

序

國立臺灣師範大學資訊系所教授們多年來推動 APCS(Advanced Placement Computer Science) 不遺餘力，得到很多大學的響應，終使此程式設計先修檢測蔚為風潮。邇來多位學子因通過此認證而進入理想大學，有提供 APCS 專屬名額的資訊科系也逐年增加中。

本書所有題目取材自網站—APCS 大學程式設計先修檢測 (<https://apcs.csie.ntnu.edu.tw/>)，每年的考試訊息均在此網站揭露。

希望讀者能領悟出說故事的能力，全盤的掌握並循序漸進的講述題目，心裡有一個比較清晰的側重點，對於資料的演進過程做好合理的安排。先能口頭語言直白表述，聽者覺得淺顯易懂，再來研究將方法轉換為程式。

追求程式的極至效率，來體現解題思路，又得兼顧通俗親民，在時間複雜度與空間複雜度或有讓步。若先進有建設性意見，歡迎 email 指教 (tbrain@mail.ntust.edu.tw)。

全華圖書王詩蕙小姐、陳奕君小姐與林宜君小姐協助此書付梓，借此片紙，聊表謝忱。

劉士華

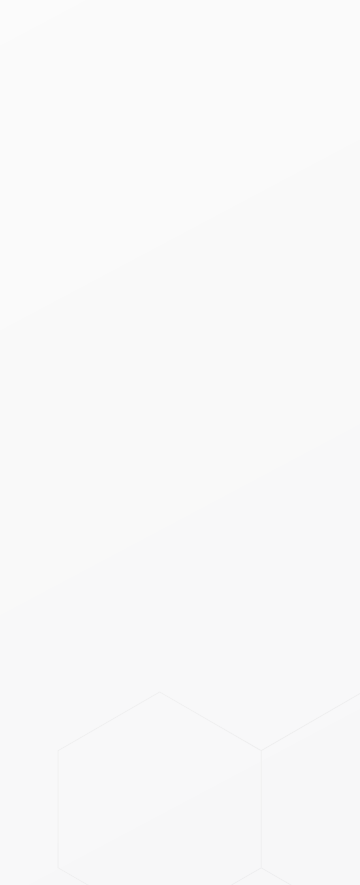


No. **1**

成績指標



2016 年 03 月 05 日 大學程式設計先修檢測
實作題

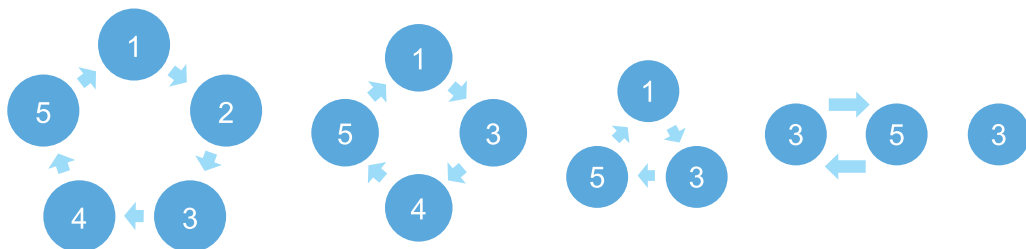




7.1 問題描述

「定時 K 彈」是一個團康遊戲，N 個人圍成一個圈，由 1 號依序到 N 號，從 1 號開始依序傳遞一枚玩具炸彈，炸彈每次到第 M 個人就會爆炸，此人即淘汰，被淘汰的人要離開圓圈，然後炸彈再從該淘汰者的下一個開始傳遞。遊戲之所以稱 K 彈是因為這枚炸彈只會爆炸 K 次，在第 K 次爆炸後，遊戲即停止，而此時在第 K 個淘汰者的下一位遊戲者被稱為幸運者，通常就會被要求表演節目。例如 N=5，M=2，如果 K=2，炸彈會爆炸兩次，被爆炸淘汰的順序依序是 2 與 4（參見下圖），這時 5 號就是幸運者。如果 K=3，剛才的遊戲會繼續，第三個淘汰的是 1 號，所以幸運者是 3 號。如果 K=4，下一輪淘汰 5 號，所以 3 號是幸運者。

此題輸入 N、M 與 K，請你計算出誰是幸運者。



◆ 圖 7.1

輸入格式

輸入只有一行包含三個正整數，依序為 N、M 與 K，兩數中間有一個空格分開。其中 $1 \leq K < N$ 。

輸出格式

請輸出幸運者的號碼，結尾有換行符號。

範例一：輸入

5 2 4

範例一：正確輸出

3

說明

被淘汰的順序是 2、4、1、5，此時 5 的下一位是 3，也是最後剩下的，所以幸運者是 3。

範例二：輸入

8 3 6

範例二：正確輸出

4

說明

被淘汰的順序是 3、6、1、5、2、8，此時 8 的下一位是 4，所以幸運者是 4。

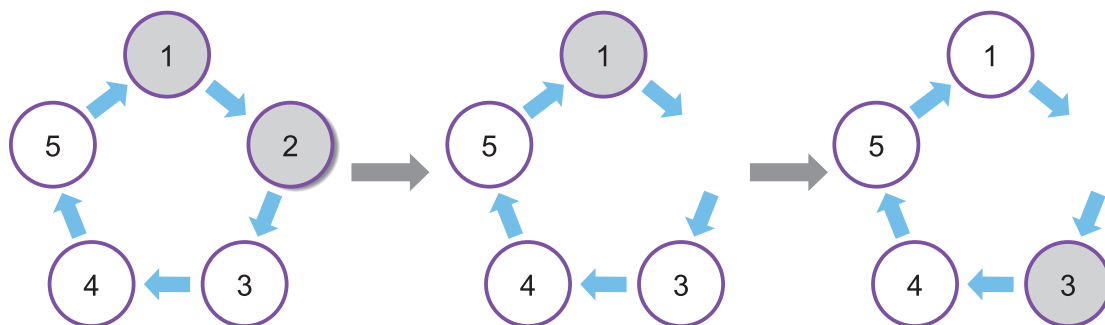
7.2 範例說明

以題目圖形為範例，初始值 N 為 5 個節點， $\{1, 2, 3, 4, 5\}$ 。每次歷程得經過 M 個節點， $M = 2$ ，所以每次的節點數字增量是 $1(M - 1)$ 。要爆炸 K 次， $K = 4$ 。以 S 代表起始點， E 代表終點，亦即爆炸點。

1. $S = 1, E = 2, K = 1$

起始點節點 1，走到節點 2，共經歷了 2 個節點。

節點 2 爆炸，下一個歷程的起始點移至下一個節點，也就是節點 3，如圖 7.2。

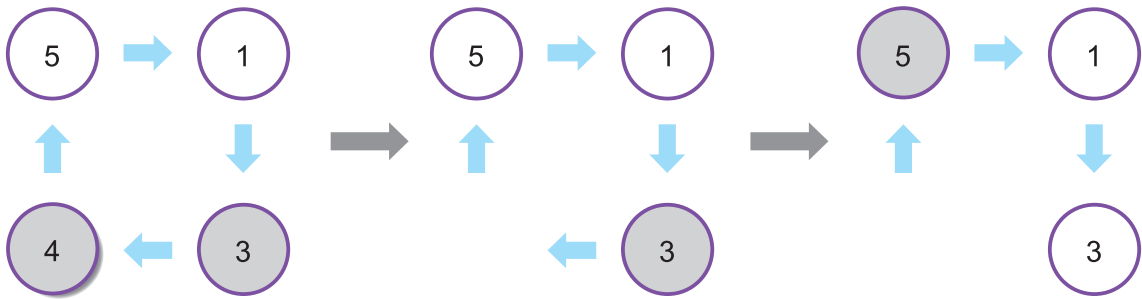


◆ 圖 7.2 $S = 1, E = 2, K = 1$

2. $S = 3, E = 4, K = 2$

起始點節點 3，走到節點 4，共經歷了 2 個節點。

節點 4 爆炸，下一個歷程的起始點移至下一個節點，也就是節點 5，如圖 7.3。

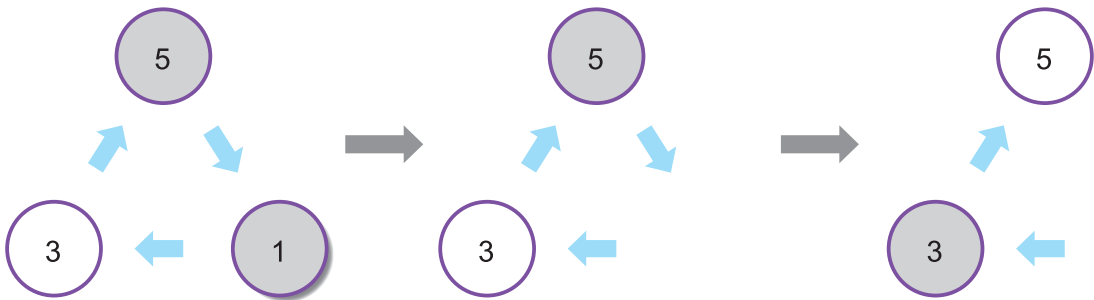


◆ 圖 7.3 $S = 3, E = 4, K = 2$

3. $S = 5, E = 1, K = 3$

起始點節點 5，走到節點 1，共經歷了 2 個節點。

節點 1 爆炸，下一個歷程的起始點移至下一個節點，也就是節點 3，如圖 7.4。



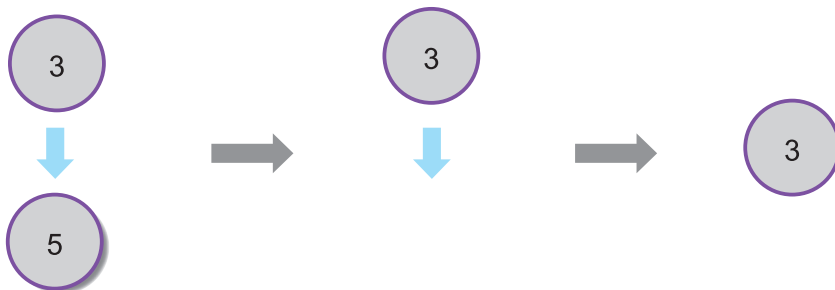
◆ 圖 7.4 $S = 5, E = 1, K = 3$

4. $S = 3, E = 5, K = 4$

起始點節點 3，走到節點 5，共經歷了 2 個節點。

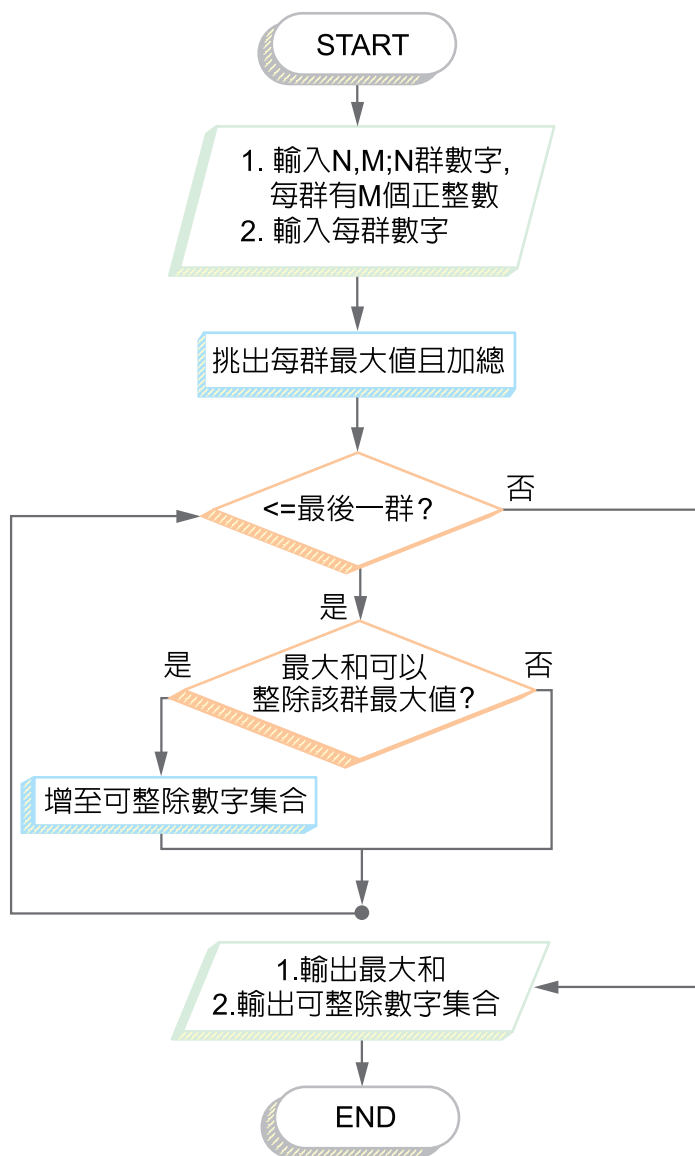
節點 5 爆炸，下一個歷程的起始點移至下一個節點，也就是節點 3，如圖 7.5。

至此，完成爆炸 4 次的要求，剩餘節點 3，是最後的幸運者。



◆ 圖 7.5 $S = 3, E = 5, K = 4$

6.3 思考方向



◆ 圖 6.1 大概思考方向流程圖



6.4 Python 程式

```
00
01     scale = input('')
02     L = scale.split()
03     M = [int(g) for g in L]
04     ma2Drow = M[0]
05     ma2Dcol = M[1]
06     ma2D = [[0 for _ in range(ma2Dcol)] for _ in range(ma2Drow)]
07     Maxnumber = []
08
09     for i in range(ma2Drow):
10         mstr = input('')
11         MS = mstr.split()
12         MM = [int(g) for g in MS]
13
14         for j in range(ma2Dcol):
15             ma2D[i][j] = MM[j]
16         maxnum = max(ma2D[i])
17         Maxnumber.append(maxnum)
18
19     maxsum = sum(Maxnumber)
20     print (maxsum)
21
22     dividend = ''
23     for i in Maxnumber:
24         if maxsum % i == 0:
25             dividend += str(i) + ' '
26
27     if dividend:
28         print (dividend)
29     else:
30         print (-1)
```

6.5 程式說明

- ❖ 第 01 行 輸入 2 個正整數 N , M , 會以字串的形式儲存在 `scale`。給定 N 群數字, 每群都恰有 M 個正整數。
- ❖ 第 02 行 使用空格作為分隔, 將字串 `scale` 切割成好幾個子字串, 儲存在陣列 `L`。
- ❖ 第 03 行 將陣列 `L` 的所有字串全部轉換成整數, 儲存在陣列 `M`。
- ❖ 第 04 行 將 `M[0]` 指派給 `ma2Drow`, `M[0]` 是之前輸入的 N 值。
- ❖ 第 05 行 將 `M[1]` 指派給 `ma2Dcol`, `M[1]` 是之前輸入的 M 值。
- ❖ 第 06 行 建立一個新陣列 `ma2D`, 垂直有 `ma2Dcol` 行, 水平有 `ma2Drow` 列, 預設值是每個元素皆為 0。
- ❖ 第 07 行 宣告空陣列 `Maxnumbers` 用來儲存每群數字的最大值。
- ❖ 第 09 行 準備輸入陣列 `ma2D` 的水平列元素, 迴圈要執行 `ma2Drow` 次。
- ❖ 第 10-12 行 輸入一系列數字, 以字串的形式儲存在 `mstr`。使用空格作為分隔, 將字串 `mstr` 切割成好幾個子字串, 儲存在陣列 `MS`。將陣列 `MS` 的所有字串全部轉換成整數, 儲存在陣列 `MM`。第一次看陣列 `MM`, 其實就像是陣列 `ma2D` 水平的第一列, 只不過它是一個單位, 必須再切割處理。
- ❖ 第 14-15 行 陣列 `ma2D` 垂直有 `ma2Dcol` 行, 得再切割 `ma2Dcol` 次。依據 `ma2D[i][j] = MM[j]` 規則, 陣列 `MM` 的第 j 個元素, 對應到 `ma2D` 的第 j 個位置。 i 是外層迴圈, 是指目前在處理陣列 `ma2D` 的第 i 列, 總共要處理 `ma2Drow` 列。全部處理完後, 輸入的數字皆以整數型態填入二維陣列 `ma2D`。
- ❖ 第 16 行 每群整數的最大值 `ma2D[i]` 指派給 `maxnum`。
- ❖ 第 17 行 將 `maxnum` 新增至陣列 `Maxnumber`。
- ❖ 第 19 行 將陣列 `Maxnumber` 的所有元素加總, 結果指派給 `maxsum`。
- ❖ 第 20 行 輸出最大和 `maxsum`。
- ❖ 第 22 行 預設空字串 `dividend`, 用來儲存能整除最大和 `maxsum` 的數字。
- ❖ 第 23 行 讀取陣列 `Maxnumber`, 逐次操作所選出的數字是否可以整除 `maxsum`。
- ❖ 第 24-25 行 如果讀取的數字可以整除 `maxsum`, 餘數等於 0, 將此數字轉為字串, 連接到字串 `dividend` 之後。
- ❖ 第 27-28 行 如果 `dividend` 字串不為空字串, 其值為 `True`, 輸出字串 `dividend`; 輸出一串可以整除最大和的數字。
- ❖ 第 29-30 行 否則, `dividend` 字串仍維持空字串, 輸出 `-1`。



11.7

程式測試

輸入

.....

9

0

4 4 2 6 8 1 6 4 7

3 7 9 6 4 6 8 2 6

1 6 3 4 2 1 4 6 1

7 3 4 2 3 8 9 3 2

9 5 2 1 9 5 6 5 8

2 4 4 2 3 7 8 4 5

2 3 1 2 6 4 3 9 1

6 2 4 5 3 8 7 1 6

5 8 1 3 7 5 2 9 7

正確輸出

.....

912385732424342149683462134536796468263549178354262297134426816
476128516792573185

說明

.....

1. 依路徑長度來檢核，譬如路徑長度 1，9->1，1->2。接著，2->3->8 同方向，路徑長度 2，8->5->7 也是。其餘路徑長度依此類推。

(1) 9 1 2

(2) 3 8 5 7

(3) 3 2 4 2 4 3

(4) 4 2 1 4 9 6 8 3

(5) 4 6 2 1 3 4 5 3 6 7



$$\begin{aligned}
 &= g(3) + 3 + 3 + 3 + 3 + 3 \\
 &= g(1) + 3 + 3 + 3 + 3 + 3 + 3 \\
 &= 1 + 3 + 3 + 3 + 3 + 3 + 3 \\
 &= 19
 \end{aligned}$$

11. 定義 $a[n]$ 為一陣列 (array)，陣列元素的指標為 0 至 $n-1$ 。若要將陣列中 $a[0]$ 的元素移到 $a[n-1]$ ，右側程式片段空白處該填入何運算式？

(A) $n+1$

(B) n

(C) $n-1$

(D) $n-2$

```

int i, hold, n;
...
for (i=0; i<= ; i=i+1) {
    hold = a[i];
    a[i] = a[i+1];
    a[i+1] = hold;
}

```



解答

(D) $n-2$



說明

$\text{hold} = a[i]$ 這三列程式是典型的資料交換寫法。

$i = 0$ 時， $a[0]$ 會換到 $a[1]$ 的位置， $a[1]$ 會換到 $a[0]$ 的位置。

$i = 1$ 時， $a[1]$ 會換到 $a[2]$ 的位置， $a[2]$ 會換到 $a[1]$ 的位置。

.....

$i = n - 2$ 時， $a[n - 2]$ 會換到 $a[n - 1]$ 的位置， $a[n - 1]$ 會換到 $a[n - 2]$ 的位置。

因此，從 $i = 0$ 直至 $n - 2$ ， $a[0]$ 會逐漸換到 $a[n - 1]$ 的位置。