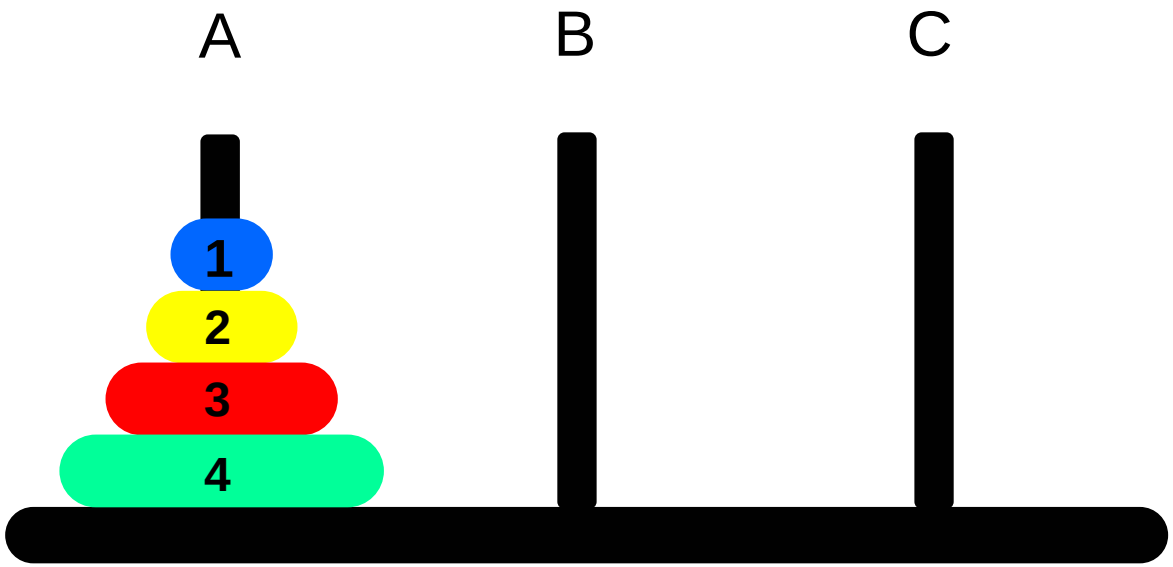
# 迭代版河內塔

丁培毅

# 迭代版河內塔

- 請撰寫一個程式，將一疊由小到大的的盤子由一個柱子上移到另外一個柱子上，並且維持原來的順序：
  - 每一個步驟只能移動一個盤子
  - 任何時候，大的盤子不可以放在比它小的盤子之上
  - 有一個輔助的柱子可以使用




程式輸出

| | |
|---|---|
| 1, A -> B | 1, B -> C |
| 2, A -> C | 2, B -> A |
| 1, B -> C | 1, C -> A |
| 3, A -> B | 3, B -> C |
| 1, C -> A | 1, A -> B |
| 2, C -> B | 2, A -> C |
| 1, A -> B | 1, B -> C |
| 4, A -> C | |

# 分析

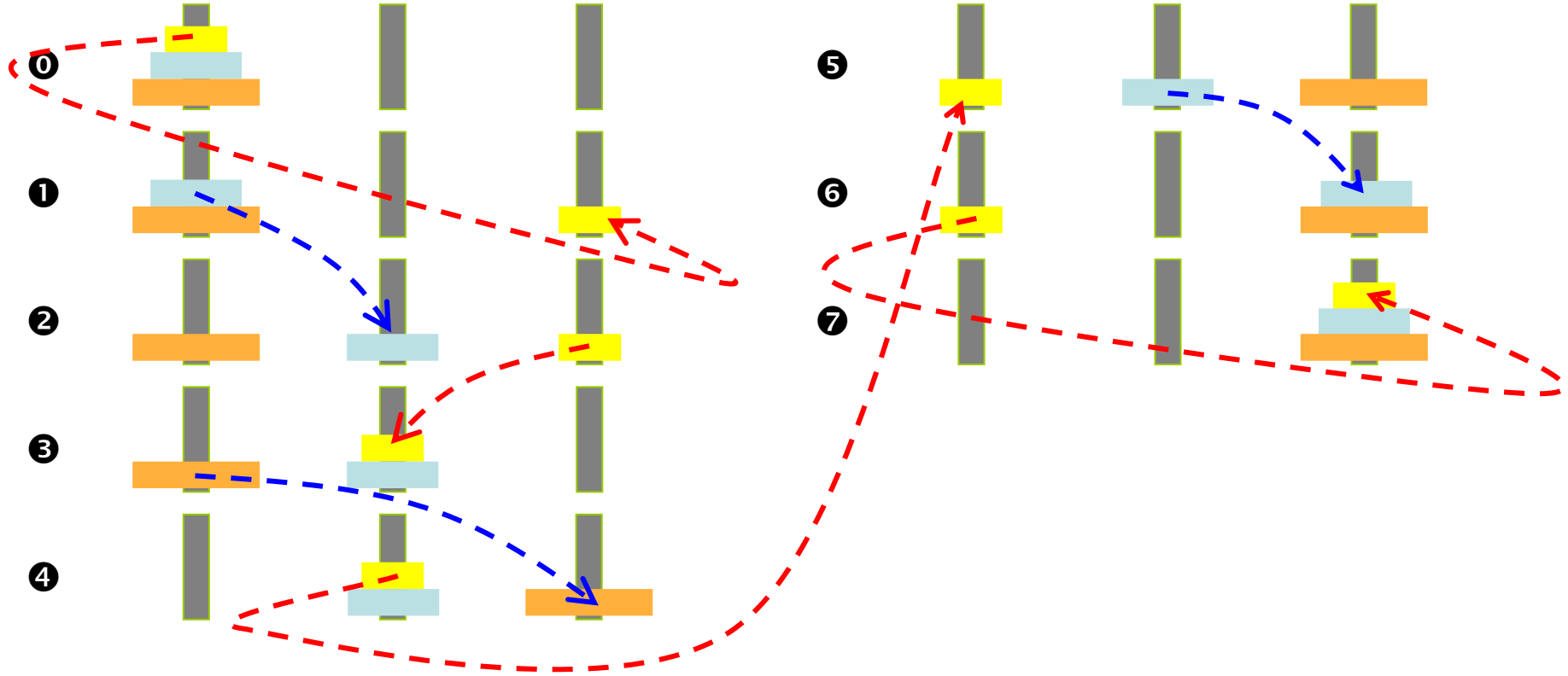
1. 河內塔是一個本質上就具有遞迴特性的問題，把 n 個盤子由 A 柱子搬到 C 柱子 (以 B 爲輔助)，基本上拆解爲三個巨集動作
   ❶ 由 A 搬移 n-1 個盤子到 B (以 C 爲輔助)
   ❷ 由 A 搬移 1 個盤子到 C
   ❸ 由 B 搬移 n-1 個盤子到 C (以 A 爲輔助)

   要設計遞迴的程式相當直覺
2. 但是要設計迴圈版本反而需要特殊的觀察如下：
   ① 基本上需要寫一層迴圈，重複執行
   “挑選一個盤子，搬移到另一柱子上”
   直到所有盤子都在 柱子 C 上
   ② 仔細觀察一下，盤子其實沒得選，每一步驟只有一個盤子可以移動，例如一開始 n 個盤子都在 A 上，只有最小的盤子可以移動，移動的目標柱子的規則視 n 爲奇數還是偶數而定 – 奇數向左 (逆時針)，偶數向右 (順時針)

# 分析 (cont'd)

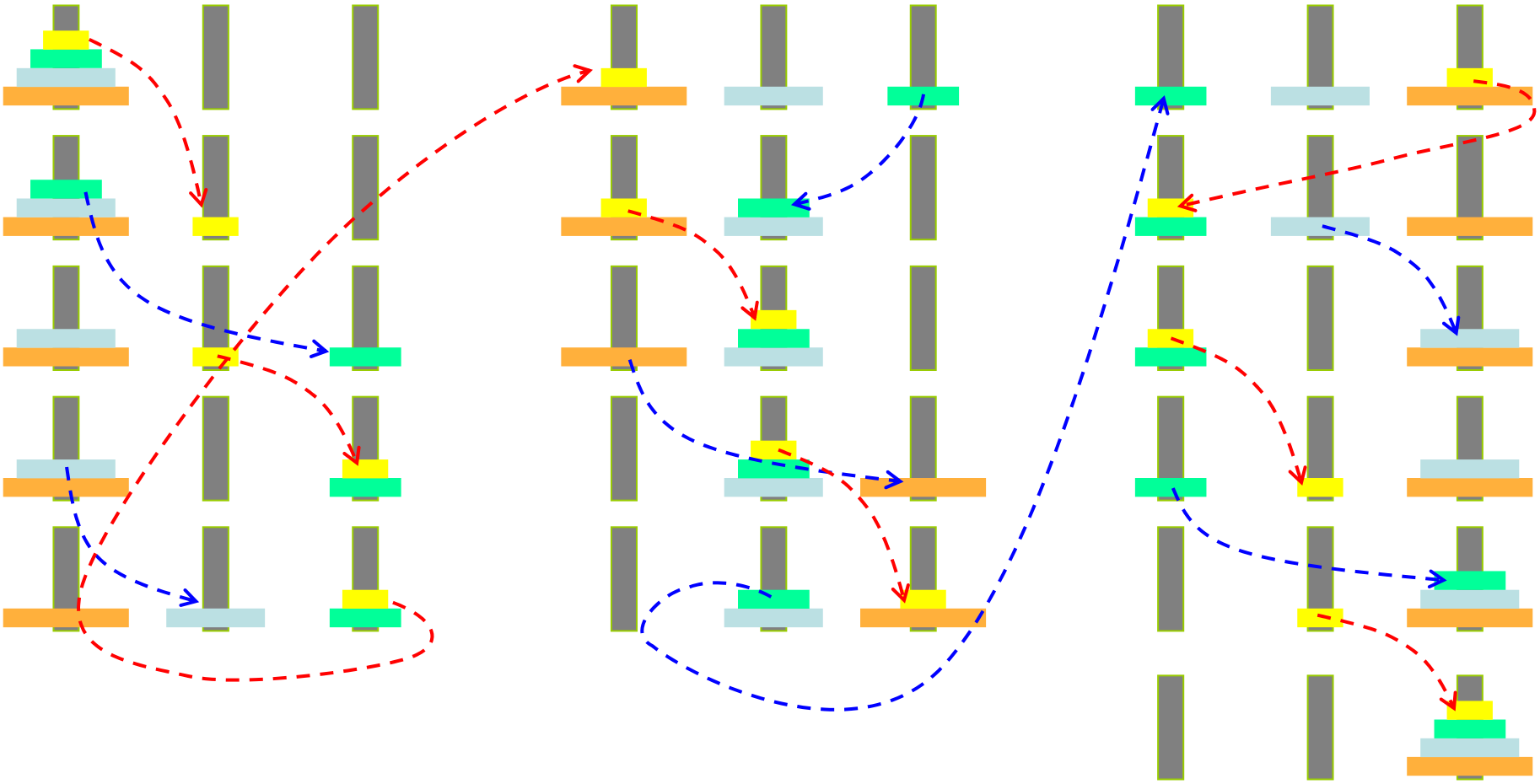
2. ③ 如果前一步驟移動最小盤子，此步驟當然不再移它，剩下兩個柱子上如果都有盤子的話，只能移動比較小的，而且目標沒得選擇 (不能是最小盤子所在的柱子，不能是自己這個柱子，所以目標柱子一定是剩下的那個柱子)；如果剩下兩個柱子裡只有一個柱子上有盤子的話，當然只能移動那個柱子上最上面的盤子，目標柱子也沒得選擇；如果剩下兩個柱子上都沒有盤子的話，其實這個程序會保證所有的盤子都已經搬到柱子 C 上了
3. 這個搬移的程序顯示在後面的兩個範例中，分別是奇數個盤子的範例以及偶數個盤子的範例，先弄清楚範例中搬移的每一個細部動作以後才能夠正確地用程式實作出來

# 盤子個數爲奇數




- **Alternate** moves between the smallest disk and a non-smallest disk.
- **For the smallest**: always move to the right (# of pieces is even), rotate if necessary; **always move to the left (# of pieces is odd), rotate if necessary**
- **For the non-smallest**: there is only one possible legal move.

# 盤子個數爲偶數



- **Alternate** moves between the smallest disk and a non-smallest disk.
- **For the smallest**: always move to the right (# of pieces is even), rotate if necessary; always move to the left (# of pieces is odd), rotate if necessary
- **For the non-smallest**: there is only one possible legal move.

# 程式實作

4. 首先根據前面步驟 2 的觀察 ①，設計下面的迴圈

```
while (the number of disks on peg C are less than n)
{
    ① move smallest disk rightward (n is even) / leftward (n is odd) one peg
    ② move smaller disk other than the smallest to the remaining peg
}
```

5. 變數設計： ❶ 用一個二維整數陣列來表示所有柱子，記錄每一個柱子上的盤子有哪些； ❷ 用一個整數陣列記錄每個柱子上目前有幾個盤子； ❸ 用一個整數變數記錄目前最小盤子所在的柱子； ❹ 用一個整數變數記錄最小盤的移動方向

```
int peg[3][MAXDISKS];
int nDiskPeg[3];
int curSPeg = 0, targetSPeg;
int dir = 1 - nDisks % 2 * 2;
```

```
while (nDiskPeg[2] < nDisks) {
    targetSPeg = (curSPeg+dir+3)%3;
①   peg[targetSPeg][nDiskPeg[targetSPeg]++] =
        peg[curSPeg][--nDiskPeg[curSPeg]];
    curSPeg = targetSPeg;

②   // 判斷 curSPeg 之外兩個柱子上的盤子哪一個
    // 比較小, 移動到另一個柱子上
}
```

6. 如果 curSPeg 是 0, 另外兩個柱子就是 1, 2
   如果 curSPeg 是 1, 另外兩個柱子就是 0, 2
   如果 curSPeg 是 2, 另外兩個柱子就是 0, 1

   然後比較兩個柱子上的最上面盤子的大小，哪一個小就移動哪一個，這裡可以用 if 條件判斷再加上兩個變數記錄判斷出來的來源柱子 srcPeg 和目標柱子 trgtPeg，如下：

```
int srcPeg, trgtPeg;
if (curSPeg == 0)
    if (peg[1][nDiskPeg[1]-1] < peg[2][nDiskPeg[2]-1])
        srcPeg = 1, trgtPeg = 2;
    else
        srcPeg = 2, trgtPeg = 1;
else if (curSPeg == 1)
    …
else if (curSPeg == 2)
    …
 peg[trgtPeg][nDiskPeg[trgtPeg]++] = peg[srcPeg][--nDiskPeg[srcPeg]];
```

   隨便挑一個範例，用上面這段程式嘗試運作一下，很快就會發現一個錯誤：如果某一個柱子上目前沒有盤子，上面程式中 nDiskPeg[i] 會是 0, nDiskPeg[i]-1 會是 -1, 使得 peg[i][nDiskPeg[i]-1] 會發生存取錯誤, 在使用陣列變數時需要非常小心地避免這樣的錯誤

7. 平常有兩種方法處理這樣的問題，

❶ “增加 if 敘述判斷”， 例如：

```
if (curSPeg == 0)
   if ((nDiskPeg[2] == 0) || (peg[1][nDiskPeg[1]-1] < peg[2][nDiskPeg[2]-1]))
      srcPeg = 1, trgtPeg = 2;
   else
      srcPeg = 2, trgtPeg = 1;
else …
```

上面這個判斷式有用到額外的假設：如果 peg[2] 沒有盤子的話，peg[1] 上一定有盤子。另外 A || B 的邏輯敘述如果 A 敘述爲真，整個敘述就爲真，B 敘述完全不會執行。

❷ “讓 peg[i][-1] 是合法的元素而且其中放符合這個程式邏輯的數值”，一開始看到這樣的描述好像有點奇怪，我們先調整陣列的維度，讓每一個柱子多一個元素，然後調整使用的方法，原來使用 peg[i][0]~ peg[i][nDisks-1]，現在使用 peg[i][1]~ peg[i][nDisks]，peg[i][0] 則放一個很大的數字，例如 99，如此當柱子上是空的時，比對的敘述不會使得空的柱子成爲 srcPeg

```
int peg[3][MAXDISKS+1];
peg[0][0] = peg[1][0] = peg[2][0] = 99;
```

```
if (curSPeg == 0)
   if (peg[1][nDiskPeg[1]] < peg[2][nDiskPeg[2]])
      srcPeg = 1, trgtPeg = 2;
   else
      srcPeg = 2, trgtPeg = 1;
else …
```

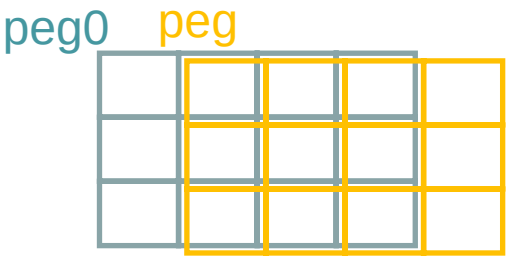
接下來我們再運用指標的語法完全實現

“讓 peg[i][-1] 是合法的元素而且其中放符合這個程式邏輯的數值”

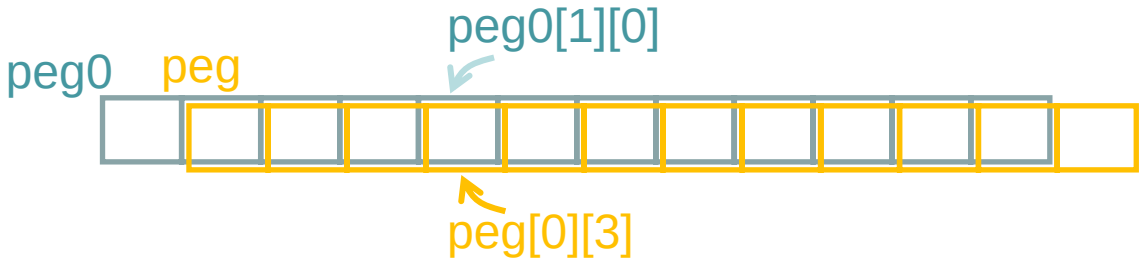
```
int peg0[3][MAXDISKS+1];
int (*peg)[MAXDISKS+1] =
    (int (*)[MAXDISKS+1]) &peg0[0][1];
peg[0][-1] = peg[1][-1] = peg[2][-1] = 99;
```

```
if (curSPeg == 0)
    if (peg[1][nDiskPeg[1]-1] < peg[2][nDiskPeg[2]-1])
        srcPeg = 1, trgtPeg = 2;
    else
        srcPeg = 2, trgtPeg = 1;
else …
```

概念上 peg[][] 陣列是 peg0[][] 陣列的平移，
元素 peg[0][0] 就對應到元素 peg0[0][1]




實際記憶體使用上 peg0 陣列與 peg 陣列的記憶體對照關係如下圖：



接下來我們再運用陣列以查表的方式簡化條件判斷敘述， 我們的輸入資料是 curSPeg: 0~2, 希望得到的輸出是 (srcPeg, trgtPeg)，表列如下

| curSPeg | (srcPeg, trgtPeg) |
|---|---|
| 0 | (1, 2) |
| 1 | (0, 2) |
| 2 | (0, 1) |

```
int table[3][2] = {{1,2}, {0,2}, {0,1}};
j = peg[table[curSPeg][0]][nDiskPeg[table[curSPeg][0]]-1] >
    peg[table[curSPeg][1]][nDiskPeg[table[curSPeg][1]]-1];

srcPeg = j==1 ? table[curSPeg][1] : table[curSPeg][0];
trgtPeg = j==1 ? table[curSPeg][0] : table[curSPeg][1];
```

或是

```
if (j==1)
   srcPeg = table[curSPeg][1], trgtPeg = table[curSPeg][0];
else
   srcPeg = table[curSPeg][0], trgtPeg = table[curSPeg][1];
```

也可以用複雜一點的表格，配合條件判斷運算式來直接存取表格元素

| curSPeg | peg[table[curSPeg][0][0]][nDiskPeg[table[curSPeg][0][0]]-1] > peg[table[curSPeg][0][1]][nDiskPeg[table[curSPeg][0][01]-1]) | (srcPeg, trgtPeg) |
|---|---|---|
| 0 | 0 | (1, 2) |
| 0 | 1 | (2, 1) |
| 1 | 0 | (0, 2) |
| 1 | 1 | (2, 0) |
| 2 | 0 | (0, 1) |
| 2 | 1 | (1, 0) |

```
int table[3][2][2] = {{1,2},{2,1},{0,2},{2,0},{0,1},{1,0}};
j = peg[table[curSPeg][0][0]][nDiskPeg[table[curSPeg][0][0]]-1] >
    peg[table[curSPeg][0][1]][nDiskPeg[table[curSPeg][0][1]]-1];
srcPeg = table[curSPeg][j][0], trgtPeg = table[curSPeg][j][1];
```

此處我們直接使用運算式 a > b 的數值存取陣列元素，如果 a 數值大於 b 則運算式 a>b 爲 1，反之如果 a 數值不大於 b 則運算式 a>b 爲 0