1999 年 IOI 試題與參考解答

第一天第一題：小花店(LITTLE SHOP OF FLOWERS)

你想把你的花店的櫥窗佈置成最賞心悅目的樣子。你有 F 束花，每束的種類都不一樣，而且至少有足夠的花瓶，排成一列。花瓶是固定在架子上的，且從 1 到 V 連續編號， V 是花瓶總數；編號由左而右，花瓶 1 在最左邊，而花瓶 V 在最右邊。每束花有一個唯一的整數編號，從 1 到 F。花束的編號是重要的：它影響到花束出現在花瓶中的必要順序，即如果 i<j 的話，則插花束 i 的花瓶必須在插花束 j 的花瓶的左側方向上。舉例而言，假設你有一束 azaleas（編號為 1）、一束 begonias（編號為 2）、和一束 carnations（編號為 3）。現在，要把它們插進花瓶裡，並確保原花束編號順序。Azaleas 必須插在 begonias 左側方向的花瓶中，而 begonias 必須插在 carnations 左側方向的花瓶中。如果花瓶比花束多的話，就有多餘的空花瓶。每個花瓶只能插一束花。

每個花瓶都有其特色（花也一樣）。因此，把某一束花插在某一個花瓶中會造成一種美感價值，此值以一個整數表示。這些美感值用一個表列出，如下所示。空的花瓶的美感值為 0。

		花瓶				
		1	**2**	**3**	**4**	**5**
花束	**1 (azaleas)**	7	23	-5	-24	16
	2 (begonias)	5	21	-4	10	23
	3 (carnations)	-21	5	-4	-20	20

按照此表，舉例而言，azaleas 放在花瓶 2 裡會很好看，而放在花瓶 4 中則難看得很。

為了達到最佳的櫥窗效果，你必須找到最大的美感值的和，且插花的方式要符合花束的必要順序。如果超過一種方式可以達到最大美感總值，則任何一種都可被接受。你只要產生一種插花方式即可。

假設 (ASSUMPTIONS)

- $1 \leqq F \leqq 100$, 其中 F 是花束總數。花束編號為 1 到 F。
- $F \leqq V \leqq 100$, 其中 V 是花瓶總數。
- $-50 \leqq A_{ij} \leqq 50$, 其中$A_{ij}$是把花束i插到花瓶j裡所得到的美感值。

輸入（INPUT）

輸入是一個檔名為 flower.inp 的文字檔。

第 1 行含有兩個數字：F 和 V。

接下來的F行：每一行含有V個整數，使得A_{ij}為輸入檔第i+1 行的第j個數字。

輸出（OUTPUT）

輸出必須是一個名為 flower.out 的文字檔，其中包含 2 行：

- 第 1 行含有你所佈置的美感總值。
- 第 2 行必須以 F 個數字表示出你的佈置，此行的第 k 個數字為花束 k 所插的花瓶編號。

舉例 (EXAMPLE)

flower.inp

```
3 5
7 23 -5 -24 16
5 21 -4 10 23
-21 5 -4 -20 20
```

flower.out

```
53
2 4 5
```

小花店(LITTLE SHOP OF FLOWERS)參考解答
試作者：黃俊棋(IOI'99 銅牌獎)

/*這是標準的 DP 題*/

```
#include <stdio.h>

int table[110][110];
int which[110][110];
int F, V;

void count(void)
{
   int i, j, k;
   int max, now;
   FILE *outfp;
   outfp=fopen("flower.out","w");
   for(j=F-2;j>=0;j--)
   { for(i=0;i<(V-F+j+1);i++)
      { max=table[j][i]+table[j+1][i+1];
         which[j][i]=i+1;
         for(k=i+2;k<(V-F+j+2);k++)
         { if((table[j][i]+table[j+1][k])>max)
            { max=table[j][i]+table[j+1][k];
               which[j][i]=k; } }
         table[j][i]=max; } }
   max=table[0][0];
   now=0;
   for(i=1;i<(V-F+j+1);i++)
      if(table[0][i]>max)
      { max=table[0][i];
         now=i; }
   fprintf(outfp,"%d\n",max);
   for(j=0;j<F;j++)
   { fprintf(outfp,"%d ",now+1);
      now=which[j][now]; }
   fprintf(outfp,"\n");
   fclose(outfp);
}

void init(void)
{
   int i, j;
```

```
   FILE *infp;
   infp=fopen("flower.inp","r");
   fscanf(infp,"%d",&F);
   fscanf(infp,"%d",&V);
   for(j=0;j<F;j++)
      for(i=0;i<V;i++)
      { fscanf(infp,"%d",&table[j][i]); }
   fclose(infp);
int main(void)
{
   init();
   count();
   return 0;
}
```

第一天第二題：隱藏碼 (HIDDEN CODES)

現在有一組字碼和一篇文章。該篇文章被認為包藏著一個訊息，由隱藏在文章中的各個字碼以一種奇特的（且可能是曖昧的）方式構成。

各個字碼和該篇文章皆為由英文大小寫字母構成的序列。大小寫字母是不同的。一個字碼的*長度(length)*以平常的方式來定義：例如字碼 **ALL** 的長度為 3。

字碼中的各個字母未必連續出現在文章中。舉例而言，字碼 **ALL** 總是出現在文章中的某一個子序列中，其形式為 **A***u***L***v***L**，其中 *u* 和 *v* 是任意的（也可能是空的）字母序列。我們稱 **A***u***L***v***L** 為 **ALL** 的一個*涵蓋序列(covering sequence)*。一般而言，一個字碼的*涵蓋序列*定義為文章中的一個子序列，其首字母與尾字母和字碼的首尾字母相同，且由該子序列中刪除某些（可能為零）個字母之後就可得到該字碼。應該注意的是，一個字碼可以出現在一個或以上的涵蓋序列中，也可能根本不出現在文章中。此外，一個涵蓋序列也可能同時是好幾個字碼的的涵蓋序列。

一個涵蓋序列由它在文章中的起點（其首字母的位置）和終點（其尾字母的位置）來識別。（文章的首字母在位置 1 上。）我們說兩個涵蓋序列（例如c_1和c_2）*不相重疊(do not overlap)*，如果c_1的起點大於(>)c_2的終點，或是c_2的起點大於c_1的終點。否則的話，我們就說這兩個涵蓋序列*重疊(overlap)*。

為了要由文章中取出隱藏的訊息，你的任務是找到一個*解(solution)*。一個*解*由一組*項目(items)*構成，每個項目均包含一個字碼以及該字碼的一個涵蓋序列，並滿足下列各條件：

a) 各個涵蓋序列互不重疊；
b) 一個涵蓋序列的長度不會超過 1000；
c) 各個字碼的長度總和達到最大值。（每個項目貢獻它所涵蓋的字碼長度，加成總和。）

如果有多個解，你只需要找出其中一個即可。

假設（ASSUMPTIONS）

- 1≦N≦100, 其中 N 是字碼總數。
- 字碼長度最多為 100 個字母。
- 1≦文章長度≦1,000,000 個字母。
- 我們說字碼 w 的涵蓋序列 c 是*右側極短(right-minimal)*，如果沒有任何 c 的嚴格字首也是 w 的涵蓋序列，所謂 c 的*嚴格字首(proper-prefix)*指的是不等於 c 本身的一個 c 的起始子序列。例如，對字碼 **ALL** 而言，**AAALAL** 是一個*右側極短*

的涵蓋序列，而 **AAALALAL** 雖然也是一個涵蓋序列，但並非右側極短。

- 在所給的文章中保證下面兩點：
  a) 在文章的任何位置，包含該位置的互相重疊的*右側極短*涵蓋序列，總數不會超過 2500 個；
  b) *右側極短*涵蓋序列的總數不會超過 10,000 個。

輸入（INPUT）

這題有兩個輸入檔：**words.inp** 與 **text.inp**。**words.inp** 含有一串字碼，而 **text.inp** 含有那篇文章。

- words.inp 檔的第 1 行含有數值 N。接下來的 N 個輸入行各含有一個字碼，字碼為一串字母，其中不含空白。各字碼由其在 words.inp 中出現的順序來識別：也就是使用整數 1 到 N 作為字碼的編號。
- text.inp 檔含有一個字母序列（以一個 end-of-line 字元加一個 end-of-file 字元結束）。這個檔案中沒有空白。

輸出（OUTPUT）

輸出必須是一個名為 **codes.out** 的文字檔。

- 第一行必須是由你的解所得到的總和。
- 接下來的每一行都是你的解中的一個項目：每行由 3 個整數 i, s, e 所構成。其中 i 是該涵蓋序列所含字碼的編號，而 s 和 e 分別是涵蓋序列的起始位置與結束位置。第一行之後的各輸出行順序不重要。

舉例 (EXAMPLE)

words.inp

```
4
RuN
RaBbit
HoBbit
StoP
```

text.inp

```
StXRuYNvRuHoaBbvizXztNwRRuuNNP
```

codes.out

```
12
2 9 21
1 4 7
1 24 28
```

（注意：由此解中可能得到的隱藏訊息是：RuN RaBbit RuN。（另一個解可能產生 RuN HoBbit RuN。）提醒你此訊息不可出現於輸出檔中。）

隱藏碼 (HIDDEN CODES)參考解答
試作者：孫維孝(IOI'99 銅牌獎)

```
/*核心演算法是 DP，但是因為資料龐大，所以要作一點手腳。*/

/*----ALLOC_MEMORY 設定：使用動態配置答案的記憶體----*/
/*我比賽的時候就是用動態配置，但是資料太多了，所以後來改用 huge 陣列*/
//#define ALLOC_MEMORY

/*----CONSOLE_OUTPUT 設定：答案輸出到螢幕----*/
/*debug 用*/
//#define CONSOLE_OUTPUT

/*----DEBUG 設定：計算執行時間----*/
/*計算效率用*/
//#define DEBUG

/*----TIME_LIMIT 設定：限制執行時間----*/
/*為避免執行時間超過用，會影響一點點效率*/
//#define TIME_LIMIT 1600

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if defined(DEBUG) || defined(TIME_LIMIT)
#include <time.h>
#endif

/*儲存可用的單字(words.inp)一律使用動態配置，因為沒問題，而且比較快*/
/*不使用 struct 的原因是每個陣列獨立的存取速度比較快*/
int       words_total = 0;              // 單字總數
char** words_str = NULL;              // 字串陣列，比對用
long** words_previous = NULL;   // 儲存上一次使用某一字串的某一字母時的位置
int*      words_max = NULL;           // 儲存每個字串的長度減一

/*輸入最大到 1M，所以使用 cyclic 的 queue，並且擁有一份 image*/
/*[0]~[1000]是 image，儲存 1001 個是預防萬一，以免存取[-1]的值*/
char    input_buffer[2002];        // 輸入的字母，比對用
long    input_score[2002];         // 到該位置最高分數
int     input_max[2002];             // 最高分數對應到的涵蓋字串的 index
int     input_index = 1001;        // 要寫入的位置
long    input_read  =  0L;            // 已經讀取的字元數，我本來只用 int 所以全溢位了

/*儲存涵蓋字串，不是全部，因為有 cut*/
/*因為左右極短都考慮了，所以最多只有 10000 個，[0]是虛擬的*/
/*有動態配置和 huge 陣列兩種，當然動態配置比較快，
   但是因為每個元素最多只會使用兩次(存入，輸出)，所以影響不大*/
```

```
#ifdef ALLOC_MEMORY
int*    ans_index = NULL;           // 使用的單字的 index
long* ans_left = NULL;              // 左端
long* ans_right = NULL;             // 右端
int*    ans_previous = NULL;        // 上一個可用不重疊的涵蓋字串的 index
#else
int     huge        ans_index[10001];
long    huge        ans_left[10001];
long    huge        ans_right[10001];
int     huge        ans_previous[10001];
#endif
int     ans_count = 1;   // 因為[0]虛擬，所以從[1]開始

/* 副程式 read_word
        顧名思義，初始化 words_*陣列群，並從 words.inp 讀入資料*/
void read_words(void)
{
int         i, j;
char        strin[256];         // 讀入字串暫存區

freopen("words.inp", "r", stdin);
scanf("%d\n", &words_total);        // 讀入單字數並且去掉結尾的換行字元
words_str = (char**)malloc(sizeof(char*) * words_total);
words_previous = (long**)malloc(sizeof(long*) * words_total);
words_max = (int*)malloc(sizeof(int) * words_total);
for(i = 0; i < words_total; i++){
   gets(strin);
   // 初始化 words_max
   words_str[i] = (char*)malloc(sizeof(char) * ((words_max[i] = strlen(strin)-1)+2));
   strcpy(words_str[i], strin);        // 初始化 words_str
   words_previous[i] = (long*)malloc(sizeof(long) * (words_max[i] + 1));
   for(j = words_max[i]; j >= 0; j--)
        words_previous[i][j] = -1L; // 將每一個字元的前次出現位置都設為-1
}
}/*read_word*/

/* 副程式 update
        每讀入一個字元就呼叫 update()更新 table*/
void update(void)
{
int    i; // 這是目前處理的單字的 index
int    pointer;   // 逐字比對時，字串的 index 存在這
int    limit_index; // 涵蓋字串最長到 1000，極限的 index(queue)存在這
int    index;   // 逐字比對時 queue 的 index 存在這
long nowa_index;   // index 實際對應到的字元的 index
long score;
```

```
for(i = words_total - 1; i >= 0; i--){ //從後面做回來，這樣 words_total 只存取一次
  /*要取右側極短*/
  if(input_buffer[input_index] != words_str[i][words_max[i]])
 // 如果單字最末字元跟現在的輸入不一樣就不是右側極短
    continue;

  pointer = words_max[i]; // 從最末字元開始比對，
  index = input_index; //不從倒數第二個開始可以避免單字只有一個字母會爆掉
  limit_index = input_index - 1000;          // 最長比對 1000 個字元
  for(nowa_index = input_read; index != limit_index; index--, nowa_index--){
  // 逐字比對
    if(input_buffer[index] != words_str[i][pointer])    // 這個字元不合
      continue;    // 比對下一個字元
    /*在這裡檢查左側極短*/
    if(words_previous[i][pointer] == nowa_index){
     // 這個位置，這個單字，這個字元已經使用過了，
      break;  // 這不是左側極短，不用比了！
    }else{
       words_previous[i][pointer--] = nowa_index;          // 更新
      if(pointer < 0){             // 全部比對完了
        score = input_score[index - 1] + words_max[i] + 1L;
        // 計算如果使用這個涵蓋字串，可以得到的分數
        if(score > input_score[input_index]){ // 比較大才用
#ifdef ALLOC_MEMORY
          ans_index = (int*)realloc(ans_index, sizeof(int) * (ans_count + 1));
          ans_left = (long*)realloc(ans_left, sizeof(long) * (ans_count + 1));
          ans_right = (long*)realloc(ans_right, sizeof(long) * (ans_count + 1));
          ans_previous = (int*)realloc(ans_previous, sizeof(int) * (ans_count + 1));
#endif
          ans_index[ans_count] = i + 1;   // Convert from 0-based to 1-based index
          ans_left[ans_count] = nowa_index;
          ans_right[ans_count] = input_read;
          ans_previous[ans_count] = input_max[index - 1];
          // 更新 table
          input_score[input_index - 1001] = input_score[input_index] = score;
          input_max[input_index - 1001] = input_max[input_index] = ans_count++;
           // 和涵蓋字串數
        }/* if(score > origin) */
        break;  // 不用比了！
      }/* if(compare finished) */
    }/*左側極短*/
  }/*for(each charact in input buffer)*/
}/*for(each word can be used)*/
}/*update()*/

/* 程式進入點 */
void main(void)
```

```
{
int eadin; // 讀進來的字元
int exti;
#if defined(DEBUG) || defined(TIME_LIMIT)
   clock_t start = clock();              // 取得時間
#endif

   read_words();     // 讀取可用的單字
#ifdef ALLOC_MEMORY
   /* 初始化 ans_* */
   ans_index = (int*)malloc(sizeof(int));
   ans_left = (long*)malloc(sizeof(long));
   ans_right = (long*)malloc(sizeof(long));
   ans_previous = (int*)malloc(sizeof(int));
#endif
   freopen("text.inp", "r", stdin);
#ifndef CONSOLE_OUTPUT
   freopen("codes.out", "w", stdout);
#endif
   for(input_read = 1; (readin = getchar()) != EOF; input_read++){
      input_buffer[input_index - 1001] = input_buffer[input_index] = (char)readin;
      update();            // 更新 table
#ifdef TIME_LIMIT
      if(clock() - start >= TIME_LIMIT)            // 檢查已執行時間
         break;   // 輸出目前最佳解
#endif
         nexti = (input_index == 2001) ? 1001 : (input_index + 1);
         input_score[nexti - 1001] = input_score[nexti] = input_score[input_index];
         input_max[nexti - 1001] = input_max[nexti] = input_max[input_index];
         input_index = nexti;
   }/*for(each character contained in text.inp)*/

   /* 輸出答案 */
   printf("%ld\n", input_score[input_index]);
   for(nexti = input_max[input_index]; nexti; nexti = ans_previous[nexti])
   // 從後面輸出回來，測試資料的參考解答也是這麼輸出的唷
      printf("%d %ld %ld\n", ans_index[nexti], ans_left[nexti], ans_right[nexti]);

#ifdef DEBUG
/* 顯示總執行時間 */
#ifndef CONSOLE_OUTPUT
      freopen("CON", "w", stdout);
#endif
      printf("%d\n", clock() - start);
#endif
}
/*   Written by wssuan(Wei-Shiau Suen) '2000
      Question to wssuan@csie.nctu.edu.tw   */
```

第一天第三題：地下城市(UNDERGROUND CITY)

你被關在卡巴多西亞 (Cappadocia) 的地下城市之一，在黑暗中想找出路時，正好找到城市的地圖。不幸地，地圖上沒有標示你在那裏。你的任務就是探索該城市，找出你發現地圖時的位置。

城市的地圖是單位正方形所形成的長方形。每個正方形可能是開放正方形，用字母 'O' 標示；也可能是牆的一部份，用 'W' 表示。地圖上有指出北方。 很幸運地，你手上有指南針，所以你可以根據地圖按正確的方向行走。剛開始時，你是在一個開放正方形上。

開始時叫用 start 程序(或函數)，沒有參數。你可以叫用 look 和 move 程序(或函數)探索該城市。

你可以叫用 look(*dir*)函數提出問題。此處 *dir* 代表你正在看的方向，可以是 'N', 'S', 'E', 'W' 字母之一，分別代表 北，南，東，西。現在假設參數 *dir* 是 'N'，如果你的北方的正方形是開放正方形,則回答會是字母 'O'；反之如果它是牆，則回答會是 'W'。同樣地，可以看而且得到其他相鄰正方形的資訊。

叫用 move(*dir*)，你可以走進四個相鄰正方形之一，此處 *dir* 代表你走的方向 o 你只可以走到開放正方形。試圖走進牆會是嚴重錯誤。 由任意開放正方形開始，可以走到該城市的任意開放正方形。

你必須叫用最少次數的 look(*dir*)來找出你發現地圖時的位置，一旦你發現那個位置，你必須叫用 finish(*x, y*) 報告出來，此處 *x* 是該位置的水平 (西-東) 座標,*y* 是垂直 (南-北) 座標。

假設(ASSUMPTIONS)

- $3 \le U \le 100$ 此處 *U* 是地圖的寬度，也就是地圖在水平 (西-東) 方向的長度，用正方形的個數表示。
- $3 \le V \le 100$ 此處 *V* 是地圖的高度，也就是地圖在垂直 (南-北) 方向的長度，用正方形的個數表示。
- 城市由牆圍繞，此牆包含在地圖中。
- 城市的西南角的座標是 (1, 1) 而且東北角的座標是 (*U, V*)。

輸入(INPUT)

輸入是文字檔，叫做 **under.inp**

- 第一行包含兩個數字: *U* 和 *V*
- 再下來 *V* 行的每行包含地圖水平方向的一列。每行包含 *U* 個字元，使得第 (*V*-*y*+2) 行的第 *x* 個字元代表地圖 (*x, y*) 位置的資訊：字母 'W' 代表牆，或字母 'O' 代表開放正方形。這些行的資料沒有空白在中間。

輸出(OUTPUT)

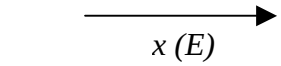
不會產生輸出檔。你的程式找出的結果必須叫用 finish(*x, y*) 報告。

舉例(EXAMPLE)

under.inp

一個可能的互動，正確地叫用 finish 結束:

互動(Interaction):	
start()	
look('N')	'W'
look('E')	'O'
move('E')	
look('E')	'W'
finish(3,5)	




```
5 8
WWWWW
WWWOW
WWWOW
WOOOW
WOWOW
WOOWW
WWOOW
WWWWW
```

Pascal 程式者的指示(INSTRUCTIONS FOR PASCAL PROGRAMMERS)

Have the following in your source file:
  uses undertpu;
This tpu will provide the following:
  procedure start; { must be called first }
  function look (dir:char):char;
  procedure move (dir:char);
  procedure finish (x,y:integer); { must be called last }

C/C++ 程式者的指示(INSTRUCTIONS FOR C/C++ PROGRAMMERS)

下列東西在你的原始檔中:
    #include "under.h"
這會提供你下列宣告:
  void start (void); /* 必須首先叫用 */
  char look (char);
  void move (char);
  void finish (int,int); /* 必須最後叫用 */

產生一個計畫 (project) 叫做 under, 它應該包括你的程式和互動庫存函數叫做 under.obj; 你必須用整合發展環境中的 project 選單, 而且選擇 open 選項來產生計畫, 然後用 add item 選項加入你的原始檔 (under.c 或 under.cpp)

和 under.obj, 用 LARGE 記憶體模式的編譯器選項 (注意: 這超越比賽規則提到的記憶體模式)。

地下城市(UNDERGROUND CITY)參考解答
試作者：黃俊棋(IOI'99 銅牌獎)

```
/*這題題目上就告訴了你解法
1.要先看某個方向能不能走
2.然後走走看
3.把不可能的刪掉
4.重覆 1.到 3.直到只剩一種可能
所以應該大家解法都是一樣的只是 1.挑某個方向的方法不一樣
偶挑方向的方法就是最簡單的先從面對方向的北邊開始
如果沒有路就挑東邊然後是南邊西邊
完全沒有最佳化也沒有錯誤檢查
所以程式應該很好懂*/

#include <stdio.h>
#include "under.h"

int U, V;
int table[110][110];
char map[110][110];
char dir[4];
int dx[4], dy[4];
int remain;

int onboard(int x,int y)   //是檢查有沒有在地圖上囉
{
   if(x<0 || x>=U)
      return 0;
   if(y<0 || y>=V)
      return 0;
   return 1;
}

void del(int x,int y,char which)   //這就是 3. 把不可能的刪掉
{
   int i, j;
   for(j=0;j<V;j++)
      for(i=0;i<U;i++)
      { if(table[j][i]==1)
         { if(onboard(i+x,j+y)==1)
            { if(map[j+y][i+x]!=which)
               { remain--;
                  table[j][i]=0; } }
            else
            { remain--;
```

```
                table[j][i]=0; } } }
}

void loop(int x,int y,int d)     //這就是所謂的 4.
{
   int i;
   char now;
   for(i=0;i<4;i++)
   { now=look(dir[(i+d)%4]);
      del(x+dx[(i+d)%4],y+dy[(i+d)%4],now);
      if(remain<=1)
      { break; }
      if(now=='O')
      { move(dir[(i+d)%4]);
         loop(x+dx[(i+d)%4],y+dy[(i+d)%4],(i+d)%4); }
      if(remain<=1)
      { break; } }
}

void count(void)
{
   int i, j;
   loop(0,0,0);
   if(remain==1)   //如果只剩下一個位置是對的就呼叫 finish()
   { for(j=0;j<V;j++)
      { for(i=0;i<U;i++)
         { if(table[j][i]==1)
            { finish(i+1,j+1);
               return; } } } }
}

void init(void)
{
   int i, j;
   FILE *infp;
   infp=fopen("under.inp","r");
   fscanf(infp,"%d",&U);
   fscanf(infp,"%d",&V);
   fgets(map[0],200,infp);
   for(j=V-1;j>=0;j--)
      fgets(map[j],200,infp);
   fclose(infp);
   for(remain=0,j=0;j<V;j++)     //一開始應該不能在牆裡吧
      for(i=0;i<U;i++)
      { if(map[j][i]=='O')
         { table[j][i]=1;
            remain++; } }
   dir[0]='N';
   dir[1]='E';
   dir[2]='S';
```

```
    dir[3]='W';
    dx[0]=0, dy[0]=1;
    dx[1]=1, dy[1]=0;
    dx[2]=0, dy[2]=-1;
    dx[3]=-1, dy[3]=0;
    start();   //別忘了呼叫 start()
}

int main(void)
{
    init();
    count();
    return 0;
}
```

第二天第一題：交通號誌(TRAFFIC LIGHTS)

Dingilville 市裡的交通號誌燈用一種不尋常的方式運作。市內有許多路口以及把路口相互連接的道路，而任意兩個不同路口之間最多只有一條道路直接連接。不會有路把路口連回原路口。一條道路行駛所需時間，兩個方向是相同的。在每一個路口都有一個交通號誌燈，在任何時刻其燈色不是藍色就是紫色。各號誌燈的顏色週期性的變換：藍燈亮一段號誌時間後即變成紫燈亮另一段的號誌時間。車輛可以由一路口出發駛向另一路口，若且唯若(if and only if)該二路口的號誌燈號在車輛駛離路口時是相同的。若車輛正好在燈號變換時到達路口，它必需採信新的燈號顏色。車輛可以在路口等待。給定該城市的地圖，包含：

- 每一道路行駛所需時間(整數)，
- 每一交通號誌燈的兩種燈號號誌時間(整數)，
- 每一交通號誌燈的起始燈色及剩餘即變換燈色的時間(整數)。

給定一起始路口及欲到達之路口，你必須找到一個所需時間為最短的路徑。若有多於一條這種路徑，你只需要輸出其中一條。

假設(ASSUMPTIONS)

- $2 \leqq N \leqq 300$，N 為路口總數，路口編號為整數 1 至 N。
- $1 \leqq M \leqq 14{,}000$，M 為道路總數。
- $1 \leqq l_{ij} \leqq 100$，l_{ij}是從路口i行駛使用連接i與j 的那條道路至路口j所需的時間。
- $1 \leqq t_{ic} \leqq 100$，t_{ic}是路口i的c色燈號號誌時間，c可以是B，代表藍色，或是P，代表紫色。
- $1 \leqq r_{ic} \leqq t_{ic}$，$r_{ic}$是路口i起始燈號色c所剩的號誌時間。

輸入(INPUT)

輸入檔案 **lights.inp** 是一個文字檔。

- 第一行包含兩個數字：起始路口及欲到達路口之編號。
- 第二行包含兩個數字：N, M
- 接下來的N行列出N個路口的資訊。第i+2 行包含第i個路口的資訊：C_i，r_{ic}，t_{iB}，t_{iP}。在此C_i是B或P，代表路口i號誌燈的初始燈色。
- 最後，接下來的M行則包含M條道路之資訊。每一行包含i和j和l_{ij}。i和j為此

道路所連接之兩個路口之編號。

輸出(OUTPUT)

輸出檔 **lights.out** 為文字檔。

如果路徑存在，則：

- 第一行為從起始路口到欲到達路口最短時間路徑所需之時間。
- 第二行列出該最短時間路徑所需經過之各路口。你必須按照行駛路徑上經過路口之順序列出。所以此行第一個整數應該是起始路口的編號，而最後的整數應為欲到達路口的編號。

如果路徑不存在，則：

- 僅輸出一行且僅有一整數 0 在該行。

範例(EXAMPLE)

lights.inp

```
1 4
4 5
B 2 16 99
P 6 32 13
P 2 87 4
P 38 96 49
1 2 4
1 3 40
2 3 75
2 4 76
3 4 77
```

lights.out

```
127
1 2 4
```

交通號誌(TRAFFIC LIGHTS)參考解答
試作者：呂宗弘(IOI'99 銅牌獎)

這一題其實是最短路徑的變形，並不會太難寫，只是在處理時間上何時可以走比較麻煩.。

程式碼:

```
#include <fstream.h>
#include <stdlib.h>

struct PQUEUE                    //最佳優先佇列
{
    unsigned short node ;       //路口
    unsigned long time ;        //抵達此路口的最短時間
    unsigned short back ;       //前一個路口
} ;
```

```
struct EDGE                         //道路
{
        unsigned short n1 , n2 ;//路口一和路口二
        unsigned long time ;        //走過此路所需時間
} huge road [14000] ;

struct NODE                         //路口
{
        unsigned long bt , pt ; //藍燈時間,紫燈時間
        unsigned long init ;        //初始狀態
} node [301] ;

ifstream fin ("lights.inp") ;
ofstream fout ("lights.out") ;
unsigned short N ;
unsigned long M ;

inline unsigned long min (unsigned long a , unsigned long b)
{
        return (a > b ? b : a) ;
}


void output (PQUEUE * queue , short nownode)
{
        unsigned short i ;

        if (queue [nownode].back == queue [nownode].node){
                fout << queue [nownode].node ;
                return ;
        }
        for (i = 0 ; i < nownode ; i ++){
                if (queue [i].node == queue [nownode].back){
                        output (queue , i) ;
                        break ;
                }
        }
        fout << ' ' << queue [nownode].node ;
}

int main (void)
{
        unsigned long li ;
        unsigned short i ;
```

```
unsigned short st , ed ;              //開始路口,終點路口
unsigned long r ;
unsigned long ttime , tn1 , tn2 , tt1 , tt2 ;       //用來存計算時間的變數
char c ;
PQUEUE queue [1000] , temp ;
unsigned long qtime [301] ;
unsigned short front , rear ;

//input
fin >> st >> ed ;
fin >> N >> M ;
for (i = 1 ; i <= N ; i ++){
        qtime [i] = 10000000L ;
        fin >> c >> r >> node [i].bt >> node [i].pt ;
        if (c == 'B')
                node [i].init = node [i].pt + r ;
        else
                node [i].init = r ;
}
for (li = 0 ; li < M ; li ++){
        fin >> road [li].n1 >> road [li].n2 >> road [li].time ;
}

//process
front = 1 ; rear = 0 ;
queue [0].node = st ;
queue [0].time = 0 ;
queue [0].back = st ;
qtime [st] = 0 ;
while (queue [rear].node != ed && front > rear){
        for (li = 0 ; li < M ; li ++){
                if (road [li].n1 == queue [rear].node){
                        ttime = queue [rear].time ;
                        tn1 = road [li].n1 ;
                        tn2 = road [li].n2 ;
                }
                else if (road [li].n2 == queue [rear].node){
                        ttime = queue [rear].time ;
                        tn1 = road [li].n2 ;
                        tn2 = road [li].n1 ;
                }
   else
        continue ;
```

```
tt1 = (ttime + node [tn1].bt + node [tn1].pt - node [tn1].init) % (node [tn1].bt
        + node [tn1].pt) ;
tt2 = (ttime + node [tn2].bt + node [tn2].pt - node [tn2].init) % (node [tn2].bt
        + node [tn2].pt) ;
if (tt1 < node [tn1].bt && tt2 >= node [tn2].bt){
        tt2 -= node [tn2].bt ;
        if (node [tn1].bt - tt1 == node [tn2].pt - tt2){
                if (node [tn1].pt == node [tn2].bt){
                        if (node [tn1].bt == node [tn2].pt)
                                continue ;
                        else
                                ttime += node [tn1].bt - tt1 + node [tn1].pt + min (node
                                        [tn1].bt , node [tn2].pt) ;
                }
                else
                        ttime += node [tn1].bt - tt1 + min (node [tn1].pt , node
                                [tn2].bt) ;
        }
        else
                ttime += min (node [tn1].bt - tt1 , node [tn2].pt - tt2) ;
}
else if (tt1 >= node [tn1].bt && tt2 < node [tn2].bt){
        tt1 -= node [tn1].bt ;
        if (node [tn1].pt - tt1 == node [tn2].bt - tt2){
                if (node [tn1].bt == node [tn2].pt){
                        if (node [tn1].pt == node [tn2].bt)
                                continue ;
                        else
                                ttime += node [tn1].bt - tt1 + node [tn1].bt + min (node
                                        [tn1].pt , node [tn2].bt) ;
                }
                else
                        ttime += node [tn1].bt - tt1 + min (node [tn1].bt , node
                                [tn2].pt) ;
        }
        else
                ttime += min (node [tn1].pt - tt1 , node [tn2].bt - tt2) ;
}
ttime += road [li].time ;
if (ttime >= qtime [tn2])
        continue ;
if (qtime [tn2] == 10000000L){
        qtime [tn2] = ttime ;
        queue [front].node = tn2 ;
        queue [front].time = ttime ;
```

```
                    queue [front].back = tn1 ;
                    i = front - 1;
                    front ++ ;
                    while (i > rear && queue [i + 1].time < queue [i].time){
                        temp = queue [i] ;
                        queue [i] = queue [i + 1] ;
                        queue [i + 1] = temp ;
          i -- ;
                    }
                }
                else {
                    for (i = rear + 1 ; i < front ; i ++){
                        if (queue [i].node == tn2){
                qtime [tn2] = ttime ;
                            queue [i].time = ttime ;
                            queue [i].back = tn1 ;
             i -- ;
                            while (i > rear && queue [i + 1].time < queue [i].time){
                                temp = queue [i] ;
                                queue [i] = queue [i + 1] ;
                                queue [i + 1] = temp ;
                   i -- ;
                            }
                            break ;
                        }
                    }
                }
            }
            rear ++ ;
}

//output
if (queue [rear].node != ed){
        fout << 0 << endl ;
}
else {
        fout << queue [rear].time << endl ;
        output (queue , rear) ;
        fout << endl ;
}

fin.close () ;
fout.close () ;
return 0 ;
```

}

第二天第二題：擺平(FLATTEN)

有一種單人遊戲，玩的是排成一列的 N 堆籌碼，每堆都由零或多個籌碼構成。參考圖 1。各堆以整數 1 到 N 識別。在此遊戲中的每一*步（move）*，你挑選某一堆，例如 *p*，並指定一個數字，例如 *m*。這樣就會由堆 *p* 中分別移動 *m* 個籌碼到相鄰的堆去。圖 2 是走一步的例子。如果 *1 < p < N*，堆 *p* 就有兩個鄰堆：*p-1* 和 *p+1*；如果 *p=1*，就只有一個鄰堆 *2*；如果 *p=N*，就只有一個鄰堆 *N-1*。注意為了能夠走這樣的一步，如果堆 *p* 有兩個鄰堆的話，它必須至少有 *2m* 個籌碼；如果它只有一個鄰堆的話，它必須至少有 *m* 個籌碼。

遊戲的目標是各堆籌碼「擺平」，也就是使各堆的籌碼數都相同，使用的步數越少越好。如果有不止一個解的話，你只需報告其中之一。

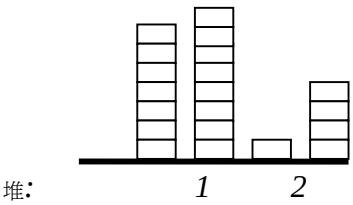


圖 1. 五堆籌碼，分別有 0、7、8、1、4 個籌碼。

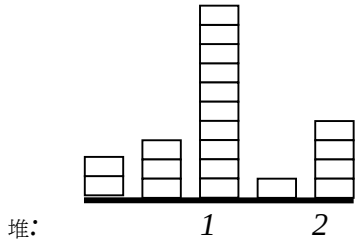


圖 2. 執行一步：p=2, m=2 之後，圖 1 中的各堆籌碼變成這樣。

假設(ASSUMPTIONS)

- 保證不超過 10,000 步就可以擺平各堆籌碼。
- $2 \leqq N \leqq 200$
- $0 \leqq C_i \leqq 2000$，其中C_i是遊戲開始時堆i中的籌碼數，$1 \leqq i \leqq N$

輸入(INPUT)

輸入為一個名為 flat.inp 的文字檔，包含兩行。

- 第 1 行：*N*
- 第 2 行：*N*個整數，其中第*i*個整數是C_i值。

輸出(OUTPUT)

輸出到一個名為 flat.out 的文字檔。

- 第 1 行：步數。（此處稱這個數值為 *M*）
- 接下來的 *M* 行，每行含兩個整數：*p* 和 *m*，代表一步。

輸出檔中各步的順序必須和遊戲的移動順序一致，也就是說，第 1 步必須寫在輸出檔的第 2 行。

範例(EXAMPLE)

flat.inp

```
5
0 7 8 1 4
```

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

擺平(FLATTERN)參考解答
試作者：謝旻錚(IOI'99 銀牌獎)

```
/* Code in C */

#include<stdio.h>
#include<stdlib.h>

struct MOVE{
      int p,m;
};

/* 記錄走法的資料結構 */

int main(void)
{
      int n,i,step,maxi,flag,t;
      long int total,avg,diff[200],opti[200],max;
      struct MOVE move[12000];
      int pile[200];
      FILE *in;
      FILE *out;
      in=fopen("FLAT.INP","r");
      out=fopen("FLAT.OUT","w");
      in=stdin;
      out=stdout;
      fscanf(in,"%d",&n);
      total=0;
      for(i=0;i<n;i++)
      {
            fscanf(in,"%d",&pile[i]);
            total+=pile[i];
      }
/* 讀入資料並記錄總籌碼數 */
      avg=total/n;
/* 先算出每堆籌碼平均數 avg */
      if(n==2)
      {
            if(pile[0]==pile[1])
            {
                  fprintf(out,"0\n");
            }
            else if(pile[0]>pile[1])
            {
                  fprintf(out,"1\n1 %d\n",pile[0]-avg);
            }
            else
            {
                  fprintf(out,"1\n2 %d\n",pile[1]-avg);
            }
            return 0;
      }
/* n=2 時為特例，可以快速找到解。方法如上 */
```

```
    for(i=0;i<n;i++)diff[i]=pile[i]-avg;
/*
算出最初跟目標狀況每堆籌碼的差值 diff[]
令 opti[i]為執行 p=i+1,m=x 所有 x 的總和負值,如執行的過程中有:1 3 跟 1 5 出現,則 opti[0]=-8
此為我的 Greedy method 的對象。
如何算出一組合理的(非正的)opti[]的方法如下面的程式碼
此法除了求出合理的以外，並求出合理的解中總和最大(絕對值和最小)的一組 opti[]
主要的想法是寫成一組聯立方程，可是經由加減消去之後發現有無限多組解。
(想法：因為可以有冗步，如每堆都執行一次 m=1 的步)
可以把 opti[]每個元素視為一個變數。解聯立方程，如下：
-opti[0]+opti[1]=diff[0]
opti[0]-2opti[1]+opti[2]=diff[1]
opti[1]-2opti[2]+opti[3]=diff[2]
.....
opti[n-2]-opti[n-1]=diff[n-1]
共 n 條方程式，求 n 個變數。可是經由加減消去法之後可知道這是一個無線多解的方程組。
但是由矩陣的常識得知這組聯立方程有一個自由度。解開後就得如下結果.....
即若將 opti[0]=v, 則 opti[i]=c[i]+v 其中 c[i]是常數
(由 opti[0]=0 可知 c[0]=0)
*/
    opti[0]=0;opti[1]=diff[0]+opti[0];
    max=(opti[0]>opti[1])?opti[0]:opti[1];
    for(i=2;i<n;i++)
    {
        opti[i]=diff[i-1]+2*opti[i-1]-opti[i-2];
        max=(max>opti[i])?max:opti[i];
    }
/* 執行到這邊 opti[i]就是我之前提出的 c[i](此取 v=0), max 是其中最大的一個。 */
    for(i=0;i<n;i++)
        opti[i]-=max;
/* 再者，因為 c[i]可能大於零，就是不合理的情況，所以減去其中最大的之後
   (就是令 s=-max)就可以得到一個 opti[]是符合我本來所提出的 opti[]
*/

/* 從此處起為 Greedy method */
    for(step=0;;step++)
    {
        flag=0;
        for(i=0;i<n;i++)
            if(pile[i]!=avg)flag=1;
        if(flag==0)break;
/* 若每堆都已經是 avg 個籌碼時結束 */
        max=0;maxi=-1;
        if(opti[0]+pile[0]>0&&pile[0]>0)t=-opti[0];else t=pile[0];
        if(max<t)
        {
            max=t;
            maxi=0;
        }
        for(i=1;i<n-1;i++)
```

```
        {
            if(opti[i]*2+pile[i]>0&&pile[i]>1)t=-opti[i];else t=pile[i]/2;
            if(max<t)
            {
                max=t;maxi=i;
            }
        }
        if(opti[n-1]+pile[n-1]>0&&pile[n-1]>0)t=-opti[n-1];else t=pile[n-1];
        if(max<t)
        {
            max=t;
            maxi=n-1;
        }
        move[step].p=maxi;move[step].m=max;
        opti[maxi]+=max;
        if(maxi==n-1)
        {
            pile[n-1]-=max;pile[n-2]+=max;
        }
        else if(maxi==0)
        {
            pile[0]-=max;pile[1]+=max;
        }
        else
        {
            pile[maxi]-=max+max;pile[maxi-1]+=max;pile[maxi+1]+=max;
        }
/* 找出一個可以走且 m 最大的走。 */
    }
    fprintf(out,"%d\n",step);
    for(i=0;i<step;i++)
        fprintf(out,"%d %d\n",move[i].p+1,move[i].m);
    fclose(in);fclose(out);
    return 0;
}
```

第二天第三題：一片土地(A STRIP OF LAND)

丁格維爾 (Dingilville) 居民試著找一個區域來建機場。你手上有土地的地圖。 地圖是由單位正方形所形成的長方形格子，每個正方形有座標(x, y) ，此處 x 是水平座標，y 是垂直座標。地圖上有顯示每個正方形的高度。

你的任務是找出面積最大的長方形區域，使得:

a) 區域裏最高和最低正方形的高度差小於或等於一個給定的限制 C，而且
b) 該區域的寬度 (即沿著西-東向的長度) 最多 100

如果有不止一個這樣的區域，你只需輸出其中一個。

假設(ASSUMPTIONS)

- $1 \le U \le 700$， $1 \le V \le 700$， 此處 U 和 V 指出地圖的大小。 更仔細地說， U 是西-東向的正方形的個數，而 V 是南-北向的正方形的個數。
- $0 \le C \le 10$
- $-30{,}000 \le H_{xy} \le 30{,}000$，此處整數$H_{xy}$ 是在座標 (x,y) 的正方形的高度，$1 \le x \le U$，$1 \le y \le$

V

- 地圖西南角的座標是 (1, 1) 而且東北角的座標是 (U, V)

輸入(INPUT)

輸入為一個名為 land.inp 的文字檔

- 第一行包含三個整數: U, V, 和 C
- 再下來的 V 行，每一行包含數個整數 H_{xy} ，此處 x = 1, …, U o 再者， H_{xy} 就是在第 (V-y+2) 行輸入的第 x 個數字。

輸出(OUTPUT)

輸出必須是叫做 land.out的文字檔，其中有四個整數指出找到的區域的位置在同一行，即 X_{min} Y_{min} X_{max} Y_{max}，此處 (X_{min} Y_{min}) 是該區域西南角正方形的座標， (X_{max} Y_{max}) 是東北角正方形的座標。

範例(EXAMPLE)

land.inp

```
10 15 4
41 40 41 38 39 39 40 42 40 40
39 40 43 40 36 37 35 39 42 42
44 41 39 40 38 40 41 38 35 37
38 38 33 39 36 37 32 36 38 40
39 40 39 39 39 40 40 41 43 41
39 40 41 38 39 38 39 39 39 42
36 39 39 39 39 40 39 41 40 41
31 37 36 41 41 40 39 41 40 40
40 40 40 42 41 40 39 39 39 39
42 40 44 40 38 40 39 39 37 41
41 41 40 39 39 40 41 40 39 40
47 45 49 43 43 41 41 40 39 42
42 41 41 39 40 39 42 40 42 42
41 44 49 43 46 41 42 41 42 42
45 40 42 42 46 42 44 40 42 41
```

41	40	41	38	39	39	40	42	40	40
39	40	43	40	36	37	35	39	42	42
44	41	39	40	38	40	41	38	35	37
38	38	33	39	36	37	32	36	38	40
39	40	39	39	39	40	40	41	43	41
39	40	41	38	39	38	39	39	39	42
36	39	39	39	39	40	39	41	40	41
31	37	36	41	41	40	39	41	40	40
40	40	40	42	41	40	39	39	39	39
42	40	44	40	38	40	39	39	37	40
41	41	40	39	39	40	41	40	39	42
47	45	49	43	43	41	41	40	39	42
42	41	41	39	40	39	42	40	42	42
41	44	49	43	46	41	42	41	42	42
45	40	42	42	46	42	44	40	42	41

(北) y

x (東)

land.out

4 5 8 11

一片土地(A STRIP OF LAND)參考解答
試作者：林語君(SEARCC'99 第四名)

```
Program Land;
{$I-}

{
Files       Land.in          Inputfile
              Land.out         Outputfile

Time Complexity O(n^3)
```

nearly, but sure exceed.

Algorithm

先解決一維的題目，盡量使複雜度接近 O(n)
接下來解決二維的時候，用兩個變數 i, j 選擇從第 i 列到第 j 列的"區域"
把這個區域壓縮成一維處理(壓縮是指只紀錄最大最小即可)

兩個變量 i, j 再加上一維處理時間，時間複雜度約爲 O(n^3)

最大的問題是資料過於龐大 490K 遠大於 64K 空間
只好不儲存重複讀取並盡量減少讀取次數
重複讀取是執行時耗時最多的部分
}

```
Uses Crt, Dos;

Const InputFile    = 'Land.in';
      OutputFile = 'Land.out';

      Max = 700;
      Maxnum = 30001;

      Debug = false;
      {For clearly seen the running speed, check Debug TRUE}

Type Range = record lo, hi, nlo, nhi : integer; end;

Var f, fo : text;

    c, count, ec, less, x, y, i, j, k, t : integer;

    s : Array[1..Max] of Range;
    {s : 1D problem array}
    done : Array[1..Max] of boolean;
    {done[i] check for the finish of sections which start from row i}
    e : Array[1..Max+1] of integer;
    {for speed improvement}

    al, ar, ax1, ay1, ax2, ay2 : longint;
    {best answer : al-ar for 1D, (ax1,ay1)-(ax2,ay2) for 2D}

    ho, mi1, mi2, se1, se2, ss1, ss2 : Word;
    {variables for timer}

{Check for file existance}
Procedure CallEnd(code : integer);
    Begin
        if code = 1 then begin
            Writeln(InputFile, ' not found.');
```

```
                Halt(1);
            end;
        End;

{Caculate runtime}
Function PrintTime(m1, s1, ss1, m2, s2, ss2 : word) : integer;
        Var tm, ts, tss : word;
        Begin
            if s2 < s1 then begin
                ts := s1 - s2;
                tm := m2 - m1 - 1;
            end;
            if s1 <= s2 then begin
                ts := s2 - s1;
                tm := m2 - m1;
            end;

            if ss2 > ss1 then tss := ss2 - ss1
                         else tss := ss2;

            PrintTime := tm*60 + ts;
        End;

{1D problem function}
Procedure Run;
        Var i, l, r, p, q : integer;
        Begin
            ar := 0; al := 0;
            l := 0; r := 0;
            p := 0; q := 0;

            While l + (ar-al) + 1 <= x do begin
                if p < q then l := p + 1 else l := q + 1;

                {for speed improvement}
                for l:=l to x-(ar-al) do begin
                    if l = x-(ar-al) then break;

                    if s[l].hi - s[l].lo > c then continue;
                    if (ar=0) and (al=0) then break;
                    if (abs(s[l].hi - s[l+(ar-al)+1].lo) > c) or
                        (abs(s[l].lo - s[l+(ar-al)+1].hi) > c) then continue;
                    if (abs(s[l].hi - s[l+(ar-al)].lo) > c) or
                        (abs(s[l].lo - s[l+(ar-al)].hi) > c) then continue;
                    if l < r+1 then
                        if (abs(s[l].hi - s[r+1].lo) > c) or
                            (abs(s[l].lo - s[r+1].hi) > c) then continue;
                    break;
                end;
                if l = x-(ar-al) then break;
                {for speed improvement}
```

```
                if p < l then p := l;
                if q < l then q := l;
                for i:=l to x+1 do begin
                    if i > x then break;
                    if s[i].hi - s[i].lo > c then break;
                    if (s[p].hi - s[i].lo > c) or (s[i].hi - s[q].lo > c) then break;
                    if s[i].hi >= s[p].hi then p := i;
                    if s[i].lo <= s[q].lo then q := i;
                end;

                if (ar=0) or (al=0) or (i-1 > l + (ar-al)) then begin
                    r := i-1;
                    al := l; ar := r;
                end;
            end;
        End;

Begin
    if Debug then begin
        GetTime(ho, mi1, se1, ss1);
        Clrscr;
    end;

    Assign(fo, OutputFile); Rewrite(fo);
    Assign(f, InputFile);

    ax1 := 1; ay1 := 1;
    ax2 := 1; ay2 := 1;

    {for speed improvement}
    Reset(f); Readln(f, x, y, c); Close(f);
    FillChar(done, sizeof(done), 0);
    for i:=1 to Max do e[i] := y;
    less := y;
    {for speed improvement}

    {Main loop}
    count := 0;
    Repeat
        Reset(f);
        if IOResult <> 0 then CallEnd(1);
        Readln(f, x, y, c);

        {Section start from row i}
        for i:=1 to y do if not done[i] then begin
            done[i] := true; inc(count);
            While done[less] do dec(less);
            if Debug then Write(count:4, ':', i:-4);
            for j:=1 to x do begin s[j].lo := Maxnum; s[j].hi := -Maxnum; end;
            ec := x * 2;

            {Running 1D function to find the answer to Section row i-j}
```

```
                for j:=i to e[i] do begin
                    for k:=1 to x do begin
                        Read(f, t);
                        if t <= s[k].lo then begin
                            s[k].lo := t; s[k].nlo := j;
                            if (j<>i) and (s[k].nlo = i) then dec(ec);
                        end;
                        if t >= s[k].hi then begin
                            s[k].hi := t; s[k].nhi := j;
                            if (j<>i) and (s[k].nhi = i) then dec(ec);
                        end;
                    end;
                    if ec <= 0 then e[i+1] := j-1;
                    Readln(f);

                    Run;
                    if (al=0) or (ar=0) then break;
                    if longint(j-i+1)*(ar-al+1) > (ax2-ax1+1)*(ay2-ay1+1) then begin
                        ax1 := al; ay1 := i;
                        ax2 := ar; ay2 := j;
                    end;
                    if longint(y-i)*(ar-al+1) < (ax2-ax1+1)*(ay2-ay1+1) then break;
                end;
                if j > less then break;
                i := j;
            end else begin
                if i > less then break;
                Readln(f);
            end;

            Close(f);
        Until count = y;
        {Main loop}

        Writeln(fo, ax1, ' ', y-ay2+1, ' ', ax2, ' ', y-ay1+1);
        Writeln(fo, (ax2-ax1+1)*(ay2-ay1+1));
        Close(fo);

        if Debug then begin
            Writeln;
            Writeln(ax1, ' ', y-ay2+1, ' ', ax2, ' ', y-ay1+1);
            Writeln((ax2-ax1+1)*(ay2-ay1+1));
            GetTime(ho, mi2, se2, ss2);
            Writeln;
            Writeln(PrintTime(mi1, se1, ss1, mi2, se2, ss2), ' seconds used.');
        end;
End.
```