1998 年 IOI 試題與參考解答

一、第一天第一題：接觸

Astro Insky 博士在某無線遙測天文台工作，最近她注意到一種由銀河中心發射出來的微波脈衝，十分令人好奇。這脈衝可能是由外太空智慧生物所發出的？或只是星球的尋常脈動而已？

任務(Task)

你需要提供一個工具去分析她檔案中的位元字串，以幫助 Insky 博士發現真相。Insky 博士希望能找到長度介於（包含）A 到 B 間的字串，這些字串較常出現於每日的檔案之中。對每一個檔案而言，她希望找到較大的 N 種不同頻率（即出現的次數）。不同字串可以互相重疊，且只考慮至少出現過一次的字串。

輸入資料(Input Data)

檔案 CONTACT.IN 存放資料序列，格式如下：

第 1 橫列：整數 A 表示最短字串長度。

第 2 橫列：整數 B 表示最長字串長度。

第 3 橫列：整數 N 表示有幾種不同頻率。

第 4 橫列：一串 0 與 1 字元，以字元 2 結束。

輸入範例(Sample Input)：

```
2
4
10
0101001001000100011110110000102
1001100111100001001001111001002
000002
```

在此例中，需要找出下列字元序列中最常出現的 10 種字串頻率，其長度介於 2 到 4 之間。

```
0101001001000100011110110000101
0011001111000010010011111001000 0
000
```

注意此處輸入檔案的第四橫列在印出時是因排版空間不夠而斷開。在此例中，字串 100 出現了 12 次，字串 1000 出現了 5 次，而 00 是最常出現的字串。

輸出資料(Output Data)

輸出檔 CONTACT.OUT 中最多可有 N 個橫列，分別列出 N 個最常出現的頻率以及相對應的各個字串。列印時應以頻率的遞減順序印出，格式如下：

frequency pattern pattern … pattern

此處frequency為其後各字串出現的頻率。各橫列中的不同字串應以其長度的遞減順序排列。同長度的各字串再依遞減的數值順序排列。如果出現的頻率數小於N種，則輸出橫列數小於N。

輸出範例(Sample Output):

對應輸入範例，輸出應如下：

```
23 00
15 10 01
12 100
11 001 000 11
10 010
8 0100
7 1001 0010
```

6 0000 111
5 1000 110 011
4 1100 0011 0001

條件限制(Constraints)

輸入檔案中最多可含 2 Megabytes。

參數 A、B 及 N 限制如下：

$0 < N \le 20$

$0 < A \le B \le 12$

二、參考解答

試作者：鐘楷閔(IOI'98 銀牌獎)

/*這題一看到題目的要求：輸入檔案最大達 2Megabytes，就知道必須找出一個時間複雜度低於或等於 O(n)的演算法，而且係數還不能太大。不過由於要求的是統計的工作，所以 O(n)是跑不掉的，因此要想辦法降低其係數。因為資料的內容為 1 或 0，故可以下列方法達成：

（1）將一字串視為二進位數字，加上長度資訊便可表示這字串，例如：01000 可視為 8，長度 5，再開一二維陣列存每一種字串的頻率，例如：01000 的頻率存在 times[5][8]中。

（2）在讀資料時，採每次讀一字元的方式，並以 table[n]表示以目前讀到的字元為結尾且長度為 n 的字串，每讀入一字元時做 table[n] =(table[n]%(2^n))*2+k 運算即可轉為下一筆長度為 n 的字串（table[n]為目前長度 n 的字串，2^n 指 2 的 n 次方，k 是下一字元的值。）例如：在資料 0001011001000…中讀到第六字元則 table[4]應等於 5，即字串 0101 讀下一字元時只須做(4%8)*2+1 得 tabl[4]=11，即第四筆長度為四的字串為 1011，接著 times[4][11]++累加頻率。

以此 DP 的方式減少係數後可在約 17 秒左右解出 2Megabytes 大小的測試值，程式碼如下（註：在 TC 下若全域變數使用超過 64K 的記憶體，會導致變數存取速度減慢，而不能在 20 秒內解出。）:*/

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

FILE *fp, *fi;
int a, b, n;
int table[22];                                  //記錄目前各種長度的字串內容
long *times[13];                                //儲存各種字串的頻率
long radix[22];
long map[22];                                   //記錄前 n 大的頻率值

void GetData( void )                            //開檔及讀 N,A,B
{
    fp = fopen( "contact.in", "r+t");
    fi = fopen( "contact.out", "w+t");
    fscanf( fp, "%d%d%d\n", &a, &b, &n);
    radix[0] = 1;
    for( int i = 1; i < 20; i++)
        radix[i] = radix[i-1] * 2;
    for( i = 0; i <= 12; i++)
    {
        times[i] = new long[radix[i]];
```

```
            for( int j = 0; j < radix[i]; j++)
                times[i][j] = 0;
        }
}

void Insert( long x)
{
        for( int i = 0; i < n; i++)
            if( map[i] < x)
                map[i] ^= x ^= map[i] ^= x;
            else if( map[i] == x)
                return;
}

void Count( void )                                          //計算每種字串出現頻率
{
        int k;
        long t = 0;
        char ch;
        while( 1 )
        {
            fscanf( fp, "%c", &ch);
            if( ch == '2')
                break;
            if( ch != '0' && ch != '1')
                continue;
            k = ch - '0';
            for( int i = a; i <= b; i++)
            {
                table[i] = table[i] * 2 + k;
                if( t >= i)
                    table[i] %= radix[i];
                if( t >= i - 1)
                    times[i][table[i]]++;
            }
            t++;
        }
        for( int i = a; i <= b; i++)
            for( int j = 0; j < radix[i]; j++)
                Insert( times[i][j]);
}

void Print( int len, int x)
{
        fprintf( fi, " ");
        for( int i = len - 1; i >= 0; i--)
        {
            fprintf( fi, "%d", x / radix[i]);
            x %= radix[i];
        }
}
```

```
void Show( void )
{
	int i = 0, j ,k;
	fprintf( fi, "%ld", map[i]);
	for( j = b; j >= a; j--)
		for( k = radix[j] - 1; k >= 0; k--)
			if( times[j][k] == map[i])
				Print( j, k);
	for( i = 1; i < n; i++)
	{
		if( map[i] == 0)
			break;
		fprintf( fi, "\n%ld", map[i]);
		for( j = b; j >= a; j--)
			for( k = radix[j] - 1; k >= 0; k--)
				if( times[j][k] == map[i])
					Print( j, k);
	}
	for( i = 0; i < 12; i++)
		delete times[i];
}
int main( void )
{
	GetData();
	Count();
	Show();
	fclose( fp );
	fclose( fi );
	return 0;
}
```

三、第一天第二題：星光滿夜

在夜晚的高空上，充滿了閃亮的星群(cluster)。所謂星群是指一群在上、下、左、右或斜對角方向相互連接的星星。任一星群不得為其他更大星群的子星群。

星群可以是相似的。兩個星群如果有相同的形狀，縱使方向不同，即稱之為相似的(similar)星群。大致來說，每一個星群有 8 個可能的轉變方向，如圖 1：

圖1. 8個相似的星群

此夜晚的天空圖(sky map)是以一個 2 維陣列表示。陣列中每一個格子(cell)代表一個星星。(以 1 代表星星，0 代表空白。)

Task（任務）

給一個天空圖，將所有星群用小寫字母標示出來。請將相似的星群用同一個小寫字母表示，不相似的星群則以不同的字母表示。請將天空圖中所有的星星(1)改成它的星群所對應的小寫字母。

Input Data（輸入資料）

輸入檔 STARRY.IN 的前二橫列為 W（天空圖的寬度）及 H（天空圖的長度）。天空圖自第 3 橫列開始，共有 H 橫列，每列 W 個字元（0 或 1）。

輸入範例: (Sample Input)

```
23
15
10001000000000010000000
01111100011111000101101
01000000010001000111111
00000000010101000101111
00000111010001000000000
00001001011111000000000
10000001000000000000000
00101000000111110010000
00001000000100010011111
00000001110101010100010
00000100110100010000000
00010001110111110000000
00100001110000000100000
00001000100001000100101
00000001110001000111000
```

在此範例中，此天空圖寬度為 23 且長度為 15。為使此圖更容易了解，圖 2 只顯示星星的位置。

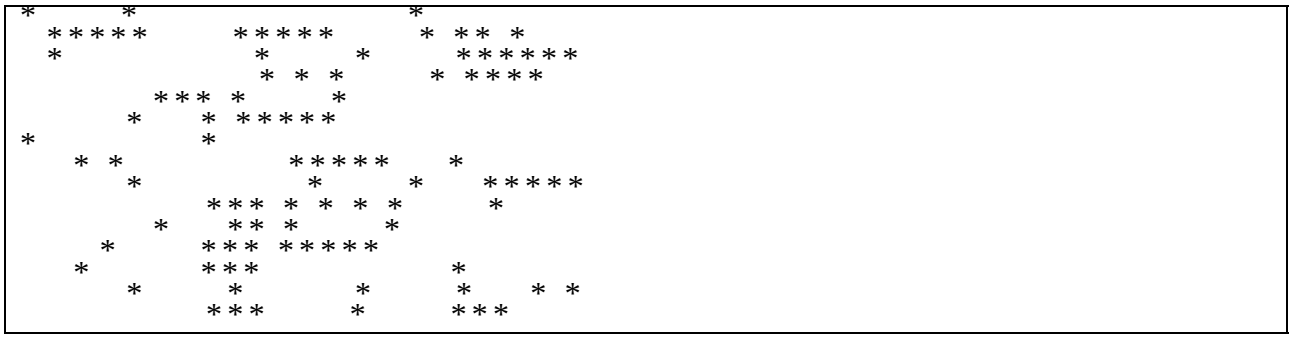


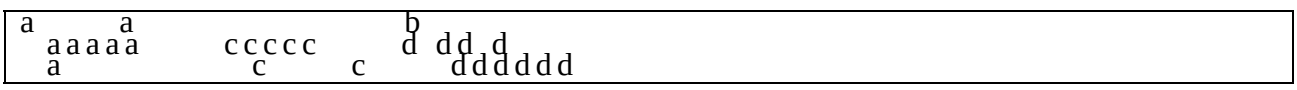
圖 2. 天空圖範例

Output Data（輸出資料）

輸出檔為STARRY.OUT，應含有與輸入檔STARRY.IN相同的天空圖，但每個星群內的星星已按照任務說明的方式改為小寫字母。底下為一組解答，而圖3是將0去掉後的示意圖。

輸出範例: (Sample Output)

```
a000a0000000000b0000000
0aaaaa000ccccc000d0dd0d
0a0000000c000c000dddddd
000000000c0b0c000d0dddd
00000eee0c000c000000000
0000e00e0ccccc000000000
b000000e000000000000000
00b0f000000ccccc00a0000
0000f000000c000c00aaaaa
0000000ddd0c0b0c0a000a0
00000b00dd0c000c0000000
000g000ddd0ccccc0000000
00g0000ddd0000000e00000
0000b000d0000f000e00e0b
0000000ddd000f000eee000
```



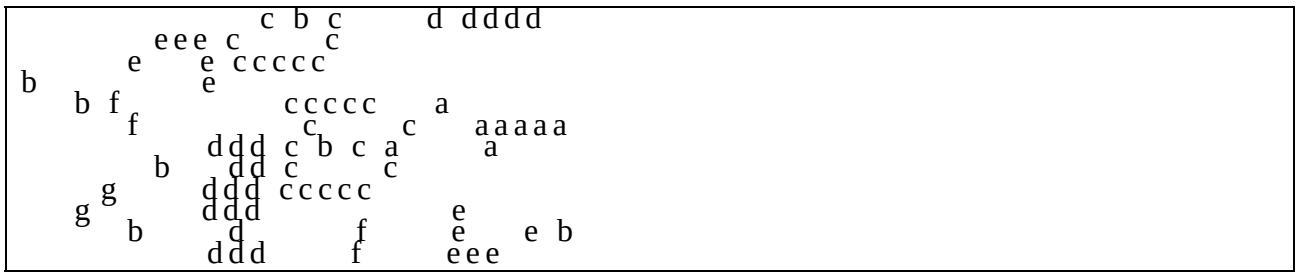


圖 3. 星群已標示完成的天空圖

Constraints（條件限制）

0 ≤ W (天空圖的寬度) ≤ 100

0 ≤ H (天空圖的長度) ≤ 100

0 ≤ 星群數≤ 500

0 ≤ 不相似的星群數 ≤ 26 (a..z)

1 ≤ 每一星群的星星數 ≤ 160

四、參考解答

試作者：江盈宏(IOI'98 銅牌獎)

```
#include<stdio.h>
/*
整體變數
map 代表讀入的資料(0 或 1),2 代表已經處理過
result 為處理完的結果
compare 是比對時使用的二維陣列
w 為寬,h 為高
*/
char map[100][100];
char result[100][100];
char compare[100][100];
int w,h;

/*Cluster 結構與陣列*/
struct Clu {
   int stars;              /*星數*/
   int w,h,l,u;            /*此星群之:w(寬),h(高),l(最左),u(最上)*/
   int pos[160][2];        /*星星之座標*/
};
struct Clu clu[26],now;
int clus;

/*由檔案讀入資料*/
void getdata()
{
   int i,j;
   FILE *inf;
   inf=fopen("starry.in","r");

   fscanf(inf,"%d",&w);
   fscanf(inf,"%d",&h);
   for(i=0;i<h;i++)
```

```
        fscanf(inf,"%s",map[i]);
    for(i=0;i<h;i++)
        for(j=0;j<w;j++)
            result[i][j]='0';

    fclose(inf);
}

/*判斷是否為合法的位置*/
int Position(int x,int y)
{
    return (x>=0 && x<h && y>=0 && y<w);
}

/*逐一找出 connected component 之函式,使用 BFS*/
void FindComponent(int x,int y)
{
    int queue[160][2],front=0,rear=0;
    int i;
    int nx,ny,zx,zy;
    int l,r,u,d;
    int move[8][2]={{-1,-1},{-1,0},{-1,1},{0,-1},{0,1},{1,-1},{1,0},{1,1}};

    queue[front][0]=x,queue[front][1]=y,front++;
    map[x][y]='2';
    now.stars=0;
    while(front!=rear) {
        nx=queue[rear][0];
        ny=queue[rear][1];
        now.pos[now.stars][0]=nx,now.pos[now.stars][1]=ny;
        now.stars++;
        for(i=0;i<8;i++) {
            zx=nx+move[i][0],zy=ny+move[i][1];
            if(Position(zx,zy))
                if(map[zx][zy]=='1')
                    queue[front][0]=zx,queue[front][1]=zy,map[zx][zy]='2',front++;
        }
        rear++;
    }

    /*找出星群之最左及最上,並將座標減去最左及最上,成相對位置*/
    u=d=now.pos[0][0];
    l=r=now.pos[0][1];
    for(i=1;i<now.stars;i++) {
        if(now.pos[i][0]<u) u=now.pos[i][0];
        if(now.pos[i][0]>d) d=now.pos[i][0];
        if(now.pos[i][1]<l) l=now.pos[i][1];
        if(now.pos[i][1]>r) r=now.pos[i][1];
    }
    now.h=d-u;
    now.w=r-l;
    now.l=l,now.u=u;
```

```
   for(i=0;i<now.stars;i++) {
      now.pos[i][0]-=now.u;
      now.pos[i][1]-=now.l;
   }
}

/*判斷星群 n 與星群 now 之異同*/
int Check(int n,int usen,int con)
{
   int i,j;
   int use[2][2]={ {0,1},{1,0} };
   int co[4][4]={ {0,-1,0,-1},{0,-1,1,1},{1,1,0,-1},{1,1,1,1} };
   for(i=0;i<clu[n].stars;i++)
      if(compare[co[con][0]*clu[n].h-co[con][1]*now.pos[i][use[usen][0]]]
                  [co[con][2]*clu[n].w-co[con][3]*now.pos[i][use[usen][1] ]]==1)
         compare[co[con][0]*clu[n].h-co[con][1]*now.pos[i][use[usen][0]]]
                  [co[con][2]*clu[n].w-co[con][3]*now.pos[i][use[usen][1] ]]=0;
      else
         return 0;
   return 1;
}

/*將已有的星群逐一比對,並做 8 次旋轉*/
int TheSame(int n)
{
   int i,j,k;
   for(i=0;i<2;i++)
      for(j=0;j<4;j++) {
         for(k=0;k<clu[n].stars;k++)
            compare[clu[n].pos[k][0]][clu[n].pos[k][1]]=1;
         if(Check(n,i,j)) return 1;
      }

   for(i=0;i<clu[n].stars;i++)
      compare[clu[n].pos[i][0]][clu[n].pos[i][1]]=0;

   return 0;
}

/*傳回 now 所代表的星群,若傳回 clus 代表為新增之星群*/
int Compare()
{
   int i;
   for(i=0;i<clus;i++)
      if(now.stars==clu[i].stars)
         if(now.w==clu[i].w && now.h==clu[i].h || now.w==clu[i].h &&
now.h==clu[i].w)
            if(TheSame(i)) return i;
   return clus;
}
```

```
/*主要解題函式,對每一個座標點尋找未發現過的星群並輸出至 result*/
void solve()
{
   int i,j,k;
   int n;
   for(i=0;i<h;i++)
      for(j=0;j<w;j++)
         if(map[i][j]=='1') {
            FindComponent(i,j);
            n=Compare();
            if(n==clus) clu[clus++]=now;
            for(k=0;k<now.stars;k++)
               result[now.pos[k][0]+now.u][now.pos[k][1]+now.l]='a'+n;
         }
}

/*輸出至檔案*/
void output()
{
   int i,j;
   FILE *outf;
   outf=fopen("starry.out","w");
   for(i=0;i<h;i++) {
      for(j=0;j<w;j++)
         fprintf(outf,"%c",result[i][j]);
      fprintf(outf,"\n");
   }
   fclose(outf);
}

/*主函式*/
int main()
{
   getdata();
   solve();
   output();
   return 0;
}
```

五、第一天第三題：宴會燈

為了使 IOI'98 盛大晚宴增添光彩，我們備有 N 盞燈泡，編號從 1 到 N。這些燈泡接到四個控制按鈕：

按鈕 1—當這個按鈕按下時，所有的燈泡改變它的狀態；也就是說，原來若是亮的就變成暗的；原來若是暗的就變成亮的。

按鈕 2—改變所有編號為奇數的燈泡的狀態。

按鈕 3—改變所有編號為偶數的燈泡的狀態。

按鈕 4—改變所有編號為 3K+1 的燈泡的狀態($K \geq 0$)，亦即編號為 1、4、7、…的燈泡。

有一個計數變數 C，用來記錄按鈕被按下的總次數。

當宴會開始時，所有燈泡均為亮的。而計數變數 C 則設定為 0。

任務(Task)

給定計數變數 C 之值，以及一部份燈泡最後的狀態資訊，請撰寫一程式來計算合乎上述條件之不同燈泡組態數。

輸入資料(Input Data)

檔名為 PARTY.IN 的檔案含有四橫列，用以描述燈泡的個數 N、按鈕按下的次數 C、及部份燈泡最後的狀態。

第一橫列含有 N 之值。

第二橫列則為計數變數 C 最後之值。

第三橫列一一列出最後狀態必須為亮著的燈泡編號，之間用一空白符號隔開，末尾用一整數-1 標示。

第四橫列一一列出最後狀態必須為暗著的燈泡編號，之間用一空白符號隔開，末尾用一整數-1 標示。

輸入範例(Sample Input)：

```
10
1
-1
7   -1
```

此範例中有 10 個燈泡，且只按下按鈕一次。編號為 7 的燈泡其最後狀態為暗的。

輸出資料(Output Data)

檔案 PARTY.OUT 必須含有所有燈泡之所有可能的最後的狀態(不允許重覆)。每一種可能狀態必須各自寫在不同的橫列上。這些狀態可用任意次序列出。

每一橫列含有 N 個字母，其中第 1 個字母表示編號為 1 的燈泡狀態，第 N 個字母表示編號為 N 的燈泡狀態。而字母為 0 (zero) 表示那盞燈是暗的；字母為 1 (one) 表示那盞燈是亮的。

輸出範例(Sample Output) :

```
0000000000
0110110110
0101010101
```

在此範例中，有三種可能的最後狀態：

1.所有的燈泡為暗的；

2.編號為 1、4、7、10 的燈泡為暗的，且編號為 2、3、5、6、8、9 的燈泡為亮著；

3.編號為 1、3、5、7、9 的燈泡為暗的，且編號為 2、4、6、8、10 的燈泡為亮著。

條件限制(Constraints)

參數 N 及 C 有下列限制：

$10 \le N \le 100$

$1 \le C \le 10000$

最後狀態被限定為暗的燈泡的數量將小於或等於 2。

最後狀態被限定為亮的燈泡的數量將小於或等於 2。

每一個輸入測試檔都至少會有一組最後狀態的解。

六、參考解答

試作者：鐘楷閔(IOI'98 銀牌獎)

/*因為每個按鈕所產生的影響為獨立的，且兩次為一循環，我們可以知道其實這四個按鈕所能產生的樣本是不多的，只有 2^4 種。另外可發現這 16 種樣本的燈泡明暗狀態皆是六個為一循環。因此甚至將這些樣本加以記錄、建表處理就可以解決這一題 。但我當時是用基本的 search 演算法加上重複盤面不往下展的 cut 暴力的解題，以下是當時的程式碼：*/

```
#include<stdio.h>
#include<conio.h>

FILE *fp, *fi;
int used[5];
int map[101], n, c, on[2] = { -1, -1}, off[2] = { -1, -1};

void GetData( void )
{
	int k;
	fp = fopen( "party.in", "r+t");
	fi = fopen( "party.out", "w+t");
	fscanf( fp, "%d%d", &n, &c);
	c %= 4;
	if( c == 0) c = 4;
	k = 0;
	while( 1 )
	{
		fscanf( fp, "%d", &on[k]);
		if( on[k] == -1)
			break;
		k++;
	}
	k = 0;
	while( 1 )
	{
		fscanf( fp, "%d", &off[k]);
		if( off[k] == -1)
			break;
		k++;
	}
	for( int i = 1; i <= n; i++)
		map[i] = 1;
}

void Do( int kind )
{
	int i;
	switch( kind)
	{
		case 1:
			for( i = 1; i <= n; i++)
				map[i] = !map[i];
			break;
		case 2:
			for( i = 1; i <= n; i += 2)
				map[i] = !map[i];
			break;
		case 3:
			for( i = 2; i <= n; i += 2)
				map[i] = !map[i];
```

```
                break;
            case 4:
                for( i = 1; i <= n; i += 3)
                    map[i] = !map[i];
                break;
        }
}

void Show( void )
{
    for( int i = 0; i < 2; i++)
        if( on[i] == -1) break;
        else if( map[on[i]] != 1) return;
    for( i = 0; i < 2; i++)
        if( off[i] == -1) break;
        else if( map[off[i]] != 0) return;
    for( i = 1; i <= n; i++)
        fprintf( fi, "%d", map[i]);
    fprintf( fi, "\n");
}

void Count( int index, int min)
{
    if( index == c || ( index < c && index % 2 == c % 2))
        Show();
    for( int i = min; i <= 4; i++)
        if( used[i] == 0)
        {
            Do( i );
            used[i] = 1;
            Count( index + 1, i + 1);
            Do( i );
            used[i] = 0;
        }
}

int main( void )
{
    GetData();
    Count(0, 1);
    fclose( fp );
    fclose( fi );
    return 0;
}
```

七、第二天第一題：圖片問題

牆壁上貼有許多長方形的海報、相片等長方形的圖片。圖片各邊均為垂直或水平。圖片的部份或全部可能被其他圖片所遮蓋。所有長方形圖片聯集體的內外框長度和，稱為總周長(perimeter)。

Task（任務）

寫一程式計算總周長。

圖 1 顯示的例子中有 7 個長方形圖片。

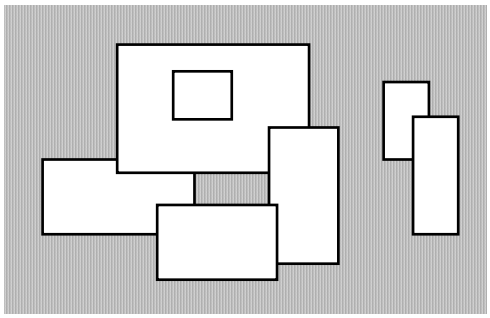
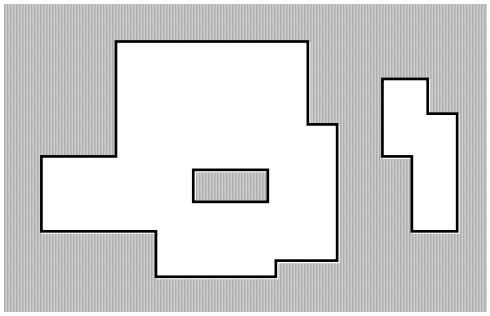


圖 1. 七個長方形圖片



其相對應的內外框即為圖 2 中所有線段的集合。

圖 2. 長方形圖片聯集體的內外框

所有長方形的端點座標均為整數。

Input Data（輸入資料）

檔案 PICTURE.IN 的第一橫列是牆上所貼的長方形圖片總數。之後每一橫列是一個長方形的左下角與右上角的整數座標。各座標的 x 值在前，y 值在後。

Sample Input（範例輸入）

```
7
-15 0 5 10
-5 8 20 25
15 -4 24 14
0 -6 16 4
2 15 10 22
30 10 36 20
34 0 40 16
```

本範例相對應圖 1 所示的各長方形圖片。

Output Data（輸出資料）

檔案 PICTURE.OUT 中應有一非負整數，即為長方形圖片聯集體的總周長。

Sample Output（範例輸出）

```
228
```

此值為上述範例的總周長。

Constraints（限制）

0 ≤ 長方形圖片數 < 5000

各座標值均在[-10000,10000]範圍內，且任一長方形圖片之面積均為正值。

輸出數值有可能需要用到 32 位元數值表示法(32-bit signed representation)。

八、參考解答

試作者：吳光哲(IOI'98 銀牌獎)

/*這題算是今年 IOI 六題中最難的一題, 我及另外三名隊友都只答對前兩筆 test data。這裡所引用的演算法與程式都是大會提供的參考解答, 相關資料請見:

http://olympiads.win.tue.nl/ioi/ioi98/contest/picture

這題是屬於幾何演算法的題目，雖然本題對於每一組測試資料都只有唯一解,但解法相當多種,大家有機會可以試著寫寫看。解法很多，在這邊我提出我覺得以前比較少見的演算法:在這個演算法中，不只是輸入的矩形要被討論，兩個矩形重疊的交集，也要視為一個新的矩形。這麼做雖然有可能會產生多達 2^n-1 個矩形，但一般來說，這種情形並不會太嚴重。
根據排容原理:

```
  | S1 ∪ S2 ∪ S3 ∪ ... ∪ Sn |
=    Si -    |Si∩Sj| +    |Si∩Sj∩Sk| -    |Si∩Sj∩Sk∩Sl|+.....
   + (-1)^(m+1)   |S1∩S2∩....∩Sm|
   全部矩形聯集的總周長
= 每個矩形的周長 - 任兩矩形交集的周長 + 任三矩形交集的周長 - .....
   + (-1)^(m+1)*全部矩形交集圖形的周長
```

又無論多少矩形的交集，都將還是矩形，因此可以利用類似動態規畫的手法，
利用 k-1 個矩形的交集得到 k 個矩形的交集。 */

```
/* Author: Michel Wermelinger (mw@di.fct.unl.pt) */
#include <stdio.h>
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) > (b) ? (b) : (a))

typedef struct {
   char sign;
   short int x, y, X, Y;
   /* sign 1,-1 表示奇或偶數個矩形的交集,
        x,y,X,Y 表示矩形左下與右上角的座標 */
} pict;

#define MAX_PICT 7000
/* 最多幾塊矩形(含交集的), 須 9*MAX_PICT bytes */
pict p[MAX_PICT];

void intersect (pict p1, pict p2, pict *p3) /* 將 p1,p2 兩矩形的交集存於 p3 */
{
   p3->x = max(p1.x, p2.x); p3->y = max(p1.y, p2.y);
   p3->X = min(p1.X, p2.X); p3->Y = min(p1.Y, p2.Y);
   p3->sign = -p1.sign * p2.sign;
}

int main (void)
{
   long perim = 0;
   int npf, npt = 0; /* 分別表示輸入的矩形數與計算時實際的矩形數量 */
   FILE *f;
   int i, j, k;

   if ((f = fopen("PICTURE.IN", "r")) == NULL) {
      printf("Empty file!\n");
      return 0;
   }

   fscanf (f, "%d\n", &npf);
   for (i = 0; i < npf ; i++) {
```

```
        k = npt++;
        fscanf(f, "%d %d %d %d\n", &p[k].x, &p[k].y, &p[k].X, &p[k].Y);
        p[k].sign = 1;
        for (j = 0;   j < k; j++)   {
           intersect   (p[j] , p[k] , &p[npt]);
           if ((p[npt].X > p[npt].x && p[npt].Y >= p[npt].y) ||
                 (p[npt].X == p[npt].x && p[npt].Y > p[npt].y)) /*  當  p[npt]  存在  */
              if (p[npt].x == p[k].x && p[npt].X == p[k].X &&
                    p[npt].y == p[k].y && p[npt].Y == p[k].Y)   {
                 npt = k;                              /* p[k]  完全被  p[j]  覆蓋則捨棄  p[k] */
                 break;
              } else
                 npt++;
        }
     }
     fclose(f);

     for (i = 0; i < npt; i++)
        perim += (long)p[i].sign * 2 * (p[i].X - p[i].x + p[i].Y - p[i].y);
     printf ("Perimeter=%ld\n", perim);
     f = fopen("PICTURE.OUT", "w");
     fprintf (f, "%ld\n", perim);
     fclose(f);
     return 0;
}
```

九、第二天第二題：聚會點

數世紀之前，亞瑟王和圓桌武士們通常在每年的元旦聚會，以重溫他們的友誼。仿效他們的活動，我們設計了一種單人玩的棋盤遊戲，在棋盤上有一個國王和數個武士，隨機置放在不同的方格上。

棋盤是一個 8x8 的方格陣列。

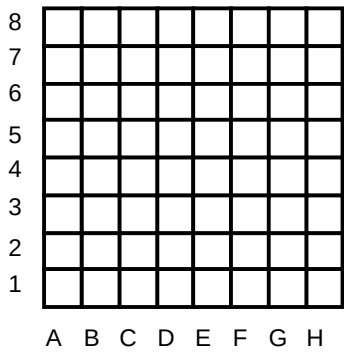


圖 1. 棋盤

國王可移動至八個鄰近方格中的任何一個，如圖 2 所示，只要不會掉到棋盤外面即可。

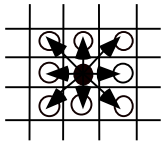


圖 2. 國王之可能棋步

武士可跳到如圖 3 所示的八個方格中的任何一個，只要不會掉到棋盤外面即可。

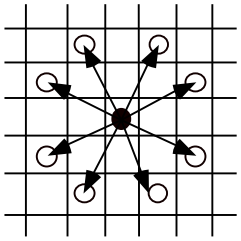


圖 3. 武士之可能棋步

在棋局進行之中，一個方格可以同時放多個棋子。因此任何棋子都不會造成其他棋子行動的阻礙。

玩家的目標是移動各棋子，在最少的步數內，讓他們聚會於同一個方格。要達成此目標，玩家必須遵照上面所說的規則移動棋子。此外，當國王和一個或多個武士處於同一方格內的時候，接下來玩家可以選擇將國王和該方格內的某一個武士一併移動，直到抵達最後的聚會點為止，就好像單個武士的走法一樣。而每次武士與國王一併移動只算一步。

Task（任務）

寫一程式，計算將所有棋子聚會於同一方格內所需之最少步數。

Input Data（輸入資料）

檔案 CAMELOT.IN 存放棋盤的初始狀態，以一個字元串列表示之。該字串最多包含 64 個不同的棋盤方格座標，第一個是國王的位置，其他則為武士的位置。每一位置表示成一個「字母—數字」組合。字母代表棋盤方格之橫座標，數字代表棋盤方格之縱座標。

Sample Input（範例輸入）

D4A3A8H1H8

在此例中，國王開始時在 D4 的位置，另有四個武士，分別在 A3、A8、H1、及 H8 的位置。

Output Data (輸出資料)

檔案 CAMELOT.OUT 應只有一個整數，代表完成聚會所需之最少移動步數。

Sample Output（範例輸出）

10

Constraints (限制)

0 ≤ 武士數≤ 63

十、參考解答

試作者：江盈宏(IOI'98 銅牌獎)

```
#include<stdio.h>
/* 在座標盤面上的位置結構 */
struct Pos{
   int x,y;
};
/* 變數宣告:
    king,knight[]是 input
    mindistance 是指某位置至某位置以騎士走法的最短步數
    knights 為武士數
    mindist 是最後的答案,開始時設為一個很大的值
*/
struct Pos king,knight[63];
int mindistance[8][8][8][8];
int knights;
int mindist=32767;
/* 讀檔函式,並將 input 轉為座標 */
```

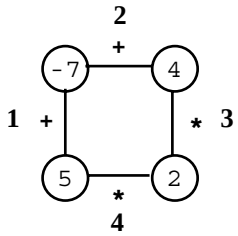
```
void GetData()
{
   FILE *inf;
   char input[150];
   int i;

   inf=fopen("camelot.in","r");
   fscanf(inf,"%s",input);

   king.x=input[0]-'A';
   king.y=input[1]-'1';

   for(i=2;input[i];i+=2) {
      knight[knights].x=input[i]-'A';
      knight[knights].y=input[i+1]-'1';
      knights++;
   }
   fclose(inf);
}
/* 此函式用以計算從 (x,y)出發到某一點以騎士走法所需的最短步數 */
/* 因終點較多,以 BFS 的方法處理,較以遞迴方法快 */
void StartAt(int x,int y)
{
   struct Pos queue[1000];
   int
moveway[8][2]={{1,2},{2,1},{1,-2},{2,-1},{-1,2},{-1,-2},{-2,1},{-2,-1}};
   int i,front=0,rear=0,nx,ny,steps=0,fx,fy;

   mindistance[x][y][x][y]=steps;
   queue[front].x=x+steps*100,queue[front].y=y,front++;
   while(front!=rear) {
      steps=queue[rear].x/100;
      nx=queue[rear].x%100;
      ny=queue[rear].y;
      rear++;
      steps++;

      for(i=0;i<8;i++) {
         fx=nx+moveway[i][0];
         fy=ny+moveway[i][1];
         if(fx<8 && fx>=0 && fy<8 && fy>=0)
            if(mindistance[x][y][fx][fy]>steps) {
               mindistance[x][y][fx][fy]=steps;
               queue[front].x=fx+steps*100;
               queue[front].y=fy;
               front++;
            }
      }
   }
}
/* 此函式為計算所有點至所有點所需的最短騎士走法步數 */
void InitMinDistance()
```

```
{
   int i,j,k,l;
   for(i=0;i<8;i++)
      for(j=0;j<8;j++)
         for(k=0;k<8;k++)
            for(l=0;l<8;l++)
               mindistance[i][j][k][l]=20;

   for(i=0;i<8;i++)
      for(j=0;j<8;j++)
         StartAt(i,j);
}
/* 主要解題函式 :
   i,j 兩變數代表所有的武士與國王最終要到的地方;k,l 代表國王與某一武士重疊時的位置;
org 為國王與武士不重疊時所需步數；而 changed 則表示重疊後所需步數。因此 org-changed
即代表可以減少的步數，subsmax 為這些步數中最大者,所以 nowdist-subsmax 即為數對(i,j,k,l)
所代表的最短距離，至於答案 mindist 則為 nowdist 中最小者*/
void Solve()
{
   int i,j,k,l,m;
   int nowdist,subsmax;
   int org,changed;

   InitMinDistance();

   for(i=0;i<8;i++)
      for(j=0;j<8;j++)
         for(k=0;k<8;k++)
            for(l=0;l<8;l++) {
               nowdist=abs(king.x-i)+abs(king.y-j);
               for(m=0;m<knights;m++)
                  nowdist+=mindistance[knight[m].x][knight[m].y][i][j];

               subsmax=0;
               for(m=0;m<knights;m++) {
                  org=abs(king.x-i)+abs(king.y-j) +
                        mindistance[knight[m].x][knight[m].y][i][j];

                  changed=abs(king.x-k)+abs(king.y-l) +
                              mindistance[knight[m].x][knight[m].y][k][l] +
                              mindistance[k][l][i][j];

                  if(org-changed>subsmax)
                     subsmax=org-changed;
               }
               nowdist-=subsmax;
               if(mindist>nowdist)
                  mindist=nowdist;
            }
}
/* 輸出至檔案的函式 */
void Output()
```

```
{
   FILE *outf;
   outf=fopen("camelot.out","w");
   fprintf(outf,"%d\n",mindist);
   fclose(outf);
}
/* 主函式 */
int main()
{
   GetData();
   Solve();
   Output();
   return 0;
}
```

十一、第二天第三題：多邊形

多邊形是一個人玩的遊戲。開始時在一含有 N 個節點的多邊形上玩，如圖一所示，在此圖中，N=4。每一個節點以一個整數標示; 每一個邊則以符號 + (加) 或符號 * (乘) 標示。這些邊的編號從 1 到 N。



圖一： 一多邊形的圖形表示法

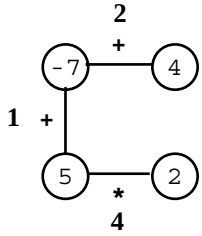
下第一步時，先移除任何一邊，之後的下法如下：

- 任取一邊，稱之為E，並令其兩端的節點為V_1及V_2；
- 先執行E與V_1，V_2之運算，並將結果標示於一個新節點，用來取代舊節點及邊。

當所有的邊都被移除時，遊戲即結束，而得分(score)則為最後留下之節點標示之值。

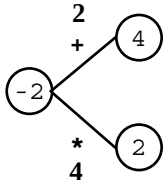
遊戲範例(Sample Game)：

考慮圖一之多邊形遊戲。玩家一開始移除標示為 3 的邊。產生如圖二之結果。



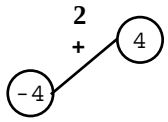
圖二： 移除標示為 3 的邊

接著，玩家選取標示為 1 的邊，



圖三： 選取標示為 1 的邊

接著，標示為 4 的邊，



圖四： 選取標示為 4 的邊

最後，標示為 2 的邊。得分為 0。

圖五： 選取標示為 2 的邊

任務(Task)

給定一個多邊形，寫一程式計算最高的可能得分，找出在第一步先移除就可導致最高得分的邊，並將所有這種邊的編號列出。

輸入資料(Input Data)

檔案 POLYGON.IN 描述一個含有 N 個節點的多邊形。此檔案含有二個橫列。第 1 橫列是數字 N。第 2 橫列含有編號 1 到 N 邊的標示。中間交錯著節點的標示 (首先是編號為 1 和 2 的兩邊之間的節點，接著是編號為 2 和 3 的兩邊之間的節點，以此類推，最後是編號為 N 和 1 的兩邊之間的節點)，均以一個空格隔開。一個邊的標示即是字母 t (表示+)或是字母 x (表示*)。

輸入範例(Sample Input):

```
4
t   -7   t   4   x   2   x   5
```

這是如圖一的多邊形的輸入檔。第二橫列以編號為 1 的邊開始。

輸出資料(Output Data)

在檔案 POLYGON.OUT 的第一橫列中，針對輸入的多邊形，你的程式必須輸出最高的可能得分。在第二橫列中，一個一個列出如在第一步先移除就可導致最高得分的邊的編號。這些邊必須依遞增的編號次序列出，並以一個空格隔開。

輸出範例(Sample Output):

```
33
1  2
```

這是如圖一之多邊形所應得的輸出檔。

條件限制(Constraints)

$3 < N \leq 50$

對任意的下法過程中，節點的標示之範圍為[-32768, 32767]。

十二、參考解答

試作者：鍾至衡(IOI'98 銅牌獎)

/*首先看題目就知道, 這題要暴力來做一定會 Time too long 的。因為假設有 n 個運算子, 那我們就得試 n!次才能挑出一個最大的結果。n = 10 時就已經要 3628800 次了。所以我們必須想更好的方法。我們先來定義一些符號以便下面的解釋: n:共有從 n 個數字, 編號從 1 到 n。MAX(a, b) : 從 a 到 b 這一串數字中可求得的最大值。注意 MAX(a, b)和 MAX(b, a)是不同的。MIN(a, b) : 從 a 到 b 這一串數字中可求得的最小值。La:第 a 個運算元, 也就是第 a 個數字左邊的運算元。這裡我們用的方法是動態規劃(Dynamic Programming)。首先,我們已知 MAX(a, a) = MIN(a, a) = a。又 if L2 == '+' MAX(1, 2) = MAX(1, 1) + MAX(2, 2); MIN(1, 2) = MIN(1, 1) + MIN(2, 2); else MAX(1, 2) = MAX(1, 1) * MAX(2, 2) 或 MAX(1, 1) * MIN(2, 2) 或 MIN(1, 1) * MAX(2, 2)或 MIN(1, 1) * MIN(2, 2)中的最大值; MIN(1, 2) = MAX(1, 1) * MAX(2, 2) 或 MAX(1, 1) * MIN(2, 2)或 MIN(1, 1) * MAX(2, 2)或 MIN(1, 1) * MIN(2, 2) 中的最小值;

為什麼我們需要最小值呢？因為可能有負號，所以最小乘最小可能會變成最大的。再來要求 MAX(1, 3)和 MIN(1, 3)，就分別算出 MAX/MIN(1, 2), MAX/MIN(2, 3)的最大最小值，再分別和 MAX/MIN(3, 3)及 MAX/MIN(1, 1)運算。往上擴展，也就是要算 MAX/MIN(a, b)，就得先有 MAX/MIN(a, x)和 MAX/MIN(x + 1, b)的值，其中 x 是所有 a <= x < b 的整數。然後就以上面那種方法找出一對最大，一對最小的值填進去，就可以再算下一層了。最後就可以算出最後一層 MAX(1, n), MAX(2, 1), MAX(3, 2), ..., MAX(n, n - 1)的值。從中挑一個最大的就是解了。*/

```
#include <stdio.h>
#define     use(c)     (((c) == n) ? n : ((c) % n))
long     num[51];
char     letter[51];
int      n;
long     max[51][51];
long     min[51][51];
long     test;
long     allmax;
int      first;

int main(void)
{
FILE     *fp;
int      i, j, k;
char     temp[2];
int      t;

fp = fopen("polygon.in", "r");
fscanf(fp, "%d", &n);
for(i = 1; i <= n; i++)
     {
     fscanf(fp, "%s%ld", temp, &num[i]);
     letter[i] = temp[0];
     }
fclose(fp);
for(i = 1; i <= n; i++)
     {
     max[i][1] = min[i][1] = num[i];
     for(j = 2; j <= n; j++)
          {
          max[i][j] = -2000000000l;
          min[i][j] = 2000000000l;
          }
     }
//  這裡就是 Dynamic Programming 的過程.
for(j = 2; j <= n; j++)
     for(i = 1; i <= n; i++)
          for(k = 1; k < j; k++)
               {
               t = (i + k) % n;
               if(t == 0)
                    t = n;
               if(letter[t] == 't')
                    {
```

```
                test = max[i][k] + max[use(i + k)][j - k];
                if(test > max[i][j])
                    max[i][j] = test;
                test = min[i][k] + min[use(i + k)][j - k];
                if(test < min[i][j])
                    min[i][j] = test;
                }
            if(letter[t] == 'x')
                {
                test = max[i][k] * max[use(i + k)][j - k];
                if(test > max[i][j])
                    max[i][j] = test;
                if(test < min[i][j])
                    min[i][j] = test;
                test = max[i][k] * min[use(i + k)][j - k];
                if(test > max[i][j])
                    max[i][j] = test;
                if(test < min[i][j])
                    min[i][j] = test;
                test = min[i][k] * max[use(i + k)][j - k];
                if(test > max[i][j])
                    max[i][j] = test;
                if(test < min[i][j])
                    min[i][j] = test;
                test = min[i][k] * min[use(i + k)][j - k];
                if(test > max[i][j])
                    max[i][j] = test;
                if(test < min[i][j])
                    min[i][j] = test;
                }
            }
allmax = -2000000000l;
// 這裡選出最後一層中最大的值.
for(i = 1; i <= n; i++)
    if(allmax < max[i][n])
        {
        allmax = max[i][n];
        first = i;
        }
fp = fopen("polygon.out", "w");
fprintf(fp, "%ld\n%d", allmax, first);
for(i = 1; i <= n; i++)
    if(allmax == max[i][n] && i != first)
        fprintf(fp, " %d", i);
fprintf(fp, "\n");
fclose(fp);
return(0);
}
```