

全國高級中等學校 112 學年度工業類科學生技藝競賽 電腦軟體設計

壹、試卷說明：

1. 請將寫好之程式原始檔依題號命名資料夾存檔，第一題取**姓名_Q1(例如李小明_Q1)**，第二題取**姓名_Q2**，依序命名存檔，

並存於 C 碟之資料夾”**姓名_Contest**”中。

2. 競賽時間 4 小時。

3 將程式及編譯成執行檔儲存在 C 碟之資料夾姓名_Contest。

貳、評分說明:本試卷共六題，每題配分不一。

1. 每題評分只有對與錯兩種，對則給滿分，錯則不給分(即以零分計算)。

2. 每解答完一題上傳(程式及執行檔)，評審人員將針對該題進行測試，若解題正確則回應正確，若解題錯誤則扣該題一分至該題零分為止，答錯之題目可繼續作答。

3. 人機介面的文數字一律使使用粗體及大小為 12。

試題 1：設計一程式能找出四位數或三位數之黑洞數 (16 分)

說明：黑洞數是指於四位數和三位數中，只要數字不完全相同，將數字由大到小的排列減去由小到大的排列，經有限次數操作後，總會縮斂得到某一個。假設一開始給定的一個四位數或三位數數字為 x_1 ， $x_2 = \text{fmax}(x_1) - \text{fmin}(x_1)$ ， $x_3 = \text{fmax}(x_2) - \text{fmin}(x_2)$ ，...， $x_n = \text{fmax}(x_n) - \text{fmin}(x_n)$ ， x_n 就是黑洞數， $\text{fmax}(x_1)$ 是將 x_1 數字由大到小的排列數， $\text{fmin}(x_1)$ 將 x_1 數字由小到大的排列數。

注意:程式可重複執行。輸入數字至少有一位數不同，0 開頭數字位數會少一位，例如輸入 0649 視為是三位數 649。

例 1：輸入 $x_1=9990$ ，則 $x_2 = \text{fmax}(x_1) - \text{fmin}(x_1)$ ， $x_2=9990-999=8991$ ， $x_3=9981-1899=8082$ ， $x_4=8820-288=8532$ ， $x_4=8532-2358=6174$ ， $x_5=7641-1467=6174$ ， $x_4=6174$ 就是四位數之黑洞數。

例 2：輸入 $x_1=639$ ，則 $x_2 = \text{fmax}(x_1) - \text{fmin}(x_1)$ ， $x_2=963-369=594$ ， $x_3=954-459=495$ ， $x_4=954-459=495$ ， $x_3=495$ 就是三位數之黑洞數。

輸入說明：

讓使用者輸入四位數或三位數。

輸出說明：

輸出黑洞數。

程式執行範例：

1. 輸入 $x_1=9891$	2. 輸入 $x_1=639$
輸出: $x_2=9981-1899=8082$ ， $x_3=8820-0288=8532$ ， $x_4=8532-2358=6174$ ，黑洞數=6174	輸出: $x_2=963-369=594$ ， $x_3=954-459=495$ ，黑洞數=495

題目二：最小編輯距離(17 分)

說明：讀入兩個英文字(word)，印出由第一個英文字轉換成第二個英文字所需要的最小編輯距離(edit distance)。編輯距離包括三種動作的成本：插入(insert)字母、刪除(delete)字母、及取代(substitute)字母。例如輸入兩個英文字"idea"及"deal"，如果刪除"idea"中的第一個字母"i"，並在最後面插入字母"l"，就可以將"idea"轉換成"deal"。再舉一個例子，如果輸入的兩個英文字是"but"及"bait"，我們先將第一個字中的字母"u"取代成字母"a"，然後再在字母"a"之後插入字母"i"，就可

以將"but"轉換成"bait"。當插入及刪除動作的成本都是 1，取代動作的成本是 2，則第一個例子將"idea"轉換成"deal"的編輯距離就是 2（包括一個刪除及一個插入）；而第二個例子將"but"轉換成"bait"的編輯距離就是 3（包括一個取代及一個插入）。由於兩個英文字間可能有多種轉換方法，且每一種轉換方法都有它的編輯距離，而我們要找出的就是兩個英文字間的最小編輯距離。

這個問題可以用 dynamic programming 的方法來解。我們定義 $D(i, j)$ 為由第一個英文字 X 的前 i 個字母轉換到第二個英文字 Y 的前 j 個字母的最小編輯距離；若 X 有 n 個字母且 Y 有 m 個字母，則由 X 轉換成 Y 的最小編輯距離就是 $D(n, m)$ 。 $D(i, j)$ 可以用下列方式計算：

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & // \text{ del } X(i) \\ D(i, j-1) + 1 & // \text{ ins } Y(j) \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) & // \text{ sub} \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

以下是上面第一個例子的計算結果，其中最右下角的數值 2 就是這個例子的最小編輯距離：

X \ Y		d	e	a	l
	0	1	2	3	4
i	1	2	3	4	5
d	2	1	2	3	4
e	3	2	1	2	3
a	4	3	2	1	2

另外，在 Visual Basic 中取得字串變數 X 的字母的方法是 $X.Chars(i)$ ，而在 C++ 及 C# 中取得字串變數 X 的字母的方法是 $X[i]$ 。

輸入說明：第一列輸入第一個英文字，第二列輸入第二個英文字，每個英文字不超過 20 個字母，不考慮輸入錯誤的情形。

輸出說明：印出由第一個英文字轉換成第二個英文字所需要的最小編輯距離。

執行範例：

輸入原來的英文單字: idea 輸入改變後的英文單字: deal 編輯距離: 2	輸入原來的英文單字: but 輸入改變後的英文單字: bait 編輯距離: 3
輸入原來的英文單字: intention 輸入改變後的英文單字: execution 編輯距離: 8	輸入原來的英文單字: apple 輸入改變後的英文單字: apple 編輯距離: 0

題目三：簡單的 3 乘 3 井字棋程式(16 分)

題目說明： 玩家(你)與電腦對弈，你打空心圈（全形 O）、電腦打叉（全形 X）、空白處用口字（中文口）呈現。你與電腦輪流在 3 乘 3 的格上打自己的符號，最先以橫、直、斜連成一線者為勝。如果雙方都下得正確無誤，最後棋盤將會被填滿而成平手。程式一開始用 9 個口字排成 3 乘 3 的井字棋，畫面如圖 1 所示，9 個口的位置對應數字鍵盤的 1~9 鍵如圖 2 所示。

你為 O、電腦為 X
位置對應數字鍵盤

圖 1

□□□
□□□
□□□
請輸入1~9(按0鍵結束):



圖 2

程式執行狀態：程式不用處理輸入錯誤的問題(如輸入英文字母)。

程式開始均由玩家(你)為起手，開始畫面如圖 1 所示。

- 結束程式：按 0 鍵結束程式執行。

- 你 O 方獲勝：**<注意：因電腦是隨機下，以下說明會與你的畫面不同>**

由玩家(你)先下在 4 的位置、電腦隨機下在 6 的位置，如圖 3 上。

接著玩家(你)下在 1 的位置、電腦隨機下在 3 的位置，如圖 3 中。

接著玩家(你)下在 7 的位置，程式判斷“你 O 方獲勝 再來一盤”，

接著程式回到開始的畫面如圖 3 下。**因你 O 方獲勝，下一輪由 O 方先下。**

- 電腦 X 方獲勝：

由玩家(你)下在 8 的位置、電腦隨機下在 1 的位置，如圖 4 上。

接著玩家(你)下在 9 的位置、電腦隨機下在 5 的位置，如圖 4 中；

接著玩家(你)下在 3 的位置、電腦隨機下在 2 的位置。程式判斷“電腦 X 方獲勝 再來一盤”。**因電腦 X 方獲勝，下一輪由 X 方先下，如圖 4 下。**

- 下的位置已有棋子：

由玩家(你)先下在 9 的位置、電腦隨機下在 4 的位置，如圖 5 左。

接著玩家(你)下在 4 的位置，程式判斷“下的位置已有棋子”，如圖 5 右。

請輸入1~9(按0鍵結束): 9
□□O
X□□
□□□

圖 5 左

請輸入1~9(按0鍵結束): 4
下的位置已有棋子
□□O
X□□
□□□
請輸入1~9(按0鍵結束):

圖 5 右

- 雙方平手：

由玩家(你)先下再電腦隨機下，依序為 4、3、2、8、9、1、7、6 的位置，

如圖 6 上。最後玩家(你)下在 5 的位置，程式判斷“雙方平手 再來一盤”

接著程式回到開始的畫面如圖 6 下。

繳交檢查時，請同時提供 exe 執行檔。

□□□
□□□
□□□
請輸入1~9(按0鍵結束): 4
□□□

圖 3 上

請輸入1~9(按0鍵結束): 1
□□□
O□X
O□X

圖 3 中

請輸入1~9(按0鍵結束): 7
O□□
O□X
O□X

圖 3 下

你 O 方獲勝 再來一盤

□□□
□□□
□□□
請輸入1~9(按0鍵結束):

□□□
□□□
□□□
請輸入1~9(按0鍵結束): 1
□X□
□□□
O□□

圖 4 上

請輸入1~9(按0鍵結束): 9
□XO
□X□
O□□

圖 4 中

請輸入1~9(按0鍵結束): 3
□XO
□X□
OXO
電腦 X 方獲勝 再來一盤

圖 4 下

□□□
X□□
□□□
請輸入1~9(按0鍵結束):

你為 O、電腦為 X
位置對應數字鍵盤

□□□
□□□
□□□
請輸入1~9(按0鍵結束): 4
□□□
O□□
□□X

圖 6 上

請輸入1~9(按0鍵結束): 2
□X□
O□□
O□X

請輸入1~9(按0鍵結束): 9
□XO
O□□
XOX

請輸入1~9(按0鍵結束): 7
OXO
O□X
XOX

請輸入1~9(按0鍵結束): 5
OXO
OXO
XOX
雙方平手 再來一盤

圖 6 下

□□□
□□□
□□□
請輸入1~9(按0鍵結束):

題目四：多台無人機操控系統(17 分)

為了操控無人機系統，請寫一程式來完成飛行操控功能：給一無人機飛行空間的長和寬(在此不考慮高度)。再給定每台無人機的起始位置，以及一連串的命令，最後求出無人機的飛行最後位置。無人機的位置包括座標(x, y)、以及面向的方向：北 (N)、南 (S)、東 (E)、西 (W)。無人機收到的命令，是由字母 'L'(無人機在原地左轉 90 度)、'R'(無人機在原地右轉 90 度)、'F'(無人機往面向的方向向前走一步，且不改變其面向的方向)。從座標(x, y)走至(x, y+1)的方向定義為北方，其他方向依此類推。無人機飛行空間是有邊界的(在此不考慮高度)，一旦無人機飛出邊界，就相當於被敵人摧毀，不過被摧毀的無人機會留下「記號」，提醒以後的無人機，避免他們再到同一個地方，被敵人摧毀，所以，對於之後的無人機，當它飛到有記號的地方時，此無人機就會忽略，會讓它被摧毀的命令。

輸入說明：

可以讓使用者輸入無人機飛行操控命令檔案(假設 input.txt)，此命令檔案內容：

1. 第一列的正整數代表共有幾台無人機。
2. 第二列有 2 個正整數，代表無人機飛行空間的右上角頂點座標，其中假設無人機飛行空間的左下角座標為(0, 0)。
3. 接下來的每 2 列為 1 組無人機測試資料：
 - 3.1 第 1 列為位置，包括 x 座標、y 座標以及所面對的方向。
 - 3.2 第 2 列為命令，由 'L'、'R'、'F' 所組成的字串。
4. 各無人機是依序飛行的，即一台無人機要飛行完成所有命令，下一台無人機才會開始飛行。
5. 所有無人機的起始位置皆在無人機飛行空間的長和寬範圍內，任何座標的最大值不可以超過 50，每個命令的長度皆不超過 100 個字元。

輸出說明：

1. 對每一組無人機測試資料，輸出一列該無人機最後所在的 x 座標、y 座標以及所面對的方向。
2. 如果一台無人機於飛行空間的長和寬座標範圍外被摧毀，則必須輸出它在被摧毀前的 x 座標、y 座標以及所面對的方向，並在最後面加一個字: Destroyed。
3. 上述資料除了顯示於螢幕外，也要輸出於一檔案(output.txt)中。

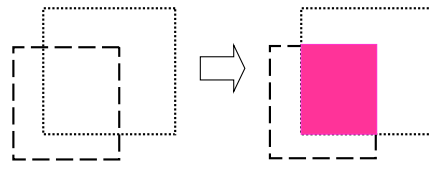
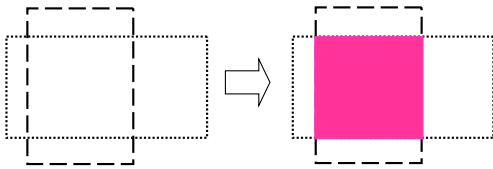
輸入範例 (input.txt)	輸出範例 (output.txt)
2 5 3 1 1 E RFRFRFRF 3 2 N FRRFLLFFRRFLL	1 1 E 3 3 N Destroyed

程式的功能，要自己寫，不可以使用現成套件或程式庫，若妳(你)的程式都完成上述功能，才可以要求檢查。

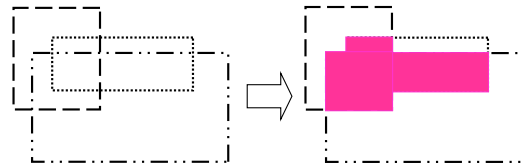
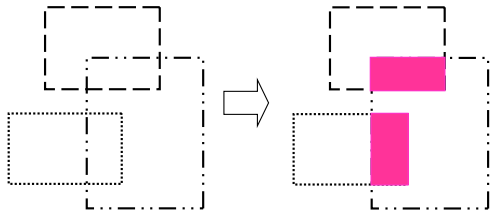
試題五：找出任意區塊間所有重疊區域之系統 (A system for finding all overlapping regions between arbitrary blocks) (17 分)

說明：(一) 幾何圖形學科有一個基本應用，從任意幾個區塊 (blocks) 找出相互間之所有重疊區域 (overlapping regions)，如下圖所示為 2、3 及 4 個區塊等各有兩個實例，每個實例的左方為任意放置區塊數，箭頭右方為找出對應區塊相互間的所有重疊區域且塗滿顏色 (如紅色)。

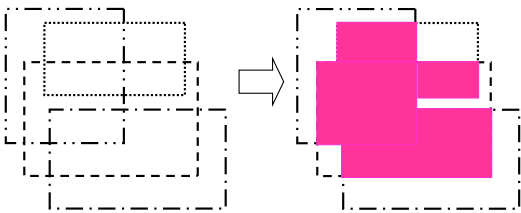
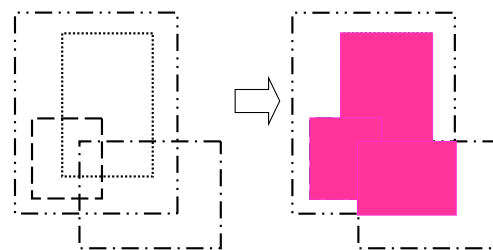
(1) 任意 2 個區塊間的所有重疊區域



(2) 任意 3 個區塊間的所有重疊區域



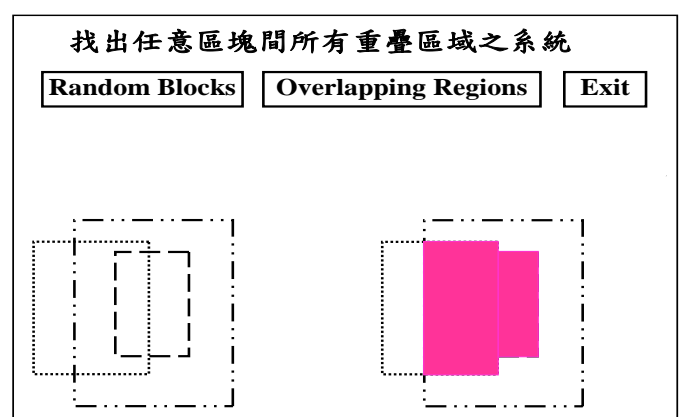
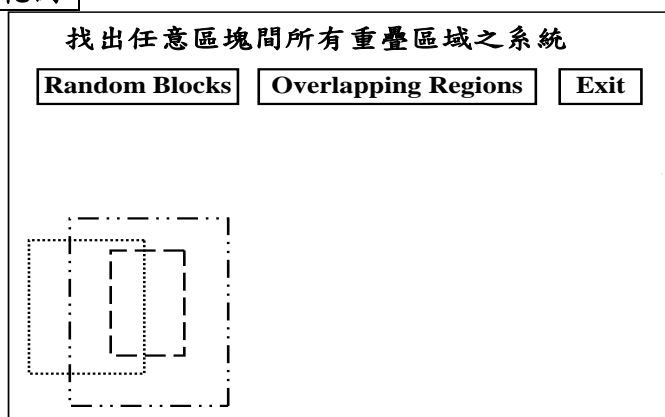
(3) 任意 4 個區塊間的所有重疊區域



(二) 找出任意放置 2~4 個區塊間的所有重疊區域之方法；可依自己的思考方法自行設計，或者可參考此演算法這四個步驟：(1) 假設每一個區塊之左下角與右上角座標分別為 $(x1,y1)$ 與 $(x2,y2)$ ；(2) 首先檢視任意兩個區塊 $b1$ 與 $b2$ 是否屬於非重疊的四種情況： $b1$ 在 $b2$ 上方、 $b1$ 在 $b2$ 下方、 $b1$ 在 $b2$ 左方或 $b1$ 在 $b2$ 右方；如不屬於，則可確認這兩個區塊是互相重疊的，再找出其重疊區域之左下角座標為區塊 $b1$ 與 $b2$ 各自 $x1$ 與 $y1$ 之最大值及右上角座標為區塊 $b1$ 與 $b2$ 各自 $x2$ 與 $y2$ 之最小值，並記錄此重疊區域；(3) 依照第(2)步驟，逐一找出每一對區塊間對應的重疊區域而紀錄之；(4) 將這些紀錄的重疊區域綜合聯集起來即為所有重疊區域 (overlapping regions)。

(三) 系統設計：(1) 請參考說明(一)與(二)的陳述，設計一個「找出任意區塊間所有重疊區域之系統」；(2) 當系統啟動或每當滑鼠點選一下 **Random Blocks** 鍵，能隨機產生 2~4 個區塊，每一個區塊左下角座標 $(x1,y1)$ 範圍各為 20~80，長度與寬度範圍各為 40~200，且須能立即清除繪圖畫布及使用不同顏色的虛線型態繪出這些區塊，如範例下圖左方所示；(3) 每當滑鼠點選一下 **Overlapping Regions** 鍵，能立即找出這些區塊間的所有重疊區域，並以紅色實心塗滿繪出，且能與左邊區塊圖相對照，如範例下圖右方所示；(4) 上述可重複操作直至滑鼠點選一下 **Exit** 鍵而離開此系統。

範例



題目六：大樓電梯行程演算設計(17 分)

一、題目描述

現在都會區高樓層(20 層上下)比比皆是，通常配置二部電梯以上。電梯內部所按目標樓層使命必達。電梯外部按壓往上或往下，有些設計使程式控制電梯各自分離操作，有些設計二部電梯連動操作。後者的好處是避免二部電梯都按以致有一部空轉，同時，計算出花費較短時間的電梯來服務。本題目採用後者。

二、操作規則

1. 二部電梯內部的按壓為電梯各自處理。

(1) 電梯往上(下)時，電梯移動往該按壓之最高(低)樓層，達到內部按壓之最高(低)樓層後轉為停止(Z)或反轉，並清除該部電梯之所有內部按壓。

(2) 途經相同方向的內部按壓之樓層電梯停留。

(3) 當電梯為停止(Z)狀態，按高(低)於此樓層則電梯往上(下)。

2. 電梯外部的按壓為二部電梯連動。當電梯外部按壓時，程式計算二部電梯到達時間，具最短者指派之。

(1) 程式先計算電梯行進時間，當碰到與 A 電梯同方向之外部按壓，同時比較 B 電梯，達到此按壓之時間較短者獲指派，依此進行直至無同方向之外部按壓。若無同方向之外部按壓，也無內部按壓的話，而有反方向外部按壓，則電梯行進方向反轉。

(2) 當電梯行進方向反轉，計算方式同步驟(1)，直到無同方向之外部按壓，也無內部按壓的話，電梯行進方向再反轉。



3. 假設電梯移動一樓層須 3 秒，內部或外部按壓之樓層停留 6 秒。



4. 依據電梯位置、方向、內外部按壓現況，當最近一次外部按壓時，程式計算二部電梯到此外部按壓樓層的時間以便決定啟動時間較短的電梯來服務。



5. *限制：程式模擬電梯向上(下)時，最高(低)之內外按壓不使用外部按壓向上(下)。

為簡化程式設計，以本題之二範例為主，變化只移動外部按壓位置，或增減內部按壓數目。

三、符號表示

、：表電梯外部連動按壓向上

、：表電梯外部連動按壓向下

、：表最近一次電梯外部連動按壓向上

、：表最近一次電梯外部連動按壓向下



電梯方向：U 表上升，D 表下降，Z 表停止



AI 及 BI：表 A 電梯及 B 電梯之內部按壓


OU 及 OD：表 A 及 B 電梯連動之外部按壓往上及往下

TAU 及 TAD：表電梯 A 上行及下行所需時間

TBU 及 TBD：表電梯 B 上行及下行所需時間

、：表電梯反轉向下(上)

、：表電梯移動向下(上)

：表電梯內部按壓

四、操作步驟

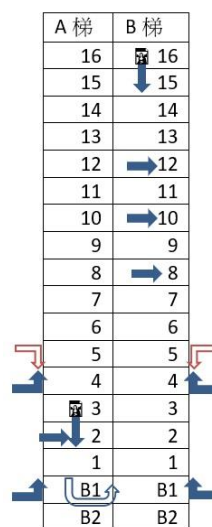
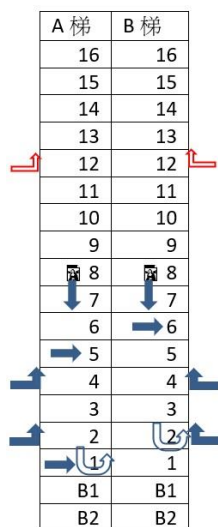
例一：

(一) 電梯位置與方向

電梯	方向	位置
A	D	8
B	D	8

最後外部按壓：

方向	位置
U	12



(二) 電梯內外部按壓現況(在 12 樓外部

(A)

(B)

按壓往上)

圖一 電梯位置、方向、內外部按壓狀況

樓層 電梯	B2	B1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
AI	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
BI	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
OU	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0
OD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TAD	0	0	27	24	21	18	9	6	3	0	0	0	0	0	0	0	0	0
TAU	0	0	33	36	39	42	45	48	51	54	57	60	63	66	0	0	0	0
TBD	0	0	0	24	21	18	15	6	3	0	0	0	0	0	0	0	0	0
TBU	0	0	0	30	33	36	45	48	51	54	57	60	63	66	0	0	0	0

由圖一(A)，電梯 A 之 1 樓及 5 樓有內部按壓(AI)，電梯 B 之 6 樓有內部按壓(BI)，外部按壓向上在 2 樓及 4 樓，最後外部按壓向上在 12 樓。執行時，當電梯 B 下行至 6 樓及電梯 A 下行至 5 樓，因各有內部按壓，各停留 6 秒，移動到下一層樓需 3 秒，故需 9 秒，即電梯 B 由 6 樓到 5 樓，電梯 A 由 5 樓到 4 樓時各加 9 秒(=6+3)。電梯 B 在 5 樓以下已無內部按壓，但其下仍有外部按壓，故要與電梯 A 比較到達時間，而電梯 A 在 1 樓仍有內部按壓，會下行至 1 樓，故電梯 B 獲指派 2 樓之外部按壓往上，餘此類推。經計算 2、4 及 12 樓外部按壓往上之 TAU 比 TBU 依序為(36:30)、(42:36)及(66:66)，故 2、4 指派給比較早到之電梯 B。

例二：以圖一(B)為例；

(一) 電梯位置與方向

電梯	方向	位置
A	D	3
B	D	16

最後外部按壓：

方向	位置
D	5

(二) 電梯內外部按壓現況(在 5 樓外部按壓往下)

樓層 電梯	B2	B1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
AI	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BI	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0

OU	0	$\frac{1}{A}$	0	0	0	$\frac{1}{A}$	0	0	0	0	0	0	0	0	0	0	0
OD	0	0	0	0	0	0	$\frac{1}{A}$	0	0	0	0	0	0	0	0	0	0
TAD	0	15	12	3	0	0	0	0	0	0	0	0	0	0	0	0	0
TAU	0	21	24	27	30	33	42	0	0	0	0	0	0	0	0	0	0
TBD	0	0	0	0	0	0	51	48	45	36	33	24	21	12	9	6	3
TBU	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

由圖一(B)，電梯 A 之 2 樓有內部按壓(AI)，電梯 B 之 8、10 及 12 樓有內部按壓(BI)，外部按壓向上在 B1 樓及 4 樓，最後外部按壓向下在 5 樓。執行時，電梯 B 於 8、10 及 12 樓有內部按壓，故下行至 7 樓已需 45 秒，電梯 A 於 B1 樓只需 15 秒，故 B1 樓外部按壓往上指派電梯 A。電梯 A 於 4 樓只需 33 秒，故 4 樓外部按壓往上指派電梯 A。電梯 A 於 5 樓只需 42 秒，因其上已無外部按壓往上，故 5 樓外部按壓往下指派電梯 A。當電梯 B 到 7 樓時 45 秒且已無按壓信號，故即可轉為停止。

程式執行

範例

輸入格式

```

Enter 1st filename:fl.txt
A D 3
B D 16
D 5
Enter 2nd filename:f2.txt
AI 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
BI 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
OU 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
OD 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
TAD 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TAU 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TBD 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TBU 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

輸出格式

```

列印結果.
樓層 B2 B1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
AI 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BI 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
OU 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OD 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
TAD 0 15 12 3 0 0 0 0 0 0 0 0 0 0 0 0 0
TAU 0 21 24 27 30 33 42 0 0 0 0 0 0 0 0 0 0
TBD 0 0 0 0 0 0 0 0 0 36 33 24 21 12 9 6 3
TBU 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```