1998年資訊奧林匹亞試題及參考解答(三)

何榮桂* 林順喜* 李忠謀* 孫春在**
*國立臺灣師範大學 資訊教育系
**國立交通大學 資訊科學系

一、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，如圖 1。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。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。

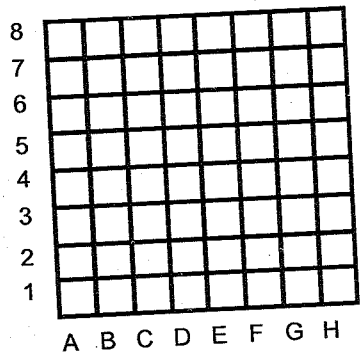


圖 1. 棋盤

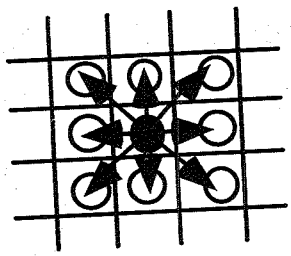


圖 2. 國王之可能棋步

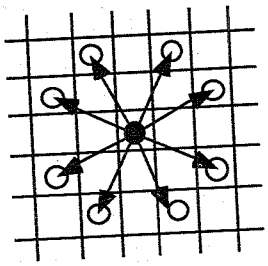


圖 3. 武士之可能棋步

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爲止，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，其他則爲武士的位置。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

二、參考解答

試作者：江盈宏(高雄市立高雄中學三年級學生，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 爲武士數
   mindist 是最後的答案,開始時設爲一個很大的值 */
struct Pos king,knight[63];
int mindistance[8][8][8][8];
int knights;
int mindist=32767;
/* 讀檔函式,並將 input 轉爲座標 */
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++;
        }
      }
    }
  }
  /* 此函式爲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
```

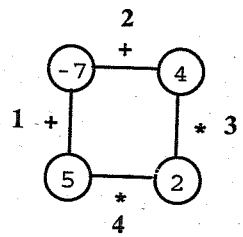
時的位置；org 爲國王與武士不重疊時所需步數；而 changed 則表示重疊後所需步數。因此 org-changed 即代表可以減少的步數，subsmax 爲這些步數中最大者,所以 nowdist-subsmax 即爲數對(i,j,k,l)所代表的最短距離，至於答案 mindist 則爲 nowdist 中最小者

```
*/
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;
  }
```

三、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

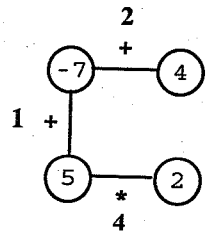
下第一步時，先移除任何一邊，之後的下法如下：

- 任取一邊，稱之爲E，並令其兩端的節點爲 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)則爲最後留下之節點標示之值。

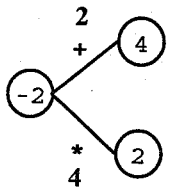
遊戲範例(Sample Game)：

考慮圖一之多邊形遊戲。玩家一開始移除標示爲3的邊。產生如圖二之結果。



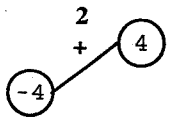
圖二： 移除標示為3的邊

接著，玩家選取標示爲1的邊，



圖三： 選取標示為1的邊

接著，標示爲4的邊，



圖四： 選取標示為4的邊

最後，標示爲2的邊。得分爲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 (首先是編號爲 1 和 2 的兩邊之間的節點，接著是編號爲 2 和 3 的兩邊之間的節點，以此類推，最後是編號爲 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。第二橫列以編號爲 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):

```
3 3
1 2
```

這是如圖一之多邊形所應得的輸出檔。

條件限制(Constraints)

$3 < N \le 50$

對任意的下法過程中，節點的標示之範圍爲[-32768, 32767]。

四、參考解答

試作者：鍾至衡（臺北市立建國中學三年級學生，IOI'98 銅牌獎）

/*首先看題目就知道，這題若用暴力法來做一定會耗時太久。因爲假設有 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。又如果 L2 == '+' MAX(1, 2) = MAX(1, 1) + MAX(2, 2); MIN(1, 2) = MIN(1, 1) + MIN(2, 2)；否則 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；爲什麼我們需要最小值呢？因爲可能有負號，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] = -20000000001;
        min[i][j] = 20000000001;
        }
    }
// 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;
```

（下轉66頁）