CHAPTER

5

遞迴

5.1 遞迴的運作方式

一個呼叫它本身的函數稱為遞迴 (Recursive)。在撰寫程式中，也常常會應用遞迴來處理某些問題，而這些問題通常有相同規則的性質，如求 n 階層

$$n! = n * (n-1)!$$
$$(n-1)! = (n-1) * (n-2)!$$
$$(n-2)! = (n-2) * (n-3)!$$
$$\vdots$$
$$\vdots$$
$$1! = 1$$

從上述得知其規則為某一數 A 的階層為本身 A 乘以 (A–1) 階層，導循此規則即可求出其結果。

C 片段程式》

```
int fact(int n)
{
    int ans;
    if (n == 1)
        ans = 1;
    else
        ans = n * fact(n-1);
    return ans;
}
```

》程式解說

在撰寫遞迴時，千萬要記住必須有一結束點，使得函數得以往上追溯回去。如上例中，當 n == 1 時，1! = 1 即為結束點。

為了能讓讀者更能體會遞迴的運作，我們以圖形來表示 n 階層的做法；假設 n = 4，其步驟如下(注意箭頭所指的方向)：

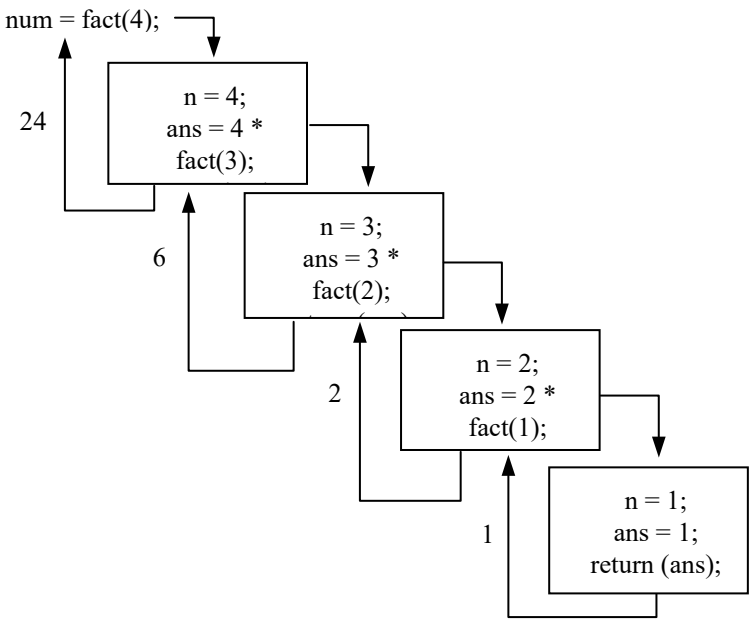


圖 5.1 計算 4! 的遞迴過程

fact(4)表示要計算 4! 的值，此時會呼叫 ans = 4*fact(3)；敘述，但此敘述中又有 fact(3)，表示要計算 3!，因此它又會去呼叫 ans = 3*fact(2)；敘述，同樣的 fact(2) 也會呼叫 ans = 2*fact(1);敘述，由於我們設定當 n == 1 時，便回傳 ans 值到上一層，注意！此時的 ans 是傳給 fact(1) 的，也因如此，使得 2*fact(1) 能加以計算出結果，將結果存於 ans 後，又往上傳給 fact(2)，因為 fact(2) 是呼叫它的，…，依此類推，最後的 4! 便可知道其答案為 4*fact(3)，即 4*6 等於 24。

上述的片段程式是以遞迴的方式執行之，而以非遞迴的方式之片段程式如下：

C 片段程式》

```
long Factorial(long n)
{
    long sum = 1;
    int  i;
    if (n == 0 || n ==1)  /* 當 n=0 或 n=1 時, 0!=1, 1!=1 */
        return (1);  /* 故直接傳回 1 */
    else {
        for (i=2; i<=n; i++ ) /* sum 記錄目前階乘之和 */
```

```
            sum *= i;  /* sum 與 i 相乘之和放回 sum 中 */
    }
    return (sum);
}
```

》程式解說

我們利用了二種處理方式，一為遞迴呼叫方式，二為非遞迴呼叫方式，二種方式的最終答案是一樣的，但撰寫的方式則不同，請讀者比較之。有關 n! 的完整程式，請參閱 5.5 節。

再舉一例，費氏數列(Fibonacci number)表示某一數為其前二個數的和，假設 n_0=1， n_1=1 則

n2= n1+n0=1+1=2
n3= n2+n1=2+1=3
:
:
所以 ni= ni-1+ni-2

其片段程式如下：

C 片段程式》

```
int fibon(int n)
{
    int ans;
    if (n == 0 || n == 1)
        ans = 1;
    else
        ans = fibon(n-1) + fibon(n-2);
    return(ans);
}
```

除了以遞迴的方式可以計算費氏數列外，當然也可以利用非遞迴的方式執行之，其片段程式如下：

C 片段程式》

```
long fibonacci(long n)
{
    long backitem1;    /* 前一項值 */
    long backitem2;    /* 前二項值 */
    long thisitem;     /* 目前項數值 */
    long i;
    if (n == 0)        /* 費氏數列第 0 項為 0 */
        return (0);
    else if (n == 1)   /* 第二項為 1 */
        return (1);
```

```
    else {
        backitem2 = 0;
        backitem1 = 1;
        /* 利用迴圈將前二項相加後放入目前項 */
        /* 之後改變前二項的值至到第 n 項求得 */
        for (i=2; i<=n; i++) {
            /* F(i) = F(i-1) + F(i-2) */
            thisitem = backitem1 + backitem2;
            /* 改變前二項之值 */
            backitem2 = backitem1;
            backitem1 = thisitem;
        }
        return (thisitem);
    }
}
```

其實編譯程式在整理遞迴時，會藉助堆疊將呼叫本身函數的下一個敘述位址儲存起來，待執行結束點後，再將堆疊的資料一一彈出來處理。

有關費氏數列之程式實作，請參閱 5.5 節。

練習題

請利用遞迴和非遞迴的方法求兩整數的 gcd(最大公因數)。

5.2 一個典型的遞迴範例：河內塔

十九世級在歐洲有一遊戲稱為河內塔(Towers of Hanoi)。此遊戲乃由法國數學家 Edouard Lucas 於 1883 年所提出的，它有 64 個大小不同的金盤子，三個鑲鑽的柱子分別為 A、B、C，今想把 64 個金盤子從 A 柱子移至 C 柱子，但可以借助 B 柱子，遊戲規則為：

1. 每次只能搬一個盤子；
2. 盤子有大小之分，而且大盤子在下，小盤子在上。

假設有 n 個金盤子(1, 2, 3, …, n–1)，數字愈大表示重量愈重，其搬移的演算法如下：

1. 假使 n = 1，則
2. 搬移第一個盤子從 A 至 C

否則

1. 搬移 n–1 個盤子從 A 至 B
2. 搬移第 n 個盤子從 A 至 C
3. 搬移 n–1 個盤子從 B 至 C

以 C 撰寫的程式如下：

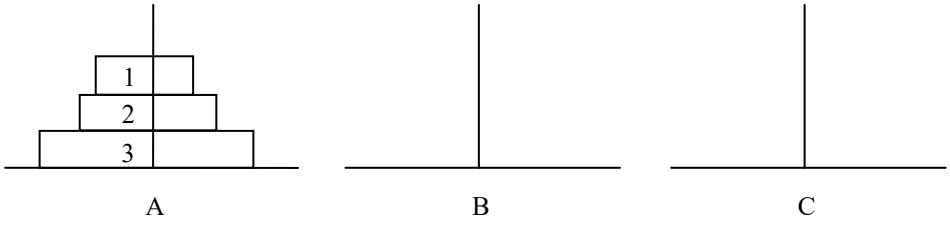
C 片段程式》河內塔

```
void tower(int n, char from, char aux, char to)
{
    if (n == 1)
        printf("Move disk 1 from %c to %c\n", from, to);
    else {
        /* 將 from 標記中的 n-1 個金盤子，藉助 to 標記的柱子，搬到 aux 標記的柱子 */
        tower(n-1, from, to, aux);
        printf("Move disk %d from %c to %c\n", n, from, to);
        tower(n-1, aux, from, to);/* 將 n-1 個盤子從 aux，藉助 from，搬到 to */
    }
}
```

》程式解說

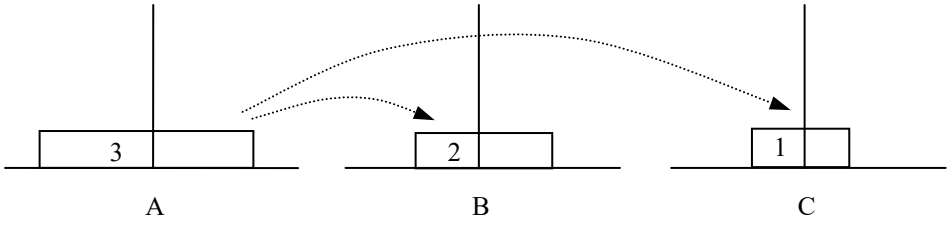
程式 tower 函數中有四個參數，表示有 n 個金盤子，從 from 標記的柱子，藉由標記 aux 的柱子，搬移到 to 標記的柱子上。

假設以 3 個金盤子為例：從 A 柱子搬到 C 柱子，而 B 為輔助的柱子。



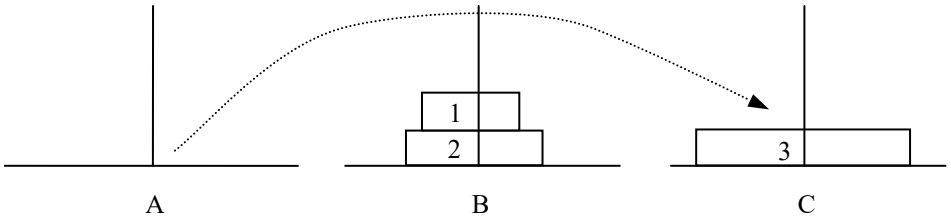
》說明

1. 將 1 號金盤子從 A 搬到 C
2. 將 2 號金盤子從 A 搬到 B，結果如下圖所示：



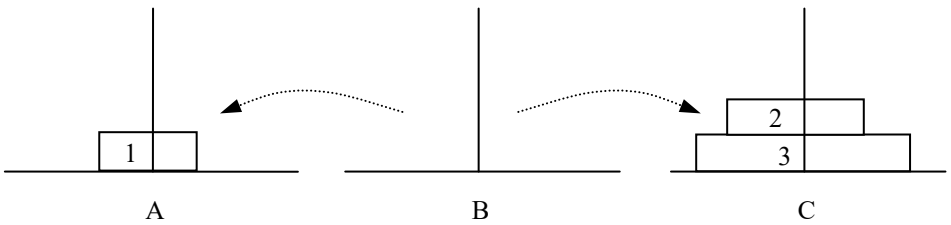
3. 將 1 號金盤子從 C 搬到 B

4. 將 3 號金盤子從 A 搬到 C，結果如下圖所示：

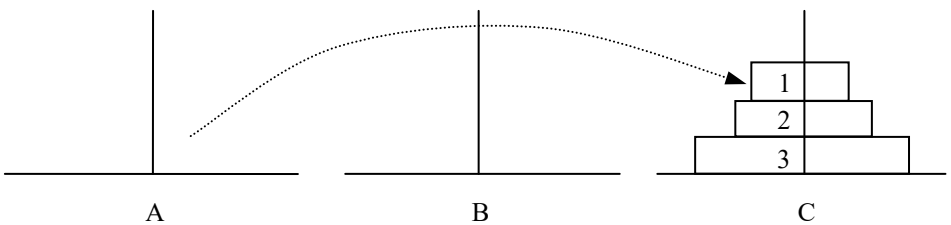


5. 將 1 號金盤子從 B 搬到 A

6. 將 2 號金盤子從 B 搬到 C，結果如下圖所示：

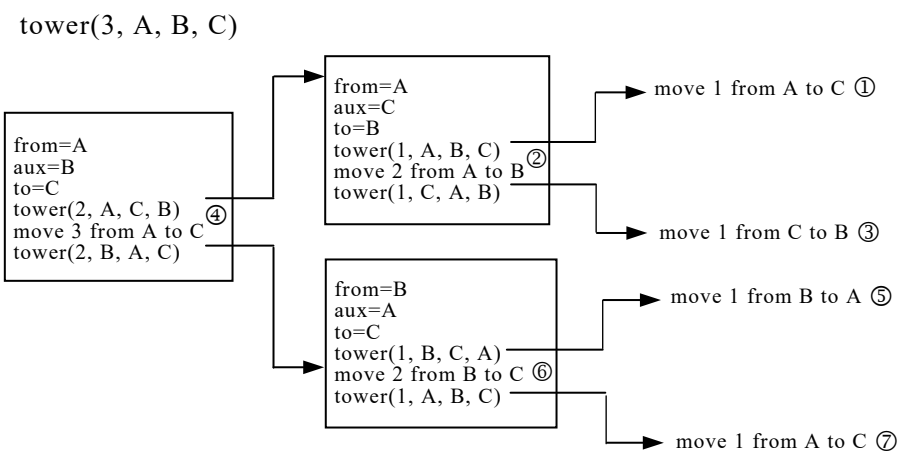


7. 將 1 號金盤子由從 A 搬到 C，結果如下圖所示：



完成了！

程式的追蹤如下：



5.4 何時不要使用遞迴？

遞迴雖然可以使用少數幾行的敘述就可解決一複雜的問題，但有些問題會花更多的時間來處理，因此我們將要探討「何時不要使用遞迴」這一主題。

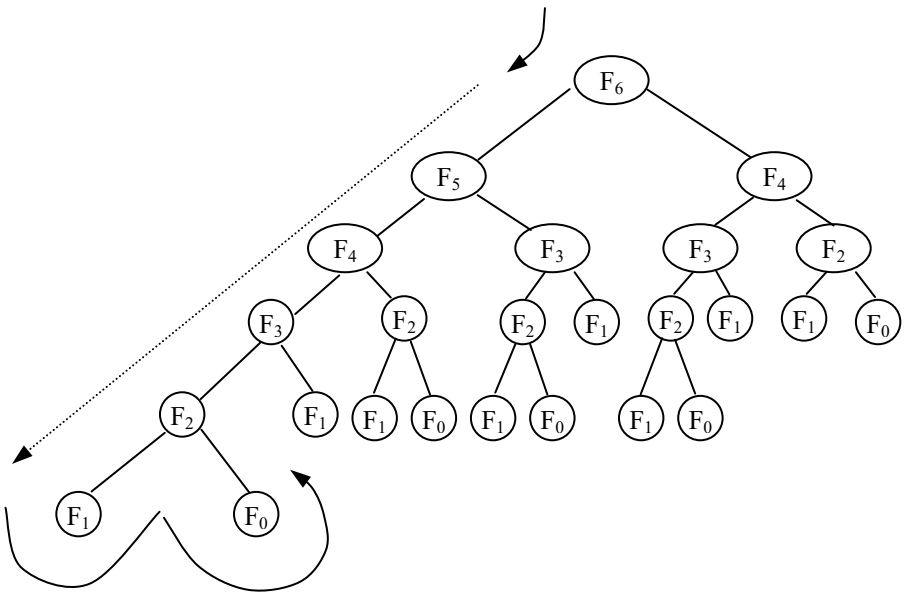
來看前面曾提及的費氏數列的計算方法，某一數乃是前二位數的和，如

$F_0 = 1, F_1 = 1, F_2 = F_1+F_0 = 2$
依此類推

$F_n = F_{n-1}+F_{n-2}$ 對 $n \geq 2$ 而言

以遞迴處理時，其遞迴樹(recursive tree)如下：

假設 n = 6



從上圖得知，F_4、F_3、F_2、F_1 重覆執行多次，像這種遞迴樹長得像灌木，而且重覆的動作又多，在這種情況不適合使用遞迴。此處可證明 F_n 其時間的複雜度是 $2^{n/2}$ 指數 (exponential) 型態。

若以非遞迴，即以反覆的(iterative)程式來執行，其程式為

C 片段程式》

```
int fibonaci(int n)
{
    int ans, i;
    int backone = 1, backtwo = 0;
    if (n == 0 || n == 1)
        ans = 1;
    else {
```

```
        for (i=2; i<=n; i++) {
            ans = backone + backtwo;
            backtwo = backone;
            backbone = ans;
        }
    }
    return ans;
}
```

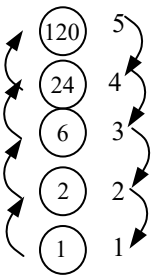
從此程式可以容易地看出其 Big-O 為 O(n)，表 5.1 是比較費氏數列以遞迴和非遞迴在不同數目下所需花費的時間。摘自 Richard Neapolitan 與 Kumarss Naimipour 所著的《Foundations of Algorithms》一書(John Barnette 出版社，1998)。

表 5.1 費氏數列利用遞迴和非遞迴所花費時間之比較
(1 *ns* = 10^{-9} second ; 1 μs = 10^{-6} second)

n	n+1	非遞迴	遞迴
40	41	41 ns	1048 μ s
60	61	61 ns	1 s
80	81	81 ns	18 min
100	101	101 ns	13 days
120	121	121 ns	36 years
160	161	161 ns	3.8 x 107 years
200	201	201 ns	4 x 1013 years

從此表可知，不當的使用遞迴去解決某一問題，可能導致需花更多時間。

再來看處理 n! 的程式，本章開始時已對 n!的遞迴有所交待了，其遞迴樹如下：以 5! 為例



圓圈內的值表示其某一階層的結果，而圓圈右邊表示 5!會先呼叫 4!，而 4!也會先去呼叫 3!，以此類推，最後 1!為 1，此時將 1 傳給 2!以計算 2!，…，最後，5!為 120。

從右邊往下做，直到 1 之後，再往上面繼續做，這種遞迴樹不像一棵灌木(長得很強壯，不是瘦高型)，而是一種簡單的喬木型式。這種情形也不太適合用遞迴，而以非遞迴的方式處理較佳，分析如下：

遞迴

```
int fact(int n)
{
    int ans;
    if (n==1)
        ans = 1;
    else
        ans = n*fact(n-1);
    return ans;
}
```

非遞迴

```
int factorial(int n)
{
    int i, ans = 1;
    for (i=2; i<=n ; i++)
        ans *= i;
    return ans;
}
```

由於非遞迴使用了較多的區域變數 (local variable)，所以直覺上以為遞迴會較好，其實不然，因為遞迴使用了更多的空間來存放暫時的結果，所以實際上遞迴花了更多的時間，因此，類似上述這種遞迴樹相當簡單，有如一條通道的，建議您還是使用非遞迴較佳。

而在遞迴樹長得有如一棵灌木 (如費氏數列的圖) 但重覆的動作很少時，此時大可利用遞迴來解決您的問題。如前述八個皇后的樹狀圖。

5.5 程式集錦

(一) 利用遞迴方式計算 n!

C 程式語言實作》

```
/* File name: factorUsingRecur.c */
/* Version 5.0, November, 2021 */

#include <stdio.h>
#include <ctype.h>

/* 函數原型宣告 */
long factorial(long);

int main()
{
    char ch;
    long n;
    printf("-----Factorial counting Using Recursive----");
    do {
        printf("\nEnter a number( 0<=n<=12 ) to count n!: ");
        scanf("%ld", &n);
```

```
        while (getchar() != '\n')
            continue;
        /* n 值在一般系統中超過 12 會產生 overflow 得到不正確的值 */
        if (n < 0 || n > 12)
            printf("input out of range !\n");
        else
            printf("%ld! = %ld\n", n, factorial(n));
        printf("Continue (y/n)? ");
        ch = toupper(getchar());
    } while (ch == 'Y');
    return 0;
}

/* 利用遞迴呼叫自己計算 N 階層 */
long factorial(long n)
{
    if (n == 1 || n== 0)    /* 設定結束點 */
        return (1);
    else
        return( n * factorial(n-1));
}
```

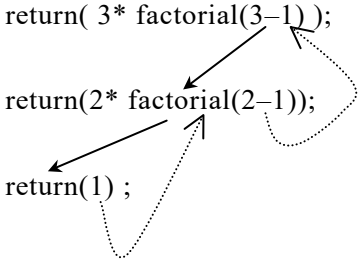
輸出結果

```
-----Factorial counting Using Recursive----
Enter a number( 0<=n<=12 ) to count n!: 5
5! = 120
Continue (y/n) ? n
```

》程式解說

ex：n = 3 時，遞迴函數執行如下：

```
if ( n == 1)
    return (1);
else
    return( 3* factorial(3–1) );
```



當執行到 return(1)時，便將此結果，傳回給 factorial(2–1)，如虛線所示，之後，又將 2*factorial(2–1) 計算出來的結果傳回給 factorial(3–1)，最後 3*factorial(3–1) 所計算的結果就是 3! 的答案。此處要記得設定結束點，否則程式會形成無窮的呼叫。

```
Enter a number(n>=0): 25
Fibonacci(25) = 75025
Contiune (y/n)? n
```

》程式解說

費氏數列為 0, 1, 1, 2, 3, 5, 8, 13, 21，其中某一項為前二項之和，且第 0 項為 0，第 1 項為 1。

(四) 利用非遞迴方式計算費氏數列

C 片段程式》

```
/* File name: fiboUsingIter.c */
/* Version 5.0, November, 2021 */

#include <stdio.h>
#include <ctype.h>

/* 函數原型宣告 */
long fibonacci(long);

int main()
{
    char ch;
    long n;
    printf("-----Fibonacci numbers Using Iterative-----");
    do {
        printf("\nEnter a number(n>=0): ");
        scanf("%ld", &n);
        while (getchar()!='\n')
            continue;
        /* n 值大於 0 */
        if (n < 0)
            printf("Input number must be > 0!\n");
        else
            printf("Fibonacci(%ld) = %ld\n", n, fibonacci(n));
        printf("Continue (y/n)? ");
        ch = toupper(getchar());
    } while (ch == 'Y');
    return 0;
}

long fibonacci(long n)
{
    long backitem1;    /* 前一項值 */
    long backitem2;    /* 前二項值 */
    long thisitem;     /* 目前項數值 */
    long i;
    if (n == 0)    /* 費氏數列第 0 項為 0 */
```

```
        return (0);
    else if (n == 1)    /* 第二項為 1 */
        return (1);
    else {
        backitem2 = 0;
        backitem1 = 1;
        /* 利用迴圈將前二項相加後放入目前項 */
        /* 之後改變前二項的值至到第 n 項求得 */
        for (i=2; i<=n; i++) {
            /* F(i) = F(i-1) + F(i-2) */
            thisitem = backitem1 + backitem2;
            /*改變前二項之值*/
            backitem2 = backitem1;
            backitem1 = thisitem;
        }
        return (thisitem);
    }
}
```

輸出結果

```
/* File name: hanoiTower.c */
/* Version 5.0, November, 2021 */

#include <stdio.h>
/* 函數原型宣告 */
void hanoiTower(int, char, char, char);

int main()
{

    int n;
    char A='A', B='B', C='C';
    printf("-----Hanoi Tower Implementation----\n");
    /* 輸入共有幾個盤子在 A 柱子中 */
    printf("How many disks in A? ");
    scanf("%d", &n);
    if (n == 0)
        printf("no disk to move\n");
    else
        hanoiTower(n, A, B, C);
    return 0;
}

/* 遞迴函數呼叫求河內塔之解 */
void hanoiTower(int n, char a, char b, char c)
{
    if (n == 1)
        printf("Move disk 1 from %c -> %c\n", a, c);
    else {
        /* 將 A 上 n-1 個盤子借助 C 移至 B */
```

```
        hanoiTower(n-1, a, c, b);
        printf("Move disk %d from %c -> %c\n", n, a, c);
        /* 將 B 上 n-1 個盤子借助 A 移至 C */
        hanoiTower(n-1, b, a, c);
    }
}
```

輸出結果

```
-----Hanoi Tower Implementation----
How many disks inA ? 3
Move disk 1 from A -> C
Move disk 2 from A -> B
Move disk 1 from C -> B
Move disk 3 from A -> C
Move disk 1 from B -> A
Move disk 2 from B -> C
Move disk 1 from A -> C
Press any key to continue
```

》程式解說

河內塔問題之目的乃在三根柱子中，將 n 個盤子從 A 柱子搬到 C 柱中，每次只移動一盤子，而且必須遵守每個盤子都比其上面的盤子還要大的原則。

河內塔問題的想法必須針對最底端的盤子。我們必須先把 A 柱子頂端 n–1 個盤子想辦法(借助 C 柱)移至 B 柱子，然後才能將最底端的盤子移至 C 柱。此時 C 有最大的盤子，B 總共 n–1 個盤子，A 柱則無。只要再借助 A 柱子，將 B 柱 n–1 個盤子移往 C 柱即可：

```
    hanoiTower(n-1, A, C, B); /* 將 A 頂端 n-1 個金盤子藉助 C 移至 B */
    hanoiTower(n-1, B, A, C); /* 將 B 上的 n-1 個金盤子藉助 A 移至 C */
```

(五) 利用遞迴方式解決 4 個皇后問題

C 程式語言實作》

```
/* File name: queen4.c */
/* Version 5.0, November, 2021 */

#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define MAXQUEEN 4
#define ABS(x) ((x>0) ?(x): -(x))  /* 求 x 之絕對值 */

/* 存放 4 個皇后之列位置，陣列註標為皇后的行值置 */
int queen[MAXQUEEN];
int totalSolution = 0;  /* 計算共有幾組解 */
```

```
void outputSolution()
{
    int x, y;
    totalSolution += 1;
    printf("Solution #%3d\n\t", totalSolution);
    for (x=0; x<MAXQUEEN; x++) {
        for (y=0; y<MAXQUEEN; y++)
            if (x == queen[y])
                printf("Q");
            else
                printf("-");
        printf("\n\t");
    }
    printf("\n");
}
```

輸出結果（請讀者檢視輸出結果是否與範例所列的答案相同）

```
Solution #  1
    --Q-
    Q---
    ---Q
    -Q--

Solution #  2
    -Q--
    ---Q
    Q---
    --Q-
```

5.6 動動腦時間

1. 發揮您的想像力，舉一、二個範例，並詳細說明其遞迴的做法。[5.1]
2. 承上題，將您所舉的範例實際以 C 語言執行之。[5.1]
3. 在 5.2 節中，若有 3 個金盤子，則需 7 次的搬移，請導出若有 n 個金盤子，則需多少次的搬移。[5.2]
4. 在 5-2 節中假設 A 和 C 柱子不能直通，需經過 B 柱子才可以，則共需多少次的搬移。[5.2]