有偶數個數字，則第一位遊戲者一定有一個必勝的策略，你要寫出策略的程式，好讓第一為遊戲者勝利。第二位遊戲者的數字會由一個電腦程式提供。這兩位遊戲者可透過放在已提供給你的 Play 模組裡的三個副程式來溝通。

這些程式是 StartGame, MyMove, 和 YourMove, 第一位遊戲者應先呼叫沒有參數的 StartGame 副程式來開始玩遊戲。

假如第一位遊戲者選擇了最左端的數字，則他應呼叫副程式 MyMove('L')。

同樣地，如果他呼叫副程式MyMove('R')，則他就是在告訴第二位遊戲者他選擇最右端的數字。第二位遊戲者(就是電腦程式)馬上做出選擇，而第一位遊戲者可從呼叫副程式 YourMove(C)後得知這個選擇，在這裡，C是一個char變數(在C/C^{++}中就寫成YourMove(&C))，C的值可能是'L'或是'R'，就看第二位遊戲者的選擇是從左端或右端而定。

輸入資料

檔案 INPUT.TXT 的第一行包含這個遊戲大小的起始值 N，N 是偶數且 2<=N<=100，接下來的 N 行中，每一行都有一個數字，代表著遊戲盤上的數字，排列次序為由左至右，每一個數字最大為 200。

輸出資料

當遊戲結束時，你的程式應輸出遊戲結果至輸出檔 OUTPUT.TXT 裏，輸出檔的第一行應有兩個數字，第一個數字是第一個遊戲者所選取的數字的總和，而第二個數字是第二位遊戲

者所選取的數字的總和。

你的程式一定要確實有玩此遊戲，而且你的輸出必須和所玩的遊戲結果吻合。

輸入和輸出的範例

圖一提供一個輸入檔包含一個起始遊戲盤和一個可能輸出檔。

INPUT.TXT	OUTPUT.TXT
6	15 14
4	
7	
2	
9	
5	
2	

圖一

GAME 之參考解答

{由下列簡單的觀察中，將帶給我們一個解決問題的方向：
因為在遊戲板上最初包含有一序列偶數個數字元素，當輪到第一個遊戲者玩時，數列的一端位在奇數的位置上而另一端則位在偶數的位置上，因此，第一個遊戲者每一次皆可以自奇數或偶數位置上任選一個數字元素。
此演算法在遊戲之前，先根據最初遊戲板上的數字加以處理，計算出 OddSum 和 EvenSum 兩個數值，OddSum 為所有奇數位置數字的總和，而 EvenSum 則為偶數位置數字的總和，當 OddSum>=EvenSum 時，第一個遊戲者每次總是自奇數位置選取一個數字將迫使第二個遊戲者必須自偶數位置中選取一數字，反之，若 OddSum<EvenSum 則亦以相同方式處理。}

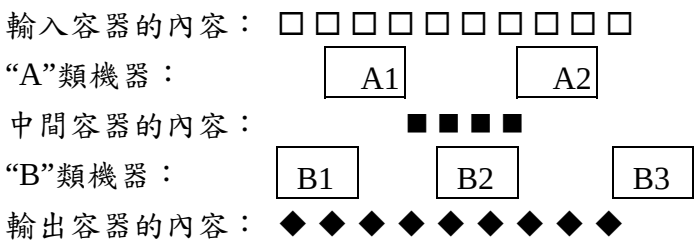
```
Program Game;
Uses Play;
Const
  MaxN=100;                    { max size of the board }
Var
  N: Word;                     { size of the board }
  Board:Array[1..MaxN] Of Word; {contents of the board }
  Sum:Word;                    { sum of the elements in the initial board }
  Sel:Word;                    { sum of the elements selected by the first player }
  Odds:Boolean;
```

```
Procedure ReadInput;
{ Global output variables: N, Board, Sum }
Var InFile: Text;i:Word;
Begin
   Assign(InFile,'input.txt'); Reset(InFile);
   ReadLn(InFile,N);
   Sum:=0;
   For i:=1 To N Do Begin
      ReadLn(InFile,Board[i]);
      Sum:=Sum+Board[i];
   End;
   Close(InFile);
End;
Procedure Preprocess;
{ Global input variables: N, Board }
{ Global output variable: Odds }
Var i:Word;
   OddSum,EvenSum:Word;
Begin
   OddSum:=0;
   EvenSum:=0;
   For i:=1 To N Do
      If Odd(i) Then Inc(OddSum,Board[i])
                Else Inc(EvenSum,Board[i]);
   {end for i};
   Odds:=OddSum>=EvenSum;
End {Preprocess};
Procedure Playing;
{ Global input variable: N, Board, Odds }
{ Global output variable: Sel }
Var M,i:Word;
   C1,                { move of the first player }
   C2:Char;           { move of the second player }
   Head,              { position of the left end of the board }
   Tail:1..MaxN; { position of the right end of the board }
Begin
   Sel:=0;
   Head:=1; Tail:=N;
```

```
  M:=N Div 2;                { number of moves for one player}
  For i:=1 To M Do Begin
    If Odds Then Begin {select from the odd position}
      If Odd(Head) Then Begin
          Sel:=Sel+Board[Head];
          Inc(Head);
          C1:='L';
        End Else Begin
          Sel:=Sel+Board[Tail];
          Dec(Tail);
          C1:='R';
        End;
    End Else Begin       {select from the even position}
      If Odd(Head) Then Begin
        Sel:=Sel+Board[Tail];
        Dec(Tail);
        C1:='R';
      End Else Begin
        Sel:=Sel+Board[Head];
        Inc(Head);
        C1:='L';
      End;
    End {Odd-Even};
    MyMove(C1);            { perform the move }
    YourMove(C2);          { obtain the second player's move }
    If C2='L' Then Inc(Head)
                Else Dec(Tail);
  End{For i};
End;
Procedure WriteOut;
Var OutFile: Text;
Begin
  Assign(OutFile,'output.txt'); Rewrite(OutFile);
  WriteLn(OutFile,Sel,' ',Sum-Sel);
  Close(OutFile);
End;
Begin { Program }
  ReadInput;
```

```
    Preprocess;
    StartGame;
    Playing;
    WriteOut;
End.
```

IOI'96 (第一天第二題)　　任務描述



圖一

工作處理 (Job Processing)

有一間工廠有一生產線。每件工作必須執行兩個作業：首先做作業"A"，然後做作業"B"。每個作業都有數部機器可以替它們執行。圖一顯示出此生產線的結構。其工作方式如下：一台"A"類機器自輸入容器中取出一件工作，執行作業"A"，然後將此工作放入中間容器中。一台"B"類機器自中間容器中取出一件工作，執行作業"B"然後將此工作放入輸出容器中。這些機器有不同的操作性能，每台機器有它自己的作業處理所需時間。

求出所有 N 件工作之作業"A"可以被完成的最早時間,假設這些工作在時間 0 時均備妥待處理(子任務 A)。也請計算出完成所有 N 件工作之兩個作業所需的最少時間量(子任務 B)。

輸入資料

輸出檔案 INPUT.TXT 含有五行的數個正整數。第一行記載 N，及工作的數量 (1<=N<=1000)。第二行給定"A"類機器的台數 M1(1<=M1<=30)。第三行含有 M1 個整數，為每一台"A"類機器的工作處理時間。第四行和第五行分別含有"B"類機器的台數 M2(1<=M2<=30)以及每一台"B"類機器的工作處理時間。工作處理時間是以每單位時間來計算的，它已包括了在處理前自一容器中取出一件工作並在處理後將之放回容器中的時間。每一工作處理時間至少為 1，至多為 20。

輸出資料

你的程式應寫入第二行到檔案 OUTPUT.TXT 中。第一行應有一正整數，即為子任務 A 的答案。第二行應含子任務 B 的答案。

輸入和輸出範例

圖二為一可能的輸入檔案以及其對應的輸出檔案。

INPUT.TXT	OUTPUT.TXT
5	3
2	5
1 1	
3	
3 1 4	

圖二

JOBS 之參考解答

{考慮下列資料結構：

```
Const
   MaxM=30;                          (* max number of machines *)
Type
   Operation='A'..'B';
   ProcTime=Array[Operation,1..MaxM] Of Word;
Var
   N:Longint;                        (* number of jobs *)
   M:Array[Operation] Of Word;(* M[op] is the number of machines of type op *)
   PTime: ProcTime;                  (* PTime[op,m] is the processing time for machine
                                        m of type op *)
```

子任務 A

明顯地，一個最佳化的工作排程可藉由下列方法達成，即每一部機器皆由時間 0 開始處理且在完成該機器所需完成的所有工作之前絕不停滯。

在時間 t 之內，op 型態之機器所能完成的最大工作量為 t div PTime[Op,m]. 因此，op 型態之機器完成所有 N 個工作之最短時間為使得下列總和：

(t Div PTime[Op,i]) (for i:=1 to M[Op]) 大於或等於 N 之 t 的最小值

下列演算法將計算出 op 型態之機器處理 N 個工作所需之的總時間 t 之最小值：

```
t:=0;
Repeat
   Inc(t);
   Processed:=0;
   For i:=1 To M[Op] Do
      Processed:=Processed+(t Div PTime[Op,i]);
```

```
    Until Processed>=N;
```

子任務 B

很明顯地，A 型態機器的工作排程與子任務 A 之排程相同。令 TAB 為完成所有 N 個工作之兩個程序所需的最短時間，我們可以假設 B 型態的機器正好於 TAB 的時間完成所有的工作且兩個連續的工作之間沒有停滯。

假使這不是最開始的工作則我們將可因此而修正最佳排程；先假設在中間容器中已有 A 型態機器所產生之工作，則這項工作便可被在此時被 B 型態機器使用。令 d 為依據最佳化排程，型態 B 之機器開始處理第一件工作的時間，TB 表示完成所有 N 個工作中之 B 程序所需花費的最短時間。基於子任務 A 的相同主張，可得之 TAB=d+TB.，我們已經有一個演算法可以計算 TB 的值，因此，可以此來發展計算延遲時間 d 的演算法。

令 Finish(Op,t) 為依據最佳化排程在 t 時間內所能完成之 OP 程序的工作量。則延遲時間 d 為滿足以下條件之最小值：

對每一個 t 而言，0 <= t < TB，在時間 d+t 內，在中間的暫存區中至少有 Finish('B',TB-t) 數量的工作是可用的。我們可以藉由給定一個 d 值來檢查其是否滿足上述之條件來求得 d 值，因此，我們可以由 d=1 開始計算並且逐漸增加 d 值，直到暫存區中的工作數量已大於 Finish('B',TB-t) 數量為止。

使用此遞增的方法將可加快計算的速度，此方法的作法如下：

起始值為 d = 1，假設上述的條件在一個給定的 d 值與介於 0,...,ts 的 t 值下可以滿足，當上述條件在 t=ts+1 不能滿足時，則將 d 累加直到條件滿足為止，在本程式中，此方法的實作以 procedure Adjust 來達成。}

```
Program Jobs;
Const
   MaxM=30;                          { max number of machines }
Type
   Operation='A'..'B';
   ProcTime=Array[Operation,1..MaxM] Of Word;
Var
   N:Longint;                        { number of jobs }
   M:Array[Operation] Of Word; { M[op] is the number of machines of type op }
   PTime: ProcTime;                  { PTime[op,m] is the processing time for machine
                                       m of type op }
   TA,              { the time needed to perform single operation A on all N jobs }
   TB: Longint;{ the time needed to perform single operation B on all N jobs }
   d :Longint;
Procedure ReadInput;
{ Global output variables: N, M, PTime }
   Var InFile: Text; i: Word;
   Begin
```

```
    Assign(InFile, 'input.txt'); Reset(InFile);
    ReadLn(InFile,N);
    ReadLn(InFile,M['A']);
    For i:=1 To M['A'] Do
      Read(InFile, PTime['A',i]);
    ReadLn(InFile);
    ReadLn(InFile,M['B']);
    For i:=1 To M['B'] Do
      Read(InFile, PTime['B',i]);
    Close(InFile);
  End {ReadInput};
Function Compute_Time(Op:Operation):Longint;
{Computes the minimal time that is needed to perform operation Op on N jobs}
{ Global input variables: M, PTime }
  Var t,Processed:Longint;
    i:Word;
  Begin
    t:=0;
    Repeat
      Inc(t);
      Processed:=0;
      For i:=1 To M[Op] Do
        Processed:=Processed+(t Div PTime[Op,i]);
    Until Processed>=N;
    Compute_Time:=t;
  End;{Compute_Time}
Function Finish(Op:Operation; t: Longint): Longint;
{ Finish(Op,t) is the number of jobs that are finished at time t
  according to the optimal schedule for single operation Op for N jobs. }
{ Global input variables: N, M, PTime }
  Var Res,UpTo: Longint;
    i: Word;
  Begin
    Res:=0;
    For i:=1 To M[Op] Do
      If (t Mod PTime[Op,i])=0 Then Inc(Res);
    { If the number of jobs that can be completed up to time t
      is more then N then decrease Res to the proper value. }
```

```
      UpTo:=0;
      For i:=1 To M[Op] Do UpTo:= UpTo+ (t-1) Div PTime[Op,i];
      If Upto >= N Then
         Res:= 0
      Else If Upto+Res>N Then
         Res:= N-UpTo;
      Finish:=Res;
   End {Finish};
Procedure Adjust;
{ Computes the delay time d when the first type B machine starts to work }
{ Global input variables: TA, TB }
{ Global output variables: d }
   Var Inter:Word;{ number of jobs in the intermediate container }
      t: Longint;
      JB:Word;
   Begin
      d:=1; t:=0; Inter:=0;
      While d+t<TA Do Begin
         Inter:=Inter+Finish('A',d+t);
         JB:=Finish('B',TB-t);       { # jobs starting at time d+t }
         While Inter<JB Do Begin    { while not enough jobs available }
            Inc(d);
            Inter:=Inter+Finish('A',d+t);
         End;
         Inter:=Inter-JB;
         Inc(t);
      End;
   End;{Adjust}
Procedure WriteOut(AnswerA,AnswerB:Longint);
   Var OutFile: Text;
   Begin
      Assign(OutFile, 'output.txt'); Rewrite(OutFile);
      WriteLn(OutFile, AnswerA);
      WriteLn(OutFile, AnswerB);
      Close(OutFile);
   End;{WriteOut}
Begin {Main}
   ReadInput;
```

```
    TA:= Compute_Time('A');
    TB:= Compute_Time('B');
    Adjust;
    WriteOut(TA, d+TB);
End.
```

IOI'96 (第一天第三題)　　任務描述

學校網路(Network of Schools)

數個學校連在電腦網路上。這些學校達成協議：每個學校將傳送軟體給某一些學校("接收學校")。

如果我們有一份新的軟體且希望所有學校都能收到一份，請計算出最少我們必須先將此軟體送給幾個學校才可(子任務 A)。

另一任務，我們要確定如果送軟體到任一學校，該軟體會到達網路中所有學校。為達成該目標，我們必須擴展接收學校。

計算最小的擴展數目，使得我們不論送軟體到任一學校，它會到達所有其他學校(子任務 B)。

每一個擴展是任一個學校每增加一個接收學校。

輸入資料

檔案 INPUT.TXT 的第一行包含一整數 N：網路中學校的數目(2<=N<=100)。學校的編號是前 N 個正整數。接下來的 N 行中每行描述某學校的所有接收學校。第 i+1 行列出第 i 個學校的接收學校編號，而且每行以 0 結束，若沒有接收學校則該行只有 0，注意，若 B 是學校 A 的接收學校，則 A 未必是 B 的接收學校。

輸出資料

你的程式應該輸出兩行到檔案 OUTPUT.TXT，第一行應該包含一正整數：子任務 A 的答案。第二行應該包含子任務 B 的答案。

輸入和輸出範例

圖一提供一個可能的輸入檔和對應輸出檔。

INPUT.TXT	OUTPUT.TXT
5	1
2 4 3 0	2
4 5 0	
0	
0	
1 0	

圖一

NET 之參考解答

{網路可以以一個有向圖來表示，圖中的每一個頂點代表學校，以(A, B)表有向圖的一個邊，若且唯若學校 B 是學校 A 的一個分支，以下就先以圖形的術語重新定義工作。

我們以 p->q 來表示圖形中從 p 至 q 的有向路徑，如果每一個頂點 q 與 D 集合中的頂點 p 皆有一路徑 p->q 存在，則稱此頂點集合為圖形 G 的支配者集合。

子任務 A 即是要找出圖形 G 中具有最少元素的支配者集合：

如果每一個頂點 p 與 CD 集合中的頂點 q 皆有一路徑 p->q 存在，則稱此頂點集合 CD 為圖形 G 的共同支配者集合。當圖形中的每一個頂點 p 和 q 間皆有路徑 p->q 及 q->p 存在，則稱此圖形為強連結圖形。

子任務 B 的解決方法即是找出使圖形 G 成為強連結圖形所需的最少數量的新的邊：

令|S|表示集合 S 的元素數量，且令 D 為圖形 G 的支配者集合，CD 為圖形 G 的共同支配者集合。當圖形 G 為強連結圖形時，我們可以證明子任務 B 之解答為 0，若 G 不是強連結圖形時則為 Max(|D|, |CD|)，即|D|與|CD|之最大值。其證明如下列 S1 及 S2 所述：

我們可以在不喪失一般性的原則下假定|D|<=|CD|，

S1. 若 D 是一個包含 p 的元素集合且 CD 包含了元素 q1,...,qk，則引入(q1, p), ... , (qk, p)等新邊將可使圖形 G 成為強連結圖形。

證明：令 u,v 為圖形 G 中的任意頂點，則在 CD 中存在一個元素 qi 使得 u->qi，因此，u->qi->p->v 為從 u 至 v 的一個路徑。

S2. 若 |D|>1 則在 D 中存在一個頂點 p 且在 CD 中存在一個頂點 q，使得在 G 中引入一個新邊(q, p)後將使 D-[p] 成為一個新的支配者集合，CD-[q]成為一個新的共同支配者集合。

證明：因為|D|>1，所以在 D 中存在不同的頂點 p1 和 p2，且在 CD 中存在不同的頂點 q1 和 q2，使得 p1->q1 且 p2->q2 ，則新邊(p,q)將是(q1,p2)，事實上，任意可藉由路徑 p2->u 從 p2 到達的頂點 u，將可藉由路徑 p1->q1->p2->u 從 p1 到達且對於任意路徑 v->q1 將有一個新路徑 v->q1->p2->q2 存在於新圖形中。

顯然的，一個圖形 G 中的共同支配者集合為轉置後的圖形 GT 的支配者集合，反之亦同。因此，我們可以藉由將圖形 G 轉置並計算其最小的支配者集合而求得圖形 G 的共同支配者集合。下列為計算最小支配者集合的策略。（集合的操作使用 Pascal 的術語）

```
Dominated:=[];
D:=[];
While there is a p not in Dominated Do Begin
   Search(p);(* put all vertices in set Reachable that are reachable from p*)
   Dominated:=Dominated+Reachable;
   D:=D-Reachable; (* exclude all elements of D that are in Reachable *)
   Include p in D;
End;
```

明顯地，此演算法中所產生的集合 D 是一個支配者集合，假設集合 D 包含頂點 p1,...,pk 且 D 並不是最小的，也就是說有一個最小的支配者集合 Q 包含頂點 q1,...ql，且 l<k。
因為 D 是一個支配者集合且 Q 是一個最小支配者集合其遵循下列法則：
對於每一個 Q 中的 qi 存在唯一的 D 中的 pi，使得 qi 可藉由路徑 pi->qi 從 pi 到達之，但是，由於每一個頂點可以從 Q 到達，因此 pk 亦可從 Q 中的一些頂點到達，稱為 qi->pk。
由此可知，從 pi 到 pk 存在一個路徑 pi->qi->pk。此演算法執行了 Search(pi) 和 Search(pk)，任一個 Search(pi) 或 Search(pk) 首先被執行，頂點 pk 被排除在 D 之外，因為 pk 可以從 pi 到達，此與 D 不是最小的假設不符。
上述的演算法加以修正可避免 union (+) 和 difference (-)之集合的操作，事實上，當 Search(p) 被執行，我們可以將 p1 包含在支配者集合中且將 p 排除在 D 之外。}

```
Program Net;
Const
   MaxN=200;                  { max number of schools }
Type
   GraphType=Array[1..MaxN,0..MaxN] Of 0..MaxN;
   VertexSet=Set Of 1..MaxN;
Var
   OutFile: Text;
   N :Word;                   { the number of schools }
   G: GraphType;              { representation of the network with graph G; }
                               { G[p,0] is the number of edges outgoing from p }
                               { the outgoing edges from p: (p, G[p,i]) 1<=i<=G[p,0]) }
   Domin,                     { dominator set }
   CoDomin: VertexSet; { codominator set }
   NoDomins,                  { number of dominator elements     }
   NoCoDomins: 0..MaxN;{ number of codominator elements }
   AnswerB: 0..MaxN;     { solution of subtask B }
   p: 0..MaxN;
Procedure ReadInput;
{ Global output variables: N, G }
   Var InFile: Text;
   i,p: Word;
   Begin
      Assign(InFile, 'input.txt'); Reset(InFile);
      ReadLn(InFile,N);
      For i:=1 To N Do
         G[i,0]:=0;
      For i:=1 To N Do Begin
```

```
        Read(InFile, p);
        While p<>0 Do Begin
          Inc(G[i,0]);
          G[i,G[i,0]]:=p;
          Read(InFile, p);
        End;
        ReadLn(InFile);
      End;
      Close(InFile);
  End { ReadInput };
Procedure ComputeDomin(Const G: GraphType; Var D: VertexSet);
{ Computes a minimal dominator set D of graph G }
{ Global input variables: N }
  Var
    Dominated, Reachable: Set of 1..MaxN;
    p: 1..MaxN;
  Procedure Search(p:Word);
    Var i: Word;
  Begin
    Exclude(D, p);
    Include(Dominated, p);
    For i:= 1 To G[p,0] Do
      If Not (G[p,i] in Reachable) Then Begin
        Include(Reachable,G[p,i]);
        Search(G[p,i]);
      End;
  End { Search };
  Begin { ComputeDomin }
    D:=[];
    Dominated:=[];
    For p:=1 To N Do
      If Not (p In Dominated) Then Begin
        Reachable:=[p];
        Search(p);
        Include(D, p);
      End;
  End { ComputeDomin };
Procedure ComputeCoDomin(Const G: GraphType; Var CD: VertexSet);
```

```
{ Computes a minimal codominator set D of graph G }
{ Global input variables: N }
  Var
    GT: GraphType;                  { transposed graph of G }
    p,q: 1..MaxN; i:Word;
  Begin { ComputeCoDomin }
    For p:=1 To N Do
      GT[p,0]:=0;
    For p:=1 To N Do              { compute the transpose of the graph G in GT }
      For i:=1 To G[p,0] Do Begin
        q:=G[p,i];
        Inc(GT[q,0]); GT[q,GT[q,0]]:=p;
      End;
  ComputeDomin(GT, CD)            { computes CD, the dominator set of GT }
  End;{ ComputeCoDomin }
Begin { Program }
  ReadInput;
  ComputeDomin(G,Domin);
  ComputeCoDomin(G,CoDomin);
  NoDomins:=0;
  For p:=1 To N Do        { count the number of elements in the set Domin }
    If p In Domin Then Inc(NoDomins);
  NoCoDomins:=0;
  For p:=1 To N Do        { count the number of elements in the set CoDomin }
    If p In CoDomin Then Inc(NoCoDomins);
  If (Domin=[1]) And (CoDomin=[1])   { strongly connected }
    Then AnswerB:=0
    Else If NoDomins > NoCoDomins
      Then AnswerB:=NoDomins
      Else AnswerB:=NoCoDomins;
  Assign(OutFile, 'output.txt'); Rewrite(OutFile);
  WriteLn(OutFile, NoDomins);
  Writeln(OutFile, AnswerB);
  Close(OutFile);
End.
{Scientific Committee IOI'96 提供}
```

IOI'96 (第二天第一題)　　任務描述

三數值序列之排序 (Sorting a Three-Valued Sequance)

排序是最常見的運算動作。假設有個特殊的排序問題：每一筆要被排序的資料(數字)都有三種可能的數值。舉例來說，當我們在排序比賽獎牌時，我們會照獎牌類(modal value)來排，也就是將所有的金牌排在最前面，再來是所有的銀牌，最後才是所有的銅牌。

在這個任務中，每一筆資料的可能數值皆為正整數 1, 2, 或是 3；你必須將所有資料排序成非遞減(non-decreasing)之次序。排序的方法必須是一系列的資料互換的動作。第 p 筆跟第 q 筆資料互換的動作，是指存在第 p 筆的資料與存在第 q 筆資料互換。

給定一系列的資料值，寫個程式來計算最少要做幾次互換的動作，才能將此系列排序完畢(子任務 A)。另外，找出這些互換的動作(子任務 B)。

輸入資料

輸入檔 INPUT.TXT 的第一行含有 N(1<=N<=1000)，此是要被排列的資料個數。接下來的 N 行各含有一筆資料的數值。

輸出資料

請輸出完成排序所須之最少互換動作次數 L 到輸出檔 OUTPUT.TXT 的第一行(子任務 A)。接下來的 L 行，請按照互換動作的次序輸出每個互換動作。每行用兩個數字 p 和 q 來代表一個互換動作。p 和 q 是兩筆被互換的資料的位置(子任務 B)。假設資料位置是從 1 到 N。

輸入和輸出範例

圖一提供一個輸入檔和一個對應輸出檔。

INPUT.TXT
9
2
2
1
3
3
3
2
3
1

OUTPUT.TXT
4
1 3
4 7
9 2
5 9

圖一

SORT3 之參考解答

{本題解答之基本觀念略述於下：

首先計算在輸入序列中 x=1,2,3 時的表現值 Na[x]，也就是說指定 Na[1]、Na[2]及 Na[3]的數值為 x=1、x=2 及 x=3 的總個數；當 x 的位置與經排序後序列 y 的位置相同時，我們稱 x 在正確位置上，並使用 x:y 來表示 x 與 y 在相同位置。接著再計算對所有 x 和 y 而言 x 在 y 位置上的數量 NEP[x,y]，上述的演算法以虛擬碼的方式陳述於下：

```
NoCh:=0;
  While Not Sorted(S) Do Begin
    If there are x and y in each other's place Then Begin
      Inc(NoCh);
      Exchange x and y;
      Update NEP[x,y] and NEP[y,x];
    End Else Begin
      If (NEP[1,2]>0) And (NEP[3,1]>0) Then Begin
        Exchange a pair of elements 3:1 and 1:2 ;
        Update NEP[1,2] And NEP[3,1];
        Inc(NoCh);
      End;
      If (NEP[2,1]>0) And (NEP[1,3]>0) Then Begin
        Exchange a pair of elements 2:1 and 1:3 ;
        Update NEP[2,1] And NEP[1,3];
        Inc(NoCh);
      End;
    End;
  End;
```

首先我們以下列的表示法代表計算交換次數的演算法：

```
Ch(S)=Min(NEP[1,2], NEP[2,1])+
          Min(NEP[1,3], NEP[3,1])+
          Min(NEP[2,3], NEP[3,2])+
          2*Abs(NEP[1,2]-NEP[2,1])
```

在對所有 x<>y 的狀況運算了 Min(NEP[x,y], NEP[y,x])後，其包含了兩種結果，當 NEP[1,2]>NEP[2,1]時 1:2, 2:3, 3:1；而當 NEP[1,2]<NEP[2,1]時則 1:3, 2:1, 3:2。第一種狀況下，演算法將會交換 2:1 及 1:3，其結果會造成元件 2 放在第 3 個位置上，因此在第二個迴圈中，將會交換 2:3 及 3:2；第二種情況也和第一種狀況十分類似。我們可據此總結，Ch(S)是一個正確的計算交換次數之演算法。

讓我們以 OCh(S)表示序列 S 經搜尋所需的最小交換次數，以下我們將會證明 Ch(S)= OCh(S)，吾人將利用歸納 OCh(S)的值而得証。

若 OCh(S)=0 或 1 則此陳述明顯地被證實；

今假設在 OCh(S)<k(k>1)的情況下，對所有的 S 而言，Och(S)=Ch(S)；
再令 S 為一個序列且 OCh(S)=k。
若是經過一個最佳化的交換次數後 S 被排序完成，
假設第一個交換運作是 x1:y1 及 x2:y2(或 x1:y2 及 x1:y1)，我們以 S'來表示其結果的序列。
我們可以將此結果分為下列兩種狀況：

C1: x1=y2 及 x2=y1 或
NEP[1,2]>NEP[2,1]及 x1=1, y1=2, x2=3, y2=1 或
x1=1, y1=2, x2=2, y2=3 或
x1=3, y1=1, x2=2, y2=3 或
NEP[1,2]>NEP[2,1]及 x1=2, y1=1, x2=3, y2=2 或
x1=2, y1=1, x2=1, y2=3 或
x1=3, y1=2, x2=1, y2=3 或

C2:所有其他對 x1, y1, x2, y2 的組合。

我們可以驗證出一個常式即在狀況一中 Ch(S')=Ch(S)-1，在狀況二中 Ch(S')>= Ch(S)，由 C1 我們可以歸納出一個交換之假設，即 Ch(S)=Ch(S')+1=OCh(S')+1 =OCh(S)；而 C2 則不符合最佳化之假設，因此最佳化序列的交換運作只有可能是 C1 之狀況。
為了要發展一個建構交換運作序列有效率的演算法，我們首先利用一個陣列 First，First[x,y] 總是包含第一個 x 在 y 的適當位置；First[x,y]是被事先處理的程序所計算出的，並且在每次交換運作後被更新。}

```
Program Sort3;
Const
  MaxN =1000;                    { max number of elements to sort }
Type
  ElemType = 1..3;
  ArrayType= Array[1..MaxN] Of ElemType;
  Matrix     = Array[ElemType, ElemType] of Integer;
Var
  N : Word;                      { number of elements to sort                }
  S : ArrayType;                 { array of elements to sort                 }
  Na: Array[ElemType] Of Word;{ Na[x] is the number of x's in the input }
  NEP : Matrix;                  { NEP[x,y] is the number of x's             }
                                   { in place of y's                           }
  First: Matrix;                 { First[x,y] is the first position of x in place of y's}
  NoCh: Word;                      { number of exchange operations             }
  OutFile: Text;
Procedure ReadInput;                 { Global output variables: N, S, Na }
  Var
    InFile: Text;
```

```
    i,j: Word;
  Begin
    Assign(InFile, 'input.txt');
    Reset(Infile);
    ReadLn(InFile, N);
    For i:=1 To 3 Do Na[i]:=0;
    For i:=1 To N Do Begin
      ReadLn(InFile,S[i]);
      Inc(Na[S[i]]);
    End;
    Close(InFile);
  End { ReadInput };
Function Min(X,Y: Word):Word;
  Begin
    If X<Y Then Min:=X
           ELse Min:=Y
  End { Min };
Procedure Preprocess;{ Global input variables: N, S, Na }
{ Global output variables: NEP, First }
  Var i,j,M:Word;
  Begin
    For i:=1 To 3 Do Begin
      For j:=1 To 3 Do Begin
        NEP[i,j]:=0; First[i,j]:=0
      End { For j };
    End { For i };
    For i:=1 To N Do Begin
      If i<=Na[1] Then Begin                          { S[i] is in place of 1's      }
        If NEP[S[i],1]=0 Then First[S[i],1]:=i; { first S[i] in place of 1's }
        Inc(NEP[S[i],1]);
      End Else If i<=Na[1]+Na[2] Then Begin       { S[i] is in place of 2's      }
        If NEP[S[i],2]=0 Then First[S[i],2]:=i; { first S[i] in place of 2's }
        Inc(NEP[S[i],2]);
      End Else Begin                                  { S[i] is in place of 3's      }
        If NEP[S[i],3]=0 Then First[S[i],3]:=i; { first S[i] in place of 3's }
        Inc(NEP[S[i],3])
      End;
    End { For i };
```

```
    NoCh:= Min(NEP[1,2], NEP[2,1])+                    { subtask A }
           Min(NEP[1,3], NEP[3,1])+
           Min(NEP[2,3], NEP[3,2])+
           2*Abs(NEP[1,2]-NEP[2,1]);
  End;{ Preprocess }
  Procedure Next(i1,i2:Byte);
  { Global input-output variables: First, NEP }
    Begin
      Dec(NEP[i1,i2]);
      If NEP[i1,i2]>0 Then Begin
        Repeat
          Inc(First[i1,i2]);
        Until S[First[i1,i2]]=i1;
      End;
    End { Next };
Procedure Pairs;
  Var M,i,x,y :Word;
  Begin
    For x:=1 To 3 Do
      For y:=x+1 To 3 Do Begin
        M:=Min(NEP[x,y], NEP[y,x]);
        For i:=1 To M Do Begin
          WriteLn(OutFile, First[x,y],' ',First[y,x]);
          Next(x,y); Next(y,x);
        End;
      End;
   End { Pairs };
Procedure Triples;
  Var M,i: Word;
  Begin
    If NEP[1,2] > 0 Then Begin
      M:=NEP[1,2];
      For i:=1 To M Do Begin
        WriteLn(OutFile, First[3,1],' ',First[1,2]);
        WriteLn(OutFile, First[1,2],' ',First[2,3]);
        Next(3,1); Next(1,2); Next(2,3);
      End;
    End Else Begin
```

```
          M:=NEP[2,1];
          For i:=1 To M Do Begin
             WriteLn(OutFile, First[2,1],' ',First[1,3]);
             WriteLn(OutFile, First[1,3],' ',First[3,2]);
             Next(2,1); Next(3,2); Next(1,3);
          End;
       End;
    End;
Begin { Program }
    ReadInput;
    Preprocess;
    Assign(OutFile, 'output.txt');
    Rewrite(OutFile);
    WriteLn(OutFile,NoCh);
    Pairs;
    Triples;
    Close(OutFile);
End.
```

IOI'96 (第二天第二題)　　任務描述

最長字首(Longest Prefix)

某些生物體的結構是由它們的組成序列來表示，這些組成是由大寫字母代表；生物學家想要把長序列分解成較短序列，每個短序列都叫做元件(primitive)。若從每個元件集合P內可以提出n個元件$p_1, \ldots, p_n$，使得元件$p_1, \ldots, p_n$的連結等於某序列S，則我們說序列S可以由元件集合P構成。元件$p_1, \ldots, p_n$的連結指的是它們依該次序放在一起，而且中間沒有空白。在連結時，相同的元件可以出現超過一次，而且未必所有P集合的元件都出現。

例如，序列 ABABACABAAB 可由下列元件集合構成：

{ A, AB, BA, CA, BBC }

S 的前 K 個字元，就是 S 的長度為 K 的字首。

寫一程式，輸入某元件集合 P 和某序列 T。計算可以由集合 P 內元件構成的最長字首的長度。

輸入資料

輸入資料出現在以下兩個檔案。檔案 INPUT.TXT 描述元件集合 P，而檔案 DATA.TXT 包含要檢查的序列 T。INPUT.TXT 的第一行包含 N，即 P 內元件的個數(1<=N<=100)，每個元件由兩個連續行所提供。第一行包含元件的長度 L(1<=L<=20)。而第二行為長度 L 的大寫(由'A'到'Z')字串元件。每一個元件都不同。

檔案 DATA.TXT 的每一行的第一個位置包含一個大寫字母。該檔案以單一句點('.')的新行結束。

此序列的長度是至少為 1，至多為 500,000。

輸出資料

計算可以由集合 P 構成的 T 的最長字首的長度，然後輸出到檔案 OUTPUT.TXT 的第一行。

輸入和輸出的範例

圖一提供兩個輸入檔和一個對應輸出檔。

INPUT.TXT
5
1
A
2
AB
3
BBC
2
CA
2
BA

DATA.TXT
A
B
A
B
A
C
A
B
A
A
B
C
B
.

OUTPUT.TXT
11

圖一

PREFIX之參考解答

{令 S 表字母組成序列、P 表元件集合，並以 Suff(S,P)表示 v 序列集合，v 包含下列兩個條件：

(1)v 是 P 中的字首元件

(2)對 u 而言 S=uv

(對兩個字母序列 u 及 v 而言，我們以 uv 表示其連結)

若且唯若在序列 Suff(S,P)中為一空序列，則 S 可以完全被元件集合 P 所構成；然而若且唯若 Suff(S,P)非空序列，則 S 必可於其右方加入一序列 u 使 Su 被元件集合 P 構成。因此假設資料檔包含被檢驗過的序列資料，那麼下列之演算法即為此工作任務之解答：

```
Res:=0;
S:=empty; NoS:=1;
ReadLn(DataFile,X);
Slength:=1;
While (X<>'.') And (NoS>0) Do Begin
   Append X to the end of S;
   Q:=Suff(S,P);
   NoS:=number of elements of Q;
   If the empty sequence is element of Q Then Res:=Slength;
   ReadLn(DataFile,X);
   Inc(Slength);
```

```
  End;
  WriteOut(Res);
```

在上述之演算法中有一個相當明顯的問題，即資料檔有時會太大以至於不能夠讀進記憶體中(除非你很明確知道你的機器中有多少剩餘之記憶體)。

幸運地是，我們並不需要將整個序列都讀進記憶體之中，讓我們觀察 Suff(Sx,P)包含了序列 S 與一個字母 x 並且可以由集合 Suff(S,P)計算出來，假設 Q=Suff(S,P)固定在執行 Next(Q,X)之前，則之後執行的 Q=Suff(SX,P)也將被固定，下列演算法即符合此一需求：

```
Procedure Next(Q,X);
  Begin
    Q1:=empty;
    Forall u in Q Do Begin
      If ux is a prefix of some primitives in P Then
        Begin
          include ux in Q1;
          If ux is equal to a primitive in P
            Then include the empty sequence in Q1
        End;
    End;
    Q:=Q1;
  End;
```

為了要改進此演算法，接著我們必須回答下列問題：

-序列 ux 是 P 中一些元件的字首嗎？

-序列 u 相等於 P 中的元件嗎？

假設下列的資料結構是元件 P 的集合：

```
Const
  MaxN=100;                          (* maximum possible number of primitives *)
  MaxL=20;                           (* maximum possible length of primitives *)
Var
  P:Array[1..MaxN,1..MaxL] Of Char;                  (* array of primitives *)
  L:Array[1..MaxN] Of Word;                          (* length of the primitives *)
```

讓我們描述一個序列 u，其為一個以(i,j)成對表示的 P 中的一個元件，所以 P[i]的字首包括了元件 P[i]中相等於 u 的前 j 個字母；而 i 則是 u 中這樣的索引的最後一個，要注意的是空的序列將以(1,0)來表示。我們將事先處理這個元件的集合並據之建一個表 T：

假設在 P[i][1..j]x 之字首 P 中沒有元件的話，T[i,j,x]將被設為 0，否則設為最後的索引值 k(P[k]中的字首為 P[i][1..j]x；P[i][1..j]表示包含於元件 P[i]中前 j 個成分之字母序列)

換句話來說，假設序列 u 被表示為(i,j)且 T[i,j,x]>0 則序列 ux 便為 P 中元件之字首；在這個狀況下，ux 是 P[T[i,j,x]]之字首且可以(T[i,j,x], j+1)來表示。程序 BuildTable 計算轉換表 T 並且建立一個陣列 Full，若且唯若表示為(i,j)的序列相等於 P 中的元件，則 Full[i,j]設為真；如此我

們即可以利用陣列 T 及 Full 很容易地將演算法 Next(Q,X)實施出來。}

```
Program Prefix;
Const
   MaxN=100;                          { maximum possible number of primitives }
   MaxL=20;                           { maximum possible length of primitives }
Var
   DataFile:Text;                     { file for the sequence to be examined }
   P:Array[1..MaxN,1..MaxL] Of Char;                  { array of primitives }
   L:Array[1..MaxN] Of Word;                          { length of the primitives }
   T:Array[1..MaxN,0..MaxL,'A'..'Z'] Of Byte;         { transition table }
   N,                                                 { number of primitives }
   ML:Word;                           { max of the length of the primitives }
   Res:Longint;                                       { length of the longest prefix }
   Full:Array[1..MaxN,1..MaxL] Of Boolean;
Type
   State=Array[1..MaxL+1] Of Record
                                i,j:Byte;
                             End;
Procedure Init;
Var M,i,j:Word;
   InFile:Text;
Begin
   Assign(InFile,'input.txt'); Reset(InFile);
   ReadLn(InFile,N);
   ML:=0;
   For i:=1 To N Do Begin
      ReadLn(InFile,L[i]);
      If L[i]>ML Then ML:=L[i];
      For j:=1 To L[i] Do Read(InFile,P[i][j]);
      ReadLn(InFile);
   End;
   Close(InFile);
   Assign(DataFile,'data.txt'); Reset(DataFile);
End;
Procedure BuildTable;{Global input variables: N, ML, P }
{Global output variables: T, Full }
Var
   i,i1,j,k:Word;
```

```
   X:Char;
Begin
   For i:=1 To N Do    {initialize the array Full}
      For j:=1 To ML Do Full[i,j]:=False;
   For i:=1 To N Do                  { compute T[i,0,x] }
      For X:='A' To 'Z' Do Begin
         k:=1;
         While (k<=N) And (P[k][1]<>X) Do Inc(k);
         If (k<=N) Then Begin
            T[i,0,X]:=k;
            Full[k,1]:=Full[k,1] Or (L[i]=1) And (P[i][1]=X);
         End Else
            T[i,0,X]:=0;
      End;
   For j:=1 To ML Do Begin
      For i:=1 To N Do Begin
         For X:='A' To 'Z' Do Begin { compute T[i,j,X] }
            If j>L[i] Then Begin
               T[i,j,X]:=0;
            End Else Begin
               i1:=T[i,j-1,P[i][j]];
               k:=1;
               While (k<=N) And
                        Not ((j+1<=L[k]) And (P[k][j+1]=X) And (i1=T[k,j-1,P[k][j]]))
                  Do Inc(k);
               If (k<=N) Then Begin
                  T[i,j,X]:=k;
                  Full[k,j+1]:=Full[k,j+1] Or (L[i]=j+1);
               End Else
                  T[i,j,X]:=0;
            End;
         End {for 'A'..'Z'};
      End {for i};
   End {for j};
End {BuildTable};
Procedure Next(Var NoS:Word; Var Q:State; X:Char; Var Complete:Boolean);
{Input: NoS is the number of prefixes in Suff(S,P),
            (Q[1].i,Q[1].j),...,(Q[NoS].i,Q[NoS].j) are the representatives of
```

```
            the prefixes in Suff(S,P),
            X is the actual element of the sequence to be examined.
  Output:NoS is the number of prefixes in Suff(SX,P),
            (Q[1].i,Q[1].j),...,(Q[NoS].i,Q[NoS].j) are the representatives of
            the prefixes in Suff(SX,P),
            Complete is True iff the empty sequence is in Q.}
    Var i,j,ii,newi,newj:word;
    Begin
      ii:=0; Complete:= False;
      For i:=1 To NoS Do Begin                                  { compute next state }
        newi:=T[Q[i].i,Q[i].j,X]; newj:=Q[i].j+1;
        If newi>0 Then Begin
          Inc(ii);
          Q[ii].i:=newi; Q[ii].j:=newj;
          Complete:=Complete Or Full[newi,newj];
        End;
      End;
      If Complete Then Begin
        Inc(ii); Q[ii].i:=1;Q[ii].j:=0;                {include the empty string}
      End;
      NoS:=ii;
    End {Next};
Procedure Process;{Global input variables: DataFile }{Global output variables: Res }
Var
    X:Char;                                          { the actual element of the sequence }
    Q:State;                                      { set of prefixes of primitives that are
                                                      suffixes of the sequence red so far }
    NoS:Word;                                         { number of the elements of Q }
    Slength:Longint;                                { length of the sequence red so far }
    Complete:Boolean;
Begin
    NoS:=1; Q[1].i:=1; Q[1].j:=0;                                  { initialize Q }
    Res:=0; Slength:=1;
    ReadLn(DataFile,X);
    While (X<>'.') And (NoS>0) Do Begin
      Next(NoS,Q,X,Complete);
      If Complete Then Res:=Slength;
      ReadLn(DataFile,X);
```

```
      Inc(Slength);
   End {While};
   Close(DataFile);
End {Process};
Procedure WriteOut(Res:Longint);
   Var OutFile:Text;
Begin
   Assign(OutFile,'output.txt'); Rewrite(OutFile);
   WriteLn(OutFile,Res);
   Close(OutFile);
End;
Begin
   Init;
   BuildTable;
   Process;
   WriteOut(Res);
End.
```

魔術

魔術矩陣(Magic Squares)

1	2	3	4
8	7	6	5

圖一：起始盤面(Initial configuration)

在魔術方塊成功之後 Bubik 先生又開發了它的平面版本，稱為魔術矩陣。此盤面含有八塊大小相同的方格子(見圖一)。在此任務中，我們考慮的版本中每一方格子的顏色均不相同，並由前八個正整數來表示。一個盤面可由顏色序列來表示，並從最左上角的方格子開始，然後依順時針方向來得到此顏色序列。舉例來說，圖一的盤面可由序列(1, 2, 3, 4, 5, 6, 7, 8)來表示。這個盤面是起始盤面(initial configuration)。

有三種基本的轉換可以使用，分別以字母'A'、'B'、'C'來表示如下：

'A'：上一列(top row)和下一列(bottom row)互換；

'B'：此矩陣向右環狀位移(right circular shift)一格；

'C'：中間四個方格子做順時針旋轉一格。

所有的盤面都可以用這三種基本轉換來產生。這些基本轉換的效果顯示在圖二上。方格外面的數字表示方格的位置。如果在位置 p 裡面的數字是 i，則這意謂著在轉換後，原來在位置 i 的方格會被移到位置 p。

子任務 A：你需要寫一個程式去求出一序列的基本轉換，使得圖一的起始盤面可以被轉換到某一特定的目標盤面(target configuration)。

子任務 B：如果你的轉換序列長度沒有超過 300，那麼此答案可額外獲得兩分。

A

	1	2	3	4	
	8	7	6	5	
	1	2	3	4	
	8	7	6	5	

B

	1	2	3	4	
	4	1	2	3	
	5	8	7	6	
	8	7	6	5	

C

	1	2	3	4	
	1	7	2	4	
	8	6	3	5	
	8	7	6	5	

圖二：基本轉換

輸入資料

輸入檔 INPUT.TXT 的第一行含有八個正整數，用來描述目標盤面。

輸出資料

你的程式必須寫入轉換序列長度 L 到輸出檔 OUTPUT.TXT 的第一行中。再接下來得 L 行中，你的程式必須依序寫出基本轉換序列的字母代號，每一行的第一個位置放一個字母。

工具

MTOOL.EXE 是放在任務目錄中的一個程式，它可以然你玩魔術矩陣遊戲。執行“mtool input.txt output.txt”就可以進行目標盤面和轉換序列的實驗。

輸入和輸出範例

INPUT.TXT
2 6 8 4 5 7 3 1

OUTPUT.TXT
7 B C A B C C B

圖三

MAGIC之參考解答

{下列以虛擬碼方式寫成的演算法產生將所有起始盤面遵循基本轉換後所變化的盤面：

```
Make the set Generated empty;
Make the set Disp empty;
Include the configuration Ini in Disp;
While Disp is not empty Do Begin
   take an element P out of Disp;
   for all basic transformation C Do Begin
      let Q be the configuration obtained by applying C to P;
      If Q is not in the set Generated Then Begin
         include Q in the set Generated;
         include Q in the set Disp;
      End
   End
End;
```

假設盤面Q已是欲達成之目標盤面則可以停止。現在就讓我們研究Generated實施的運作情形：所有不同盤面的數目是 8!=40320，若是將其全部儲存於陣列中未免太大了，我們可以利用函數 Rank 來克服此問題，其對應到 0..8!-1 個盤面中的一個。我們可以定義 Rank(Q)為在 Q 之前依據非拼音文字所排列 1..8 不同的盤面數而得之函數。

讓我們觀察每一個基本轉換 C，其必然只有一個單一的倒轉。也就是說對任一個盤面 Q 而言，假設 C 轉換 Q 到 P 則其倒轉轉換則是 P 到 Q；相反地，假設 C 的倒轉轉換為 P 到 Q，則 C 轉換即為 Q 到 P。這些基本轉換的倒轉有以下三種：

A:即 A 本身；

B:此矩陣向左環狀位移(right circular shift)一格；

C:中間四個方格子做逆時針旋轉一格。

假設盤面 Q 是在演算法中利用轉換法 C 將盤面 P 轉換而成的，則基本轉換的序列將轉換起始盤面為 Q，而其最後一個動作是 C；假使我們知道 C 及 Q，則我們可以利用 C 的倒轉轉換計算 P。陣列 Last 之資料結構為 Array[0..8!-1] Of Char，我們使用陣列 Last 共有兩個目的，若且唯若盤面 Q 未曾經過轉換則 Last[Rank(Q)] =' '；若 Q 是盤面 P 經過 C 轉換後所得，則 Last[Rank(Q)]將被設為 C。下述的連結是由倒轉轉換所提供，我們可以對目標盤面 T 利用轉換將整個序列予以組合。

```
S:='';                    (* string S is set to empty          *)
While T <> Ini Do Begin
   X:=Last[Rank(T)];
   S:=X+S;                (* append X to the left end of S *)
   Apply_1(T,X,P);    (* Apply the inverse of X to T     *)
   T:=P;                  (* link to backward                    *)
End (* While *);
```

我們使用佇列來執行分配者 Disp(也就是說利用佇列之先進先出的策略)，各項目將按其插入之順序依次出現；任何一個簡單的實驗將可展現這個演算法所產生之序列的最大長度是 22。

此分配者佇列的實施方法提供了一個最佳解。因為對每一個盤面 T 而言，此演算法產生了一個基本轉換的最短序列；我們將利用歸納最佳解之長度以證明之。先讓我們以 l(T)來表示對盤面 T 以演算法所產生之序列的長度，假設在執行演算法的期間，佇列包含盤面 T1 到 Tk，其中 T1 為 head，則下列兩個條件必須予以固定：

1) l(T1)<=...<=l(Tk)

2) l(Tk)<=l(T1)+1

我們將利用歸納佇列運作數目的方法來證明此問題。最開始時由於只有起始盤面在佇列中且 l(Ini)=0，故上述式子本身不會有所改變；假使 head T1 在佇列中被釋放出來，則 T2 將成為新的 head，此時我們有 l(Tk)<=l(T1)+1<= l(T2)+1，而剩餘的不等式並不會有所影響。有另外一個現象即當 T1 被佇列所釋放時，此時新的盤面 Q(即將 T1 利用 C 轉換所得之盤面)將被放進佇列之中；此時 l(Q)=l(T1)+1，因此不等式固定第一、二項後成為：l(Tk)<=l(T1)+1=l(Q)及 l(Q)=l(T1)+1<=l(T2) +1。

由前述可以得知假設盤面依 T1,...,Tn 的順序加入到佇列中則 l(T1)<=...<=l(Tn)。

令 T(k)表所有盤面 Q 之集合，同時 k 表示利用基本轉換由 Ini 到 Q 的最小長度，此最佳化的證明即可利用對 k 之歸納而得，以下即為此證明之說明：

T(1)的組成是那些可以對 Ini 做基本轉換所得之盤面，這顯然地固定 k 必須等於 1；假設令所有的 l<k，再令 Q 表示 T(k)之盤面，則必然有一 P 為 T(k-1)之盤面，且其是利用基本轉換 C 將 P 轉換為 Q。依據歸納之假設 l(P)=k-1，P 被演算法所產生出來時即被插入到佇列之中；當演算法將 P 由佇列中被釋放時，同時檢查在佇列中 Q 是否已被放入佇列中，若是則累加其次數。假設 Q 是新的盤面則演算法將自動產生 Q，同時 l(Q)=l(P)+1=k；假設 Q 已被產生過了，

則 l(Q)<=l(P)=k-1，同時其性質亦不變。}

```
Program Magic;
Const
   Size=8;         { Size of the sheet }
   M     =40320; { =Size! }
Type
   Trans =Array[1..Size] Of 1..Size;
   Config=Array[1..Size] Of 1..Size;
Const
   BT    :Array['A'..'C'] Of Trans=((8,7,6,5,4,3,2,1), { basic transformations }
                                    (4,1,2,3,6,7,8,5),
                                    (1,7,2,4,5,3,6,8));
   BT_1:Array['A'..'C'] Of Trans=((8,7,6,5,4,3,2,1), { inverses of the basic }
                                    (2,3,4,1,8,5,6,7), { transformations          }
                                    (1,3,6,4,5,7,2,8));
   Ini :Config=(1,2,3,4,5,6,7,8);                  { the initial configuration }
Var
   T       :Config;                                        { the target configuration }
   Answer:String;                { the solution sequence of basic transformations }
   Fact :Array[0..Size] Of Longint;                { array of factorial values }
   Last :Array[0..M] Of Char;
                    { Last[Rank(T)] is the last character of a sequence of basic }
            { transformations that transforms the initial configuration to T. }
                          { If Last[Rank(T)]=' ' then T has not been generated. }
Procedure ReadInput;
{ Global output variable: T }
   Var InFile:Text;
      i:Word;
   Begin
      Assign(InFile,'input.txt'); Reset(Infile);
      For i:=1 To Size Do Read(Infile,T[i]);
      Close(Infile);
   End { ReadInput };
Procedure ComputeFact;     { Computes the factorial values }
   Var i:Word;
   Begin
      Fact[1]:=1;Fact[0]:=1;
      For i:=2 To Size Do
```

```
        Fact[i]:=i*Fact[i-1];
    End;
Function Rank(Const P:Config): Word;
{ Rank(P) is the number of permutations that precedes P }
{ according to the lexicographic ordering.                    }
    { Global input variables: Size, Fact }
    Var Res,l,i,j:Word;
    Begin
      Res:=0;
      For i:=1 To Size Do Begin
        l:=0;                    { l is the number of elements of P in positions }
                                   { 1..i-1 that are less than P[j]                  }
        For j:=1 To i-1 Do
          If P[j]<P[i] Then Inc(l);
        { Keeping fixed the first i-1 elements of P there can be (P[i]-1-l) }
        { numbers that are less than P[i] in position i in permutations.    }
        { The number of permutations Q such that the first i-1 elements     }
        { are the same as in P but Q precedes P in the lexicographic        }
        { ordering is (P[i]-1-l)*Fact[Size-i].                              }
        Res:=Res+(P[i]-1-l)*Fact[Size-i];
      End { For };
      Rank:=Res;
    End { Rank };
Procedure Apply(Const T:Config; X:Char; Var R:Config);
{ R is obtained by applying the basic transformation X }
{ to the configuration T                                  }
    Var i:Word;
    Begin
      For i:=1 To Size Do R[i]:=T[BT[X][i]];
    End { Apply }
Procedure Apply_1(Const T:Config; X:Char; Var R:Config);
{ R is obtained by applying the inverse of the basic }
{ transformation X to the configuration T                 }
    Var i:Word;
    Begin
      For i:=1 To Size Do R[i]:=T[BT_1[X][i]];
    End {Apply_1};
Function Equal(Const R,T:Config): Boolean;
```

```
{ Checks equality of the configurations R and T }
   Var i:Word;
   Begin
      i:=1;
      While (i<=Size) And (R[i]=T[i]) Do Inc(i);
      Equal:= i>Size;
   End { Equal };
Procedure Generate(Const T: Config);
{ Generates a sequence of basic transformations that transforms the          }
{ initial configuration to T. Last[Rank(T)] will be the last element of the sequence. }
{ Global input-output variable: Last }
   Const
      Qs=7000;    { Queue size }
   Var
      Queue:Array[0..Qs-1] Of Config;
      NotFound:Boolean;
      Head,Tail:Word; { head and tail of the queue }
      R,S: Config;
      X: Char;
   Procedure InitGener;
   Var i:Word;
   Begin
      For i:=0 To M Do Last[i]:=' '; { initialize }
      Last[0]:='.';                          { 0=Rank(Ini), sentinel }
   End;
   Procedure InitQueue;
   { initialize the queue }
   Begin
      Head:=0; Tail:=1;
      Queue[0]:=Ini;                          { put Ini into the queue }
   End { InitQueue} ;
   Procedure Enqueue(Const Q:Config);
   Begin
      Queue[Tail]:=Q;
      Inc(Tail); If Tail=Qs Then Tail:=0;
   End { Enqueue };
   Procedure Dequeue(Var Q:Config);
   Begin
```

```
    Q:=Queue[Head];
    Inc(Head); If Head=Qs Then Head:=0;
  End { Dequeue };
  Function NotMember(Const Q:Config; X:Char):Boolean;
  { Checks membership of Q in the set of generated configurations.          }
  { If it is not generated then marks it as generated by setting the value }
  {   of Last[Rank(Q)] to X.} { Global input-output variable: Last }
  Var RankQ:Word;
  Begin
    RankQ:=Rank(Q);
    If Last[RankQ]=' ' Then Begin
      NotMember:=True;
      Last[RankQ]:=X;
    End Else
      NotMember:=False;
  End { NotMember };
  Begin { Generate }
    InitGener;
    InitQueue;
    NotFound:=True;
    While NotFound Do Begin
      Dequeue(R);
      For X:='A' To 'C' Do Begin              { apply all basic }
        Apply(R, X, S);                       { transformations to R }
        If NotMember(S,X) Then Begin          { S is a new configuration }
          If Equal(T,S) Then Begin            { T=R*C decomposition found }
            NotFound:= False;
            Break;                                { exit the loop }
          End;
          Enqueue(S);
        End { If new tr. };
      End { For j };
    End { While };
  End { Generate };
Procedure Compose(Const T: Config; Var S:String);
{ Composes the sequence of basic transformations from the array Last }
{ following the link provided by the inverse transformation.          }
  { Global input variable: Last }
```

```
  Var
    RankQ:Word; X:Char;
    P,Q   : Config;
  Begin
    Q:=T;
    RankQ:=Rank(Q);
    S:='';
    While RankQ <> 0 Do Begin { while Q<>Ini                    }
      X:=Last[RankQ];
      S:=X+S;                  { append X to the left end of S }
      Apply_1(Q,X,P);          { Apply the inverse of X to Q   }
      Q:=P;                    { link to backward              }
      RankQ:=Rank(Q);
    End { While };
  End { Compose };
Procedure WriteOut;    { Global input variable: Answer }
  Var OutFile:Text;
    L,i:Word;
  Begin
    Assign(OutFile,'output.txt'); Rewrite(OutFile);
    L:=Length(Answer);
    WriteLn(OutFile,L);
    For i:=1 To L Do   WriteLn(OutFile,Answer[i]);
    Close(OutFile);
  End { WriteOut };
Begin { Program }
  ReadInput;
  ComputeFact;
  Generate(T);
  Compose(T, Answer);
  WriteOut;
End.
```

{ Scientific Committee IOI'96 提供}