第九屆(1997 年)國際資訊奧林匹亞競賽試題分析(三)

何榮桂*、龔律全**、陳康本***
*國立臺灣師範大學 資訊教育系
**國立臺灣大學 資工系
***國立臺灣大學 電機系

1. 第二天第二題題目(Image:Character Recognition)

寫一個程式來做文字辨識

Details:

每一個字母樣本(ideal character image)是由 20 行，每行 20 個數字所組成，且每一個數字皆是'0'或'1'。圖 1a 是一個字母樣本的範例。

在 FONT.DAT 檔內有 27 個依以下順序排列的字母樣本：

○abcdefghijklmnopqrstuvwxyz(在此'○'代表空白)

在 IMAGE.DAT 檔有一或更多可能扭曲的字母影像，每一個影像的可能扭曲方式如下：

1.(最多)一行重複出現(duplicated line)且緊跟在原始行之後。

2.(最多)一行被刪除(missing line)。

3.有些'0'被改成了'1'。

4.有些'1'被改成了'0'。

沒有任何影像會同時有上述第一項及第二項同時發生，且在測試資料中，每張影像最多有 30%的 0-1 對變。

如有重複出現行時(duplicated line)，則重複出現行及原始行都可能有 0-1 對變，且對變盤也可能不一樣。

Task：

寫個程式來辨識一或更多的存在 IMAGE.DAT 檔的字母影像，而原始字母樣本則是存在 FONT.DAT 檔。辨識時，請選擇需要最少的 0-1 對變扭曲方式的字母樣本當作辨識出之字母。但如果你假設兩行之間有一行是另一行之重複行時，只有 0-1 對變較少那一行須要被計算進去。所有測試資料的字母影像一定可以被一個一個獨立的辨識出來，假若你的程式寫的好的話。而且每組測試資料一定有一組最佳解。正確的解應該會使用到 IMAGE.DAT

檔內所有資料行。

Input：

兩個輸入檔的第一行是一正整數 N (19<=N<=1200)

代表之後的行數如底下之範例：

N

(digit1)(digit2)(digit3).....(digit20)

(digit1)(digit2)(digit3).....(digit20)

.....

每一行都有 20 個數字，而這些 0 與 1 數字之間沒有空白。

FONT.DAT 含有字母樣本，且一定有 541 行。但每一組測試資料可能用不同的 FONT.DAT 檔

Output：

你的程式必須產生一個命名爲 IMAGE.OUT 的輸出檔。而輸出檔內只有一行辨識結果的字串(string)。輸出格式就是一行的 ASCII 文字字串。輸出檔字串內不應有任何間隔(字與字之間不可以有間隔)，假若你的鄉辨識不出某個字母，就在正確的位置輸出'?'。

注意：

上述輸出格式規定與競賽規則中所規定一"輸出需有間隔符號"不符。請遵守此新輸出格式規定。

評分方式

分數是所有正確辨識出的字母的百分比。

Sample files:

Incomplete sample showing the beginning of FONT.DAT(space and 'a')	Sample IMAGE.DAT showing an 'a' corrupted
FONT.DAT	IMAGE.DAT
00000000000000000000	19
00000000000000000000	00000000000000000000
00000000000000000000	00000000000000000000
00000000000000000000	00000000000000000000
00000000000000000000	00000011100000000000
00000000000000000000	00100111011011000000
00000000000000000000	00001111111001100000
00000000000000000000	00001110001100100000
00000000000000000000	00001100001100010000

00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000011100000000000 00000111111011000000 00001111111001100000 00001110001100100000 00001100001100010000 00001100000100010000 00000100000100010000 00000010000000110000 00000001000001110000 00001111111111110000 00001111111111110000 00001111111111000000 00001000000000000000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000	00001100000100010000 00000100000100010000 00000010000000110000 00001111011111110000 00001111111111110000 00001111111111000000 00001000010000000000 00000000000000000000 00000000000001000000 00000000000000000000 00000000000000000000
Figure 1a	Figure 1b

Sample Output

IMAGE.OUT	Explanation
a	Recognised the single character 'a'

2.參考解答

```
#include<fstream.h>
#include<stdlib.h>
#include<assert.h>
#include<string.h>
#define         NUM_CHAR    (27)
#define         N_DIGIT     (20)
#define         MAX_LINE    (1200)
#define         UNKNOWN     (-1)
#define         ERR_MAX(10000)
enum{CORRUPT, NORMAL, DUPLICATE};
//-------------------------------------------------
// global data 全域性資料
const char matchStr [ ] =
 " abcdefghijklmnopqrstuvwxyz";
// store font data 儲存字型資料
char
font[NUM_CHAR][N_DIGIT][N_DIGIT+1];

// store input data 儲存輸入資料
char data[MAX_LINE][N_DIGIT+1];
//store the number of lines 儲存輸入資料的
行數
int nLine;
//      store the minimum error until line No.n
// 儲存到 n 行爲止的最小錯誤值
int nError[MAX_LINE+1];
// store the previous lineno to obtain the
optimal value
// CORRUPT or NORMAL or DUPLICATE ,
will be used to backtrace the
// optimal answer
// 儲存到 n 行爲止的最小錯誤值是從那
一行走過來的，可能是
// 少一行、正常或多一行)，用來印出最
後的答案
int prev[MAX_LINE+1];
// store the optimal matched char that lead to
line n
// 儲存從上一行到這一行的最小錯誤值
所對應的字元辨識結果
char chr[MAX_LINE+1];
//-------------------------------------------------
// read font data into memory 此函式讀入字
型
void ReadFont(void)
{
      int i,j,k;
      // open font file   開字型檔以供輸入
      ifstream input("font.dat");
      // check if file has been opened   檢查
是否開檔成功
      assert(input);
      // ignore the first '540' 去掉第一行的
540
      input >> i;
      // read 27 characters 讀入 27 個字型資
料
      for(i=0;i<NUM_CHAR;i++) {
            for(j=0;j<N_DIGIT;j++) {
                  for(k=0;k<N_DIGIT;k++) {
                        input >> font[i][j][k];
```

```
            }
            }
        }
}
//-----------------------------------------------------
// read input data into memory 此函式讀入輸入資料
void ReadData(void)
{
    int i,j;

    // open input file 開輸入檔以供輸入
    ifstream input("image.dat");
    // check if file has been opened 檢查是否開檔成功
    assert(input);
    // read the number of lines 讀入輸入資料的行數
    input >> nLine;
    // read nLine line data 讀入 nLine 行的資料
    for(i=0;i<nLine;i++) {
        for(j=0;j<N_DIGIT;j++) {
            input >> data[i][j];

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

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

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

```
    // 或 lineno-21 行，這三者中再加上
之間的那個字型的 error 中最小的
    for(minErr=ERR_MAX,i=0;i<=2;i++) {
        if( lineno - i - 19 < 0 )
            continue ;
        err=ComputeError( lineno - i -
19) ;

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

```
// 主函式
int main(void)
{
   ReadFont();
   ReadData();
   Init();
   ComputeError(nLine);
   OutputAnswer();
   return 0;
}
```

3. Image 得分分布

分數	人數*	累積人數	百分比	累積百分比
0-10	64	64	28.96	28.96
11-20	11	75	4.98	33.94
21-30	13	88	5.88	39.82
31-40	16	104	7.24	47.06
41-50	12	116	5.43	52.49
51-60	19	135	8.60	61.09
61-70	10	145	4.52	65.61
71-80	5	150	2.26	67.87
81-90	29	179	13.12	81.00
91-100	42	221	19.00	100.00

*「人數」表示「人(次)數」

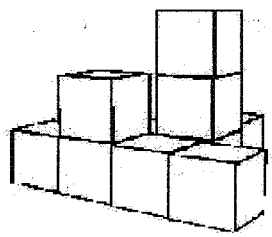
此題是所有題中較爲簡單者，雖然也有不少考生 (28.96%)得分在 10 分以下，但得分在 80 分以上也不在少數(32.12%)。

4.第二天第三題題目(Stack:Stacking Containers)

Neptune 貨運公司有一貨櫃儲存倉庫(depot)。倉庫接收貨櫃後，儲存一段時間後再移出。貨櫃只在每小時整點鐘時(on the hour)移入倉庫。並在倉庫儲存正整數個小時後移出。當貨櫃移入時，貨櫃就附有文件，標明它期望(expected)被移出的時間。貨櫃被真正移出的時間和期望移出的時間可能提前或延後，但最多不超過 5 個小時。在本題中，時間以遞增正整數表示，最大值不超過 150。倉庫的儲貨空間(storage space)之上有一吊車(crane)。可以把貨櫃移出或移入倉庫。也可以把倉庫內的貨櫃自一點移動到另一點，吊車在儲貨空間上移動，不影響儲貨空間。

Task：

寫一個好的吊車操作程式，使得使用吊車的次數爲最少。倉庫儲貨空間是矩形立方體，長爲 X，寬爲 Y，高爲 Z。長寬高會被告知。每個貨櫃是 1*1*1 的正立方實心



體(cube)。貨櫃可以堆在儲貨空間的地板上，或是疊在一堆貨櫃的正上面。吊車只可以移動一堆貨櫃最上面的一個。

一個貨櫃若由一點用吊車移動到任何一點，不管是移入，移出，或倉庫內移動，都算成一次操作移動(move)。吊車在貨櫃來到(arrival)和移出之間操作。假設吊車操作移動不花任何時間。當儲貨空間滿了後，你的程式必須拒絕接受移入任何貨櫃的要

求。當倉庫快滿時，你的程式也許有可能變得比較沒有效率或根本無法正常操作。你的程式在任何時候都可以拒絕接受移入新貨櫃的要求。

Input：

你的程式必須和一個模擬程式模組(simulation module)相連接(interact)，該模組會提出貨櫃移動要求(action)，及必需訊息(messages)。儲貨空間剛開始時完全沒有貨櫃。寫作時提供的測試程式會傳回一小組有意義，從頭到尾的測試資料。每個貨櫃以一唯一正整數編號。你的程式在任何時候都可以呼叫下列函數(function)：

int GetX();傳回儲貨空間的長度(整數值)；

int GetY();傳回儲貨空間的長度(整數值)；

int GetZ();傳回儲貨空間的長度(整數值)；

X,Y,Z 值不超過 32。

下列函數(function)傳回移動要求(action)~即要求某貨櫃移出或移入的必須資訊。貨櫃只在每小時正點鐘會到達而要求移入。貨櫃移出則絕對不在每小時的正點鐘上。在此，假設，每次一有新貨櫃要求移入，就代表剛好又經過一小時了。

int GetNextContainer()；

傳回一個正整數，代表下一個要求被移入或移出的貨櫃編號，若已無任何貨櫃要求移入或移出，則傳回 0，代表程式必須正常結束，不管當時倉庫內是否還有任何貨櫃。

int GetNextAction()；

傳回一個整數，1 代表移入新貨櫃。2 代表移出貨櫃。

int GetNextStorageTime()；

傳回以小時計算的貨櫃預期移出的時間(由程式開始執行算起)。這個值是爲了讓你的鄉預先規劃。真正貨櫃移出時間會有少許差異。但最多相差加減不超過 5 小時。當 GetNextAction 傳回 1 後，本函數才會傳回有意義的值。

上述幾個函數呼叫的次序並不重要。接連呼叫 GetNextContainer，GetNextAction 及 GetNextStorageTime 永遠傳回同一貨櫃的移動要求資訊，直到該貨櫃被你的程式拒絕，依要求移入或移出爲止。在此之後上述函數傳回下一個貨櫃移動要求的資訊。

Output：

當你的程式獲得貨櫃移動要求的資訊，計算後可以使用下列函數來操作吊車

int MoveContainer(int x1,int y1, int x2, int y2)；

把堆放在地板位置 x1,y1 那一堆貨櫃的最上面一個移到位置 x2,y2 那一堆貨櫃的最上面，傳回 1 若這個操作是合法的，傳回 0 若捆作不合法(亦即是不可能發生)。

void RefuseContainer()；

回報不願意接收新到貨櫃。

void StoreArrivingContainer(int x, int y)；

把新到貨櫃放到地板位置 x,y 那一堆貨櫃的最上面。

void RemoveContainer(int x, int y) ;

把放在地板位置 x,y 那一堆貨櫃的最上面一個移出倉庫。

若你的程式無法針對一個移動要求(action)執行上述任一函數時，則必須正常結束。不合法的操作會被忽略，對模擬狀態及你的得分都不產生任何影響。

你的程式不可以產生任何輸出檔案。你呼叫的函數自動會產生記錄檔，並用以評分。

操作次序：

你的程式應依得到的每一次貨櫃移動要求(action)決定如何操作吊車移入，移出(並有可能內部移動)貨櫃。也可以決定拒絕接收新到貨櫃。

Library：

你的程式必須和一個叫 StackLib 的程式庫集相接(link)，標準 C 及 C++程式庫集已經包括那些多加的函數，只要程式加上適當的 header file 即可。範例程式叫 TESTSTK.CPP 或 TESTSTK.C。

評分方式

你的程式會被測試數個回合，每一回合會有一組最佳解。依此用下列方式評分：先計算你程式操作吊車的總次數；對每一拒絕接收移入的貨櫃，罰多加五次操作(move)；你的程式若提早正常結束則對每一未移入及移出的貨櫃，罰多加五次操作；分數計算以和最佳解相差的百分比決定，若捆作次數是最佳解的兩倍以上，則得 0 分。程式操作次數等於最佳解，得 100 分，在此和兩倍最佳解之間則依相差比例得皆，超過得 0 分。

5. 參考解答

```
#include<limits.h>
#include<iostream.h>
#include<stdlib.h>
////////////////////////////////////////////////////////
// stack.cpp wrote Dec 4 1997,Cape Town, SA
//
//              by Lu-chuan Kung                  //
////////////////////////////////////////////////////////
#define MAX_N 33
//#define MYDEBUG
// 以下爲大會所提供的程式庫
extern "C" int GetX();//此函式傳回倉庫的長度
extern "C" int GetY(); //此函式傳回倉庫的寬度
extern "C" int GetZ(); //此函式傳回倉庫的高度
extern "C" int GetNextContainer();//傳回下一個貨櫃的編號
extern "C" int GetNextAction();   //取得下一個要求的動作
extern "C" int GetNextStorageTime();//取得下一個貨櫃的取出時間
extern "C" void RefuseContainer();      //拒絕放進貨櫃
extern "C" void StoreArrivingContainer(int x, int y); // 把貨櫃放在(x,y)
extern "C" void RemoveContainer(int x, int y);
```

```
// 取出(x,y)的貨櫃
// 將 (x1,y1) 的貨櫃移至 (x2,y2)
extern "C" int MoveContainer(int x1, int y1,
int x2, int y2);
int nx,ny,nz;
int hour;
int huge stack[MAX_N][MAX_N][MAX_N];
int huge
expTime[MAX_N][MAX_N][MAX_N];
int top[MAX_N][MAX_N];
void Init(void)
{
    nx = GetX();
    ny = GetY();
    nz = GetZ();
}
//此函式找到倉庫中最低的(x,y)，並傳回其
高度
int FindLowest(int &x,int &y,int &minexp)
{
  int min_z = INT_MAX,i,j;
  for(i=0;i<nx;i++) {
    for(j=0;j<ny;j++) {
      if( top[i][j] < min_z ) {
      min_z = top[i][j];
        x = i;
        y = j;
      if( min_z > 0 )
      minexp = expTime[i][j][ min_z-1 ];
          }
        }
    }
    return min_z;
}
//傳回除了(ex,ey)之外的座標，最低的座標
int FindLowestElse(int &x,int &y,int ex,int
ey)
{
    int i,j;
    int min_z = INT_MAX;
    for(i=0;i<nx;i++) {
      for(j=0;j<ny;j++) {
        if( i==ex && j == ey )
          continue;
        if( top[i][j] < min_z ) {
          min_z = top[i][j];
          x = i;
          y = j;
        }
      }
    }
    return min_z;
}
// 放入新的貨櫃
void GetNew(int conno,int exptime)
{
    int z,x,y,minexp;
    int cx,cy,cz;
    int i,j;
    int moveBack = 1;
    // 先找到最低的(x,y)座標
    z = FindLowest(x,y,minexp);
```

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

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

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

```
            stack[x][y][i] = stack[x][y][i+1];
        expTime[x][y][i]=expTime[x][y][i+1];
        }
        top[x][y] --;
}
// 主函式
int main(void)
{
        int no,action,time;
        Init();
        hour = 0;
        while(1) {
            no = GetNextContainer() ;
            if( no == 0 ) // 貨櫃程式結束
                break;
action=GetNextAction(); //取得下一個動作
            switch( action ) {
            case 1: // 放入一個新的貨櫃
            time=GetNextStorageTime();
                        GetNew(no,time);
                        hour++;
                        break;
            case 2: // 移出一個指定的貨櫃
                        Remove(no);
                        break;
            }
        }
        return 0;
}
```

6. Stack 得分分布

分數	人數*	累積人數	百分比	累積百分比
0-10	111	111	50.23	50.23
11-20	9	120	4.07	54.30
21-30	2	122	0.90	55.20
31-40	44	166	19.91	75.11
41-50	11	177	4.98	80.09
51-60	21	198	9.50	89.59
61-70	6	204	2.71	92.31
71-80	12	216	5.43	97.74
91-100	5	221	2.26	100.00

*「人數」表示「人(次)數」

從考生的得分分布來看，此題也相當難，約有一半(50.23%)得分在10分以下。分數愈高，考生的人數有遞減現象。

7.結語

以上是1997年IOI之中譯試題、參考解答及全體考生之得分分配，提供對參加IOI選拔有興趣的同學參考。雖然參賽者每年都增加，但從每題之得分分布來看，考生間及隊與隊間(請參閱本刊207期第九屆(1997年)國際資奧林匹亞(IOI)競賽報告)的實力頗爲懸殊。整體而言，我隊實力在前10名，個別表現也很好，以我們學生的潛力，應該可以更好。