2000 年 IOI 試題與參考解答

1、中間數(Median Strength)

題目(PROBLEM)

一個新的太空實驗室用到 N 個物件，其代號從 1 到 N，且 N 是奇數。每一個物件都有不同且未知的能量值，這些值為自然數。每一個能量值 Y 都滿足 1<=Y<=N 的條件。你必須寫一個程式，來找出能量值為中間數的物件，換句話說，找出物件 X，讓能量值小於物件 X 能量值的物件數，等於能量值大於物件 X 能量值的物件數。可惜的是，只有一種設備(device)能用來比較能量值的大小：給定三個不同的物件，該設備(device)能找出能量值為中間數的物件。

函式庫（LIBRARY）

給定一個名為 device 的函式庫，此函式庫含有三個動作：

- GetN，一開始就呼叫，且不含任何參數；回傳 N 的值。
- Med3，呼叫時需給 3 個不同物件的代號當參數；回傳這 3 個物件中能量值為中間數的物件代號。
- Answer，欲結束時呼叫，有一個參數，就是能量值為中間數的物件代號；並結束程式的執行。

此 device 函式庫會產生兩個文字檔：MEDIAN.OUT 及 MEDIAN.LOG。MEDIAN.OUT 的第一行含一整數：呼叫 ANSWER 時所傳的參數值。第二行也含一整數：你的程式呼叫 Med3 的次數。而 MEDIAN.LOG 則是你的程式與函式庫之間的呼叫記錄。

Pascal 的程式設計者：請在程式加入下列指令

```
uses device;
```

C/C++的程式設計者：請在程式內加入下列指令

```
#include "device.h"
```

並新建一個名為 MEDIAN.PRJ 的計畫（project）。並將 MEDIAN.C（MEDIAN.CPP）及 DEVICE.OBJ 加到此計畫中。

試驗（EXPERIMENTATION）

你如果要用此函式庫來做些試驗就必須先建立一個文字檔 DEVICE.IN。此檔有兩行內容。第一行含一整數：物件的個數 N。第二行含非特定順序的 1 至 n 整數：其中第 i 個整數為第 i 個物件的能量值。

範例（EXAMPLE）

DEVICE.IN

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

上述 DEVICE.IN 檔表示有 5 個物件且每個物件能量為：

Label	1	2	3	4	5
Strength	2	5	4	3	1

正確的解答可如下的 5 個函式庫呼叫：

1. GetN (如用 Pascal）或 GetN() (如用 C/C＋＋）回傳 5。
2. Med3(1, 2, 3) 回傳 3。
3. Med3(3, 4, 1) 回傳 4。
4. Med3(4, 2, 5) 回傳 4。
5. Answer(4)

限制條件（CONSTRAINTS）

- 如 N 代表物件，則 N 為奇數且 5<=N<=1499。
- 如 i 代表物件代號，則 1<=i<=N。
- 如 Y 代表物件能量值，則 1<=Y<=N，且所有能量值都不同。
- Pascal 函式庫名稱為：device.tpu
- pascal 函式宣告為

```
function GetN : integer;
function Med3(x, y, z : integer) : integer;
procedure Answer(m : integer);
```

- C/C++函式庫名稱為：device.h，device.obj（請用 large 記憶體模式）
- C/C++函式宣告（header);

```
int GetN(void);
int Med3(int x, int y, int z);
void Answer(int m);
```

- 程式不可呼叫 Med3 函式多於 7777 次。
- 你的程式不可存取其他任何檔案。

解答

```
/* 王尹 */
/*
使用三分搜索法來把全部的物件間的順序定出來
另外，當加入的物件數為 n 的一半時
不難看出，最邊邊的兩個是用不到的
不但它們不會是 Median，把它們刪除，也不會影響結果

程式碼本身，就應該說明得很清楚了
*/

#include"device.h"

int NumberOfObj,NumberOfExist,Bound;
int Sequence[751];

void Insert(int a){ //用來加入第 a 號物件
 int left=-1,right=NumberOfExist;

 while(right-left>=3){
  int t=(right-left)/3;

  int k=Med3(a,Sequence[left+t],Sequence[right-t]);

  if(k==Sequence[left+t])
   right=left+t;
  else if(k==Sequence[right-t])
   left=right-t;
  else{
   left=left+t;
   right=right-t;
  }
 }

 if(right-left==2){
  int k=Med3(a,Sequence[left],Sequence[left+1]);
  if(k==a)
   right=left+1;
  else
   left=left+1;
 }
```

```
  int i;
  for(i=NumberOfExist;i>right;i--)
    Sequence[i]=Sequence[i-1];
  Sequence[right]=a;

  NumberOfExist++;
}

void main(){
  NumberOfObj=GetN();
  Bound=(NumberOfObj+1)>>1; //二分之一

  Sequence[0]=1;
  Sequence[1]=2;
  NumberOfExist=2;

  int k=3;
  while(k<=Bound){
    Insert(k);
    k++;
  }

  while(k<=NumberOfObj){
    Insert(k);

    int i; //最邊邊的兩個丟掉
    for(i=0;i<NumberOfExist-2;i++)
      Sequence[i]=Sequence[i+1];
    NumberOfExist-=2;

    k++;
  }

  Answer(Sequence[0]);
}
```

2、區塊堆疊(Building with Blocks)

問題 (PROBLEM)

一個單位立方體為 1 乘 1 乘 1 (1×1×1) 之方塊， 而其角落皆座落於整數 x, y, z 之三軸座標上。一個三度空間的實心物體 (簡稱為"實心物體") 為至少一個相連的單位立方體 (見圖示一) ，一個實心物體之體積為所包含單位立方體的數目。一個區塊是一個體積不超過 4 的實心物體，如果一個區塊可以藉著一連串旋轉及位移轉變成另一個區塊 (非鏡射)，兩個區塊就是同一類型。本題有 12 種區塊型態 (見圖示二)， 圖示中顏色僅用於協助辨別實心物體的結構， 顏色並無任何作用。

在滿足下列條件下， 我們稱一個實心物體 S 可被分解成一組 D 的區塊： 此組 D 之區塊可組成 S， 且 D 中之任兩個區塊不可共用同一個單位立方體。

你的任務是寫一個程式， 在給予可能之區塊型態及一個實心物體 S 的情形下， 將 S 分解成最小數目的區塊。你只需輸出分解後每個區塊的型態。

輸入 (INPUT)

在所有輸入檔中， 任一單位立方體皆用可將(x + y + z)值最小化之角落的 x, y, z 座標值表示。

TYPES.IN 描述所有可能區塊型態，此檔案將敘述如下並用於所有測試。此檔將包含圖示二所展示之 12 個區塊型態，並依型態號碼排序。每一種區塊型態以數個連續行表示：第一行為一個整數 I ($1 \le I \le 12$)，其表示區塊型態種類。第二行為一個整數 V ($1 \le V \le 4$)，其表示區塊型態的體積。在接下來之 V 行，每行各包含三個整數 x, y, z 來描述其中一個單位立方體 ($1 \le x, y, z \le 4$)。

BLOCK.IN 描述一個實心物體：第一行有一個值 V ($1 \le V \le 50$) 來描述其體積。接下來 V 行，每行各包含三個整數 x, y, z 來描述其中的一個單位立方體($1 \le x, y, z \le 7$)。

輸出 (OUTPUT)

輸出檔為 BLOCK.OUT。第一行為一個整數 M，而 M 是分解所給予實心物體之最小區塊數目。第二行為此 M 個區塊之區塊型態。一個輸入檔可能有多個可能分解方式，而你只需提供一個分解方式。

輸入及輸出範例(EXAMPLE INPUT AND OUTPUT)

TYPES.IN

```
1
1
1 1 1
2
2
1 1 1
1 2 1
3
3
1 1 1
1 2 1
1 3 1
4
3
1 1 1
1 2 1
1 1 2
5
4
1 1 1
1 2 1
1 3 1
1 4 1
6
4
1 1 1
1 2 1
1 1 2
1 2 2
7
4
1 1 1
1 2 1
1 1 2
1 1 3
8
4
1 1 1
1 2 1
1 3 1
1 2 2
9
4
1 2 1
1 3 1
1 1 2
1 2 2
10
4
2 1 1
1 2 1
2 2 1
2 1 2
11
4
1 1 1
1 2 1
2 2 1
1 1 2
12
4
2 2 1
2 1 2
1 2 2
2 2 2
```

BLOCKS.IN

```
18
2 1 1
4 1 1
2 3 1
4 3 1
2 1 2
3 1 2
4 1 2
1 2 2
2 2 2
3 2 2
4 2 2
2 3 2
3 3 2
4 3 2
4 2 3
4 2 4
4 2 5
5 2 5
```

BLOCK.OUT

```
5
7   10   2 10   12
```

注意 (NOTE)

1. 輸入檔 BLOCK.IN 描述圖示一之”馬”形狀的實心物體。
2. 其他分解方式如下：以下各行為可能解法輸出檔之第二行（即各區塊型態）。

   2 7 10 11 12
   2 7 11 11 12
   4 4 7 10 11
   4 4 9 10 11

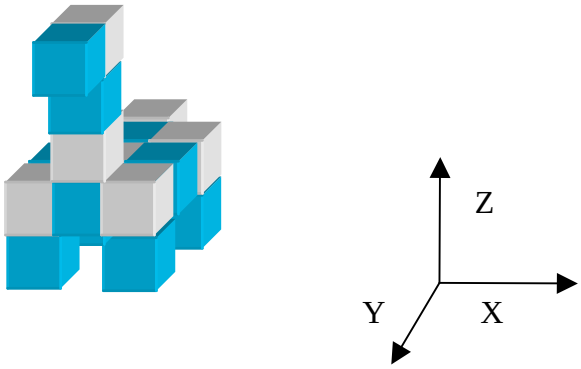

圖 1. Horse

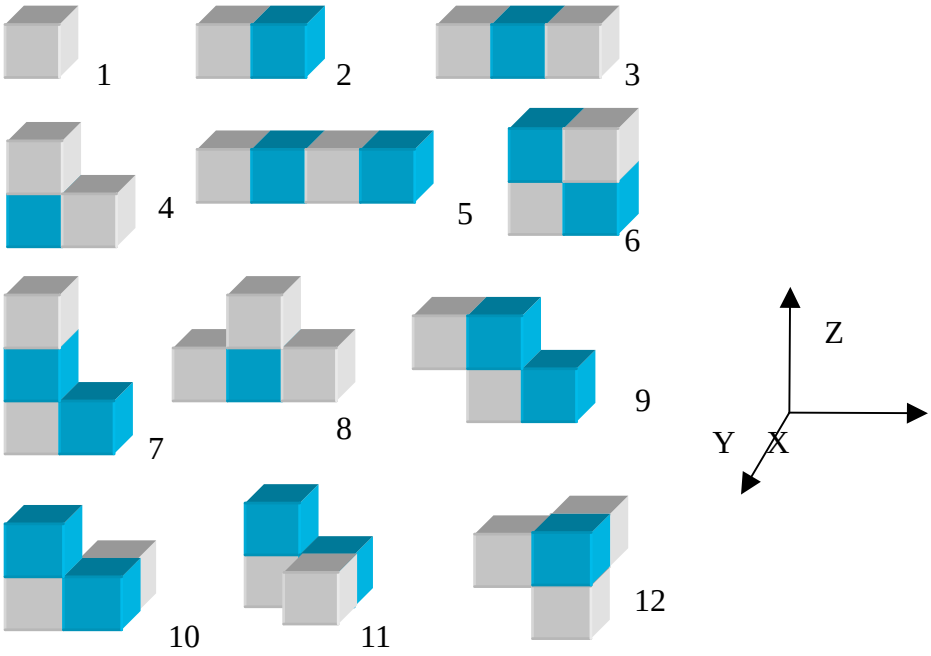

圖 2. The 12 block types

解答

```
/* 張琮翔 */
/*
    BLCOK 是 IOI 歷年來少見的 Search 題，而且程式寫起來也很複雜及龐大，以解題方法來說是六題中最單純的，但 implement 卻是最複雜及最困難的。

    這題的演算法很簡單，只有 Branch & Bound 而已，但 Bound 得選得好才能快速的剪枝。最直接的 Lower Bound 就是 V/4，因為小積木最大只有 4*4 而已...有了此 Bound 後，再把積木由大到小一個一個放放看就可以剪去許多分枝了。

    而這題 implement 的困難點在於積木的旋轉上，每一個積木最多能轉出 24 種不同積木出來，而三度空間的旋轉乍看之下很難實做，但其實只要先從二維平面的旋轉出發，就能發現三度空間的旋轉和二維平面實際上是一模一樣的（這題有個小地方要注意，題目上給的座標竟然是從 1 開始，而不是一般我們熟悉的 0，如果座標以 1 為最小值，那麼旋轉會變得非常難實做....所以我在做這題時把全部的座標全部平移至原點，也就是(0,0,0)，這樣子旋轉就只是簡單的映射而已。

    只要知道這點，然後旋轉後要記得去除重複的積木，在拿剩下來的積木去 Search 就行了。
*/
#include<stdio.h>
#include<stdlib.h>

typedef struct _block /* 宣告小積木的 struct */
{
    int i; /* 小積木編號 */
    int v;     /* 小積木體積 */
    int x[4],y[4],z[4];   /* 小積木各點座標 */
}Block;

typedef struct _object  /* 小積木拼起來大物體的 struct */
{
    int v;     /* 物體體積 */
    int x[50],y[50],z[50];    /* 物體各點座標  */
    int maxx,maxy,maxz;  /* 為了方便,把物體 x,y,z 座標的最大值也記錄下來 */
}Object;
Block block[12*24];
Object obj;
int bcount;    /* 小積木總數 */
int leftv;   /* DFS 用的,記錄物體剩下多少體積還沒拼出來 */
int usecount[12];   /* 記錄每一種方塊用了幾個 */
int map[7][7][7];   /* 物體的 3 維對應 map */
```

```
void swap(int *a,int *b)
{
	int tmp=*a;
	*a=*b;
	*b=tmp;
}
Block shift(Block b)	/* 把積木 b 平移回原點 */
{
	int i,j,mx=100,my=100,mz=100;
	for(i=0;i<b.v;i++)
	{
		if(b.x[i]<mx)	mx=b.x[i];
		if(b.y[i]<my)	my=b.y[i];
		if(b.z[i]<mz)	mz=b.z[i];
	}
	for(i=0;i<b.v;i++)
	{
		b.x[i]-=mx;
		b.y[i]-=my;
		b.z[i]-=mz;
	}
	for(i=0;i<b.v;i++)
		for(j=i;j<b.v;j++)
		{
			if(b.x[i]>b.x[j])
			{
				swap(&b.x[i],&b.x[j]);
				swap(&b.y[i],&b.y[j]);
				swap(&b.z[i],&b.z[j]);
			}
			else if(b.x[i]==b.x[j])
				if(b.y[i]>b.y[j])
				{
					swap(&b.y[i],&b.y[j]);
					swap(&b.z[i],&b.z[j]);
				}
				else if(b.y[i]==b.y[j])
					if(b.z[i]>b.z[j])
						swap(&b.z[i],&b.z[j]);
		}
	return b;
}
```

```
Block rot(Block b,int ii,int n)      /* 以 ii 為軸,順時鐘旋轉積木 b 90 度 n 次 */
{
    /* ii    0   1   2
       axis  x   y   z   */
    int *p1,*p2;
    int *s1,*s2;
    Block tmp;
    int i;
    if(n==0)  return b;
    tmp=b;
    switch(ii)
    {
        case 0:  /* 以 x 為軸做旋轉,所以是在 yz 平面上旋轉 */
                 p1=&tmp.y[0];
                 p2=&tmp.z[0];
                 s1=&b.y[0];
                 s2=&b.z[0];
                 break;
        case 1:  /* 以 y 為軸做旋轉 */
                 p1=&tmp.x[0];
                 p2=&tmp.z[0];
                 s1=&b.x[0];
                 s2=&b.z[0];
                 break;
        case 2: /* 以 z 為軸做旋轉 */
                 p1=&tmp.x[0];
                 p2=&tmp.y[0];
                 s1=&b.x[0];
                 s2=&b.y[0];
                 break;
    }
    /* 旋轉的核心演算法,畫個圖出來就很清楚了 */
    /*
                    |
                    |
        (-y1,x1)    | .(x1,y1)(以此點開始做旋轉)
            3.      |
        ------------+----------
                    |   1.(y1,-x1)
                2.  |
      (-x1,-y1)     |
                    |
```

```
            就是這樣子而已，每個平面的旋轉都是一樣的...
        */
        for(i=0;i<b.v;i++)
        {
            switch(n)
            {
                case 1:
                    *p1=*s2;
                    *p2=-*s1;
                    break;
                case 2:
                    *p1=-*s1;
                    *p2=-*s2;
                    break;
                case 3:
                    *p1=-*s2;
                    *p2=*s1;
                    break;
            }
            p1++;    p2++;
            s1++;    s2++;
        }
        return tmp;
}
int check(Block b)/* 檢查積木 b 有沒有出現過了 */
{
        int i,j;
        for(i=0;i<bcount;i++)
            if(b.v==block[i].v)
            {
                for(j=0;j<b.v;j++)
                    if(b.x[j]!=block[i].x[j] || b.y[j]!=block[i].y[j] || b.z[j]!=block[i].z[j])
                        break;
                if(j==b.v)return 1;
            }
        return 0;
}
```

```
void rotate(Block b)    /* 把積木 b 做所有可能的旋轉 */
{
    int i,j,k;
    Block tx=b,ty=b,tz=b;
    /*  這裡我是把每一個方向 x,y,z 全部都轉 3 次,總共要轉 3*3*3 次
        所以當然會有重複的，重複的再檢查去除
    */
    for(i=0;i<4;i++)
    {
        tx=rot(b,0,i);
        for(j=0;j<4;j++)
        {
            ty=rot(tx,1,j);
            for(k=0;k<4;k++)
            {
                tz=rot(ty,2,k);
                tz=shift(tz);
                if(!check(tz)) /* 檢查如果沒有重複就得到新的一種積木 */
                    block[bcount++]=tz;
            }
        }
    }
}
int cmp(const void *a,const void *b) /* Qsort 用的排序函式,把積木從大排到小 */
{
    return ((Block *)b)->v - ((Block *)a)->v;
}
void shiftobj()/* 把物體位移到原點(0,0,0) */
{
    int i,mx=100,my=100,mz=100;
    for(i=0;i<obj.v;i++)
    {
        if(obj.x[i]<mx)mx=obj.x[i];
        if(obj.y[i]<my)my=obj.y[i];
        if(obj.z[i]<mz)mz=obj.z[i];
    }
    for(i=0;i<obj.v;i++)
    {
        obj.x[i]-=mx;
        obj.y[i]-=my;
        obj.z[i]-=mz;
        if(obj.x[i]>obj.maxx)    obj.maxx=obj.x[i];
        if(obj.y[i]>obj.maxy)    obj.maxy=obj.y[i];
        if(obj.z[i]>obj.maxz)    obj.maxz=obj.z[i];
        map[obj.x[i]][obj.y[i]][obj.z[i]]=1;
    }
}
```

```
int canput(Block b,int x,int y,int z)   /* 測試積木 b 能不能放在(x,y,z)上 */
{
      int i;
      for(i=0;i<b.v;i++)
            if(map[x+b.x[i]][y+b.y[i]][z+b.z[i]]==0)
                  return 0;
      return 1;
}
void putblock(Block b,int x,int y,int z) /* 把積木 b 放到(x,y,z)上 */
{
      int i;
      for(i=0;i<b.v;i++)
            map[x+b.x[i]][y+b.y[i]][z+b.z[i]]=0;
}
void getblock(Block b,int x,int y,int z) /* 把已放下去的積木 b 從(x,y,z)拿起來 */
{
      int i;
      for(i=0;i<b.v;i++)
            map[x+b.x[i]][y+b.y[i]][z+b.z[i]]=1;
}
int dfs(int ii,int deep,int maxdeep)
/* ii:目前用到的積木編號 deep:目前已經擺了幾塊積木 maxdeep:最多要用幾塊 */
{
      int i,j,k;
      if(deep==maxdeep)
      {
            if(leftv==0)    /* 如果剛好用了 maxdeep 塊而且物體也全部都被拼出來
了，那就找到囉... */
                  return 1;
            return 0;
      }

      /* 如果物體剩下的體積>目前物體體積*(最多積木數-已用積木數）就不用往下
找了 */
      /* 也就是說就算一直遞到最後一層也不可能把 leftv 給減到 0 的... */
      if(leftv>block[ii].v*(maxdeep-deep))  return 0;

      /* 如果目前積木體積比物體剩下的還大,那就往下用比較小的積木 */
      while(ii<bcount && block[ii].v>leftv)  ii++;
      while(ii<bcount)
      {
            for(i=0;i<=obj.maxx;i++)
                  for(j=0;j<=obj.maxy;j++)
                        for(k=0;k<=obj.maxz;k++)
                              if(canput(block[ii],i,j,k))
                              {
                                    putblock(block[ii],i,j,k);
                                    leftv-=block[ii].v;
                                    usecount[block[ii].i-1]++;
                                    if(dfs(ii,deep+1,maxdeep)) return 1;
```

```
                                usecount[block[ii].i-1]--;
                                leftv+=block[ii].v;
                                getblock(block[ii],i,j,k);
                        }
                ii++;
        }
        return 0;
}
void main(void)
{
        int i,v,x,y,z,j;
        int pflag=1;
        bcount=0;
        freopen("types.in","r",stdin);
        while(scanf("%d",&i)==1)   /* 從 types.in 把小積木資料讀進來 */
        {
                scanf("%d",&v);
                block[bcount].i=i;
                block[bcount].v=v;
                for(j=0;j<v;j++)
                {
                        scanf("%d %d %d",&x,&y,&z);
                        block[bcount].x[j]=x;
                        block[bcount].y[j]=y;
                        block[bcount].z[j]=z;
                }
                block[bcount]=shift(block[bcount]);
                bcount++;
                rotate(block[bcount-1]);      /* 把小積木旋轉.. */
        }
        qsort(block,bcount,sizeof(Block),cmp); /* 從大到小把積木排序好 */
        freopen("block.in","r",stdin);
        scanf("%d",&obj.v);
        for(i=0;i<obj.v;i++)
        {
                scanf("%d %d %d",&x,&y,&z);
                obj.x[i]=x;
                obj.y[i]=y;
                obj.z[i]=z;
        }
        shiftobj();             /* 把物體平移至原點上 */
        for(i=obj.v/4,leftv=obj.v;i<=obj.v;i++)/* 這裡是 Search 的核心,每次都嘗試看看
能不能用 i 塊積木堆起來 */
                if(dfs(0,0,i))   /* 只要一找到能堆出物體的方法，那就可以直接跳離了，因
為這一定是最少積木數 */
```

```
            break;
        printf("%d\n",i);
        for(i=0;i<12;i++)
            for(j=0;j<usecount[i];j++)
            {
                if(pflag)
                    pflag=0;
                else
                    putchar(' ');
                printf("%d",i+1);
            }
        putchar('\n');
}
```

3、迴文 (Palindrome)

題目(PROBLEM)

所謂「迴文」,是一種對稱的字串。該字串由左往右讀,或由右往左讀,內容都相同。你的任務是設計一個程式,在加入最少的字元的情況下,將一個輸入的字串轉變成「迴文」。

舉個例子,要將字串"Ab3bd"轉換成「迴文」,只需加入 2 個字元,即可得到"dAb3bAd"或"Adb3bdA"。然而,只加入 1 個字元,不論放於何處,絕對不可能將字串"Ab3bd"轉換成「迴文」。

輸入(INPUT)

輸入檔的檔名是 PLAIN.IN。檔案中的第一行是一個整數,其為輸入字串的長度 N (3<=N<=5000)。第二行則是一個長度為 N 的字串。該字串是由'A'到'Z'的大寫英文字母,'a'到'z'的小寫英文字母,以及'0'到'9'的數字所組成。大寫字母和小寫字母需視為不同的字元。

輸出(OUTPUT)

輸出檔的檔名是 PLAIN.OUT。請在該檔案的第一行放一個整數,該整數就是要將輸入字串轉變成迴文所需加入的最少字元數。

輸入與輸出範例(EXAMPLE INPUT AND OUTPUT)

PALIN.IN

```
5
Ab3bd
```

PALIN.OUT

```
2
```

解答

```
/* 許恒瑞 */
/*
    這題是第一天中最簡單的一題，可以利用最長共同子序列 LCS(Longest Common Sequence)實作出來，要點就是把輸入的字串反向後和之前的字串作 LCS，便可得到頭尾相同的最長子字串。*/

#include<stdio.h>
#include<string.h>

int len;    // 輸入字串長度
char s1[5001], s2[5001];   // s1 字串存正向，s2 字串存反向

int Max(int n1, int n2)
{
    return n1 > n2 ? n1 : n2;
}

int LCS()
{
   /*    這裡為實作 LCS，傳回共同子字串的長度      */

    int table[2][5001], now = 0, i, j;
    memset(table, 0, sizeof(table));
    for(i = 0; i < len; i++, now = !now)
        for(j = 0; j < len; j++)
            if(s1[i] == s2[j])
                table[now][j + 1] = table[!now][j] + 1;
            else
                table[now][j + 1] = Max(table[now][j], table[!now][j + 1]);
    return table[!now][len];
}

void Getdata()
{
    int ans;
    scanf("%d", &len);
    scanf("%s", s1);
    strcpy(s2, s1);
    strrev(s2);
    ans = len - LCS();
    printf("%d\n", ans);
}
```

```
int main()
{
	freopen("palin.in", "r", stdin);
	freopen("palin.out", "w", stdout);
	Getdata();
	return 0;
}
```

4、停車問題(Car Parking)

題目(PROBLEM)

一個在長城旁的停車中心有一橫列的停車位，橫列的一個端點為左端，另一個端點為右端，橫列上停滿車子，每輛車子屬於一種車型，而同車型的車子可能有數輛。每種車型以一個整數為代號。一群員工試圖將橫列上的車子就下列方法由左至右將車子以車型代號遞增順序排列：在每一輪排序中，每一個員工可同時由一個停車位將車開出，並將此車停入任一個在同一輪排序中所空出之車位。在任一輪排序中，可以允許有些員工不參與移動車子。就效率而言，我們希望以最少輪排序將車子重新排列。

假設 N 為車子總數,W 為員工總人數。在所給定車子的車型與員工總人數的情況下，你的任務是寫一個程式來重新排列車子，並使得所需之排序的輪數不超過「N/(W-1)」。「N/(W-1)」為不小於 N/(W-1)之最小的整數。而最小的排序輪數將不大於「N/(W-1)」。

以下列例子來說：假設有十輛車子停在停車中心的橫列上，車型代號有 1，2，3 及 4 號等數種。員工總人數為 4 人。車子的初始排列由左至右以車型代號表示為

2 3 3 4 4 2 1 1 3 1

在此問題上，最小的排序輪數為 3，在每一輪排序後之車子的車型代號排列如下：

第一輪後：2 1 1 4 4 2 3 3 3 1

第二輪後：2 1 1 2 4 3 3 3 4 1

第三輪後：1 1 1 2 2 3 3 3 4 4

輸入（INPUT）

輸入檔名為 CAR.IN。輸出檔的第一行有三個整數。第一個整數為車子總數 N（2<=N<=20000）。第二個整數為可能的車型代號總數 M（2<=M<=50），車型代號為 1 至 M 之整數。每種車型至少有一輛車子。第三個整數為員工總人數 W（2<=N<=W）。輸入檔第二行有 N 個整數，第 i 個整數為停車位橫列上由左邊數起第 i 輛車子的車型代號。

輸出（OUTPUT）

輸出檔名為 CAR.OUT。輸出檔的第一行只有一個整數 R，即所需的排序輪數。接下來的 R 行描述第一輪到第 R 輪的排序過程。在此，每一行的第一個整數 C 為該輪次所移動的車子數。爾後的 2C 個整數代表車子移動前、後的位置(車位位置是以 1 至 N 之整數從左端算起)。譬如，前兩個數代表其中一輛車子移動的方式：即第一個數為移動前的位置，而第二個數為移動後的位置。再下來的兩個整數以同樣的方式表示另一輛車子由那個車位移入那一個車位。以下每二個整數以此方式描述其他車子之移動。當然，有可能存在一個以上的方法，在同樣的排序輪數下排序車子，你的程式只需輸出一組解即可。

輸入及輸出範例（EXAMPLE INPUT AND OUTPUT）

CAR.IN

```
10  4  4
2  3  3  4  4  2  1  1  3  1
```

CAR.OUT

```
3
4  2  7  3  8  7  2  8  3
3  4  9  9  6  6  4
3  1  5  5  10  10  1
```

部分給分（PARTIAL CREDIT）

假設你的程式輸出為 R 輪排序，而 Q 為 ⌈N/(W-1)⌉。如果輸出檔的任何輪排序輸出錯誤或無法產生要求之正確車子排列，你的分數將為 0 分。除此之外，你的分數將以下列方式計算

- R <= Q　　100%　總分
- R = Q+1　　50%　總分
- R = Q+2　　20%　總分
- R >= Q+3　　0%　總分

解答

```
/* 蔡濬帆 */
/*
這一題要求移動的回合數必須小於 N/(W-1)，有一個很直觀的方法可以達成這個目的，就是把第一部不在正確位置的車開到該去的地方，而被擠出來的車子再去找下一個位置，如此可以構成一個循環。如果還剩下人的話，就再找下一個循環。有時候人手不夠，沒有辦法完成一個循環，那麼最後一部車就無法到達正確位置，不過就算如此也已經有 W-1 部車移到正確位置了。
*/
#include <stdio.h>

void input(void);
void run(void);

const int MAXCAR=20000;
const int MAXNUM=50;

int N, M, W;
int value[MAXCAR];
int eachtotal[MAXNUM];
int locate[MAXNUM];
int locatetop[MAXNUM];
int huge moveway[MAXCAR];
int total;
int runlen[MAXCAR/(MAXNUM-1)];

int findhead(void);
int findnext(int l);
void addmove(int a, int b, int r);

int main(void)
{
	input();
	run();
	return 0;
}
```

```
void input(void)
{
	FILE *in;
	int i;
	in = fopen("car.in", "r");
	fscanf(in, "%d%d%d", &N, &M, &W);
	for(i=0; i<N; ++i) {
		fscanf(in, "%d", &value[i]);
		--value[i];
		++eachtotal[value[i]];
	}
	fclose(in);
	locate[0] = 0;
	for(i=1; i<M; ++i)
		locate[i] = locate[i-1] + eachtotal[i-1];
	for(i=0; i<M; ++i)
		locatetop[i] = locate[i] + eachtotal[i];
}

void run(void)
{
	int run, p, headlocate, nowlocate, nextlocate, i, headvalue;
	FILE *out;
	for(run=0; findhead()!=-1; ++run) {
		p = W;
		while(p>1) {
			headlocate = findhead();
			if(headlocate==-1)
				break;
			nextlocate = headlocate;
			while(p>0) {
				--p;
				nowlocate = nextlocate;
				nextlocate = findnext(value[nowlocate]);
				if(p>0 || nextlocate==headlocate) {
					addmove(nowlocate, nextlocate, run);
					++locate[value[nowlocate]];
				}
				else
					addmove(nowlocate, headlocate, run);
				if(nextlocate==headlocate)
					break;
			}
			//move
			headvalue = value[headlocate];
			for(i=total-2; moveway[i]!=headlocate; i-=2)
				value[moveway[i+1]] = value[moveway[i]];
			value[moveway[i+1]] = headvalue;
		}
	}
```

```
	out = fopen("car.out", "w");
	fprintf(out, "%d\n", run);
	int k, t;
	k = 0;
	for(i=0; i<run; ++i) {
		fprintf(out, "%d", runlen[i]/2);
		for(t=0; t<runlen[i]; ++t, ++k)
			fprintf(out, " %d", moveway[k]+1);
		fprintf(out, "\n");
	}
	fclose(out);
}

int findhead(void)
{
	int i;
	for(i=0; i<M; ++i) {
		while(value[locate[i]]==i && locate[i]<locatetop[i])
			++locate[i];
		if(locate[i]<locatetop[i])
			return locate[i];
	}
	return -1;
}

int findnext(int l)
{
	int i;
	for(i=locate[l]; i<locatetop[l]; ++i) {
		if(value[i]!=l)
			return i;
	}
	return -1;
}

void addmove(int a, int b, int r)
{
	moveway[total] = a;
	moveway[total+1] = b;
	runlen[r] += 2;
	total += 2;
}
```

5、郵局(Post Office)

題目(PROBLEM)

在一條筆直的公路上有數個村落(village)，此公路可用一條整數軸代表之。每一個村落在公路上的位置(position)以整數座標表示，兩個村落不會落在相同的位置上。定義兩個位置(position)的距離是他們的整數座標差之絕對值。

郵局將蓋在某些村落內，但不見得每一個村落都會有郵局。令郵局的位置與其所座落的村落相同。我們希望決定，郵局該蓋在哪些村落，才能使得每一村落至離其最近郵局距離的和為最小。

給定每一個村落的位置及郵局數，請寫一個程式，計算每一村落至離其最近郵局距離之最小總和及所有郵局的位置。

輸入（INPUT）

輸入檔檔名為 POST.IN。第一行有兩個整數：第一個為村落數 V ($1 \le V \le 300$)，而第二個為郵局數 P ($1 \le P \le 30$)，且 $P \le V$ 。第二行則包含 V 個遞增整數，代表每一個村落的位置(position)。每一個位置 X 的限制為 $1 \le X \le 10000$ 。

輸出（OUTPUT）

輸出檔檔名為 POST.OUT。第一行有一整數 S，就是每一村落至離其最近郵局距離之最小總和。第二行則有 P 個遞增整數，每個整數代表每個不同郵局的位置。如果有數組解答，你的程式只需輸出其中一組即可。

輸入及輸出範例（EXAMPLE INPUT AND OUTPUT）

POST.IN

```
10  5
1  2  3  6  7  9  11  22  44  50
50
```

POST.OUT

```
9
2  7  22  44  50
```

部分給分 (PARTIAL CREDIT)

如果郵局位置與你的程式所算出的距離最小和不符則給 0 分。否則你的得分以 Table 1 的方式計算。假如你的程式輸出和 S，而實際最小的解和為 S_{min}，那麼計算 $q = S / S_{min}$，你的分數 c 即如表一所示。

Table 1

$q = S / S_{min}$	$q = 1.0$	$1.0 < q \le 1.1$	$1.1 < q \le 1.15$	$1.15 < q \le 1.2$	$1.2 < q \le 1.25$	$1.25 < q \le 1.3$	$1.3 < q$
c	10	5	4	3	2	1	0

解答

```
/* 許恒瑞 */
/*
    這題是典型的 DP 題，雖然題目是部份給分，但其實只要方法對了，所得到的
都是最佳解，從 N 個村莊裡選 M 個蓋郵局，可以這樣看，一開始都沒有郵局，然後
郵局一個一個加進來，加進來的同時，可以決定這個郵局給那幾個村莊，而且可以
規定郵局一定從一邊加入。

    來舉個例子：
    一開始 10 個村莊要蓋 5 個郵局，先假設最後幾個村莊要共用一個郵局，但究
竟要幾個村莊呢？就每種情況都試試看，決定好了之後就解決一個郵局了，剩下的
村莊仍是連續的，再假設剩下的最後幾個村莊也要共用一個郵局，如此下去，當然，
當你決定要那幾個村莊共用一個郵局的時候，你也要把這一段所需要的成本考慮進
來，因為一個情況都可以由多個較小的情況來說明，底下列一個公式來幫助了解：

    從 N 個村莊裡選 M 個蓋郵局的最小成本
                      N-1
    C(N, M) = Σ     C(i, M - 1) + D(i + 1, N)
                    i=M-1

    D(N1, N2)表示從 N1 到 N2 只有一個郵局時所需的成本
*/

#include<stdio.h>
#include<string.h>
#define Size 300

int N, M;
huge long value[Size], sum[Size][Size], best[Size][30];

/*
    value  是存原本輸入的值
    sum    是存從 N1 到 N2 所需的成本
    best   是存從第 1 個到第 N 個村莊需 M 個郵局的最佳值
*/
```

```
void Init()
{
	int i, j, k, center;
	memset(sum, 0, sizeof(sum));
	memset(best, 0, sizeof(best));
	for(i = 0; i < N; i++)
		for(j = i + 1; j < N; j++)
		{
			center = i + (j - i) / 2;
		/*
			決定幾個村莊共用一個郵局的起點和終點後
			郵局必蓋這幾個村莊的中間，所以先算出郵局蓋在那
			底下的迴圈就是計算所需的成本
		*/
			for(k = i; k <= j; k++)
				if(k >= center)sum[i][j] += value[k] - value[center];
				else sum[i][j] += value[center] - value[k];
		}
	for(i = 0; i < N; i++)best[i][0] = sum[0][i];
}

long Count(int end, int post)
{
	if(post == 0)return sum[0][end];	/*	若已不再需要分開，剩下的全部共用一個郵局	*/
	if(best[end][post])return best[end][post];	/*	若之前已經算過，則直接傳回結果	*/
	long min = 1000000000, temp;
	for(int i = end - 1; i >= post - 1; i--)	/*	計算所有的情況找出此情況的最佳解	*/
	{
		temp = Count(i, post - 1) + sum[i + 1][end];
		if(temp < min)min = temp;
	}
	best[end][post] = min;
	return min;
}
```

```
void Out(int end, int post)
{
/*
    此函式用來回朔找出當郵局的村莊
    也是假設某一種情況，若加入的方法的最佳解會等於目前的最佳解，
    則可以確定那一段的村莊共用一個郵局，進而找出中間的村莊
*/
    for(int i = end - 1; i >= post - 1; i--)
        if(best[i][post - 1] + sum[i + 1][end] == best[end][post])
        {
            Out(i, post - 1);
            printf("%d ", value[i + 1 + (end - i - 1) / 2]);
        }
}

int main()
{
    long ans;
    freopen("post.in", "r", stdin);
    freopen("post.out", "w", stdout);
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++)
        scanf("%ld", &value[i]);
    Init();
    ans = Count(N - 1, M - 1);
    printf("%ld\n", ans);
    Out(N - 1, M - 1);
    putchar('\n');
    return 0;
}
```

6、牆(Walls)

題目(PROBLEM)

有一個國家，建造了一些牆，以連結某些小鎮。這些牆彼此不會相交。所有的牆將整個國家分成一些封閉的區域。如果要從一個區域到另一個區域，可以選擇經過小鎮，或是穿過牆。對於任意兩個小鎮 A 和 B 而言，最多只存在一道牆，一端在小鎮 A，另一端在小鎮 B。只需順著牆走，並在需要時穿過其他小鎮，一定可以從一個小鎮到達另一個小鎮。其他的條件限制可參考輸入檔的格式。

有一個俱樂部，其會員散居於各個小鎮，每個小鎮最多只居住一位會員，有些小鎮可能沒有任何會員居住。當有聚會時，會員會選擇（小鎮外的）某個區域作為聚會的地點。會員們的交通工具是自行車。會員們不可穿過小鎮，以避開小鎮壅塞的交通。會員們也希望穿過的牆數目越少越好，畢竟要穿過牆頗費力氣。每位會員必須穿過某些數目（可能為 0）的牆，才能到達聚會的地點。而在選擇聚會地點時，希望所有的會員所需穿過的總牆數（簡稱，穿牆總數）為最少。

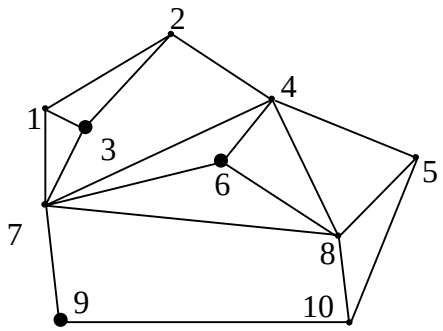

圖 1

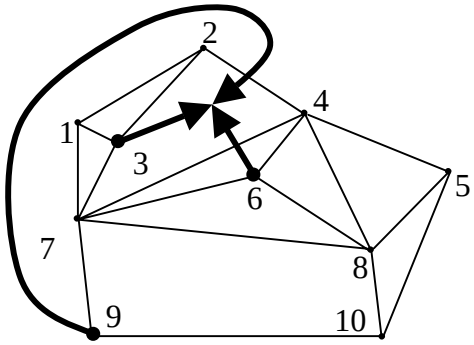

圖 2

小鎮由 1 到 N 的整數來標示，N 即為小鎮的總數。在圖 1 中，具有標示的節點即代表了小鎮，連接各個節點的線段，則代表牆。現假設有三位會員，分別居住於第 3，6，及 9 號小鎮。圖 2 列出了對這三位會員而言，最佳的聚會區域，以及每位會員應採取的路線。本例中的穿牆總數為 2：分別是居住於第 9 小鎮的會員必須穿過連接第 2 與第 4 小鎮的牆，以及居住於第 6 小鎮的會員必須穿過連接第 4 與第 7 小鎮的牆。

請設計一個程式，程式的輸入是小鎮，區域，以及俱樂部會員居住的小鎮的描述，請幫會員們選擇最佳的聚會區域，以及計算出最少穿牆總數。

輸入（INPUT）

輸入檔的名稱為 WALLS.IN。第一行是一個整數，表示區域的數目 M，($2 \le M \le 200$)。第二行也是一個整數，表示小鎮的數目 N ($3 \le N \le 250$)。第三行是一個整數，表示會員的數目 L，($1 \le L \le 30$ ，$L \le N$)。第四行含有 L 個大小不同的整數，由小到大排列，分別表示每位會員所居住的小鎮的代碼。

接下來還有 2M 行，每兩行對應到一個區域。這 2M 行的第 1 和第 2 行對應於第 1 個區域，第 3 和第 4 行對應於第二個區域，依此類推。在對應於每個區域的兩行輸入中，第一行定義了該區域的周邊總共有多少個小鎮（假設以 I 表示），第二行則列出了 I 個整數，就是這 I 個小鎮的代號，其列出的次序，就是如果順著該區域的周邊依順時針的方向行走，依序會碰到的小鎮的次序。唯一的例外是包含了所有的小鎮與所有的區域的外圍區域（這個區域是輸入檔中最後定義的區域）。該區域的小鎮代碼是依照逆時針的次序列出。每個區域依照其在這 2M 行中出現的次序來編號。第一個出現的區域其代碼為 1，第二個出現的區域其代碼為 2，依此類推。敬請留意，輸入資料包含了由所有小鎮和牆所組成的區域，這其中包括了"外圍"的區域。

輸出（OUTPUT）

輸出檔的名稱為 WALLS.OUT。第一行必須是一個整數，就是最少穿牆總數。第二行也需是一個整數，說明最佳的聚會區域代碼。最佳的聚會區域可能有好幾個，你的程式只需輸出其中一個。

輸入及輸出範例（EXAMPLE INPUT AND OUTPUT）

以下的輸入及輸出檔，對應於出現於上文中的範例。

WALLS.IN

```
10
10
3
3 6 9
3
1 2 3
3
1 3 7
4
2 4 7 3
3
4 6 7
3
4 8 6
3
6 8 7
3
4 5 8
4
7 8 10 9
3
5 10 8
7
7 9 10 5 4 2 1
```

```
2
3
```

解答

```
/* 蔡濬帆 */
/*
這一題是最短路徑的應用。先求出任兩個區域間的最短距離(最少穿牆數目)，然後
以每個區域來作測試，看所有的人到那一個區域集合的穿牆數目總和最小。
*/
#include <stdio.h>

void input(void);
void run(void);

const int MAXREGION=201;
const int MAXTOWN=251;
const int MAXPEOPLE=30;
int region, town, people;

int live[MAXPEOPLE];
int huge wallside[MAXTOWN][MAXTOWN];
int huge touch[MAXTOWN][MAXREGION];
int touchnum[MAXTOWN];
unsigned char map[MAXREGION][MAXREGION];

int main(void)
{
	input();
	run();
	return 0;
}

void input(void)
{
	int i, p, k, t, last, first;
	FILE *in;
	in = fopen("walls.in", "r");
	fscanf(in, "%d%d%d", &region, &town, &people);
	for(i=0; i<people; ++i)
		fscanf(in, "%d", &live[i]);
	for(i=1; i<=region; ++i)
	for(k=1; k<=region; ++k)
		map[i][k] = 255;
	for(i=1; i<=region; ++i)
		map[i][i] = 0;
	for(i=0; i<region; ++i) {
		fscanf(in, "%d", &p);
		for(k=0; k<p; ++k) {
			last = t;
			fscanf(in, "%d", &t);
			touch[t][touchnum[t]] = i+1;
			++touchnum[t];
			if(k>0) {
				if(wallside[last][t]!=0) {
```

```
                        map[wallside[last][t]][i+1] = 1;
                        map[i+1][wallside[last][t]] = 1;
                    }
                    else {
                        wallside[last][t] = i+1;
                        wallside[t][last] = i+1;
                    }
                }
                else
                    first = t;
            }
            if(wallside[first][t]!=0) {
                map[wallside[first][t]][i+1] = 1;
                map[i+1][wallside[first][t]] = 1;
            }
            else {
                wallside[first][t] = i+1;
                wallside[t][first] = i+1;
            }
        }
        fclose(in);
}

void run(void)
{
        int i, k, t;
        FILE *out;
        //all pair shortest path
        for(t=1; t<=region; ++t)
        for(i=1; i<=region; ++i)
        for(k=1; k<=region; ++k) {
            if((int)map[i][t]+(int)map[t][k]<(int)map[i][k])
                map[i][k] = map[i][t]+map[t][k];
        }
        int sum, minsum, p, now, meet;
        minsum = -1;
        for(t=1; t<=region; ++t) {
            sum = 0;
            for(i=0; i<people; ++i) {
                p = 255;
                now = live[i];
                for(k=0; k<touchnum[now]; ++k)
                    if(map[touch[now][k]][t]<p)
                        p = map[touch[now][k]][t];
                sum += p;
            }
            if(minsum==-1 || sum<minsum) {
                minsum = sum;
                meet = t;
            }
        }
```

```
        out = fopen("walls.out", "w");
        fprintf(out, "%d\n%d\n", minsum, meet);
        fclose(out);
}
```