資料結構與演算法入門：第 2 章

窮舉法與遞迴

悠太翼 Yuuta Tsubasa

July 16, 2025

什麼是窮舉法？

- 窮舉法（Brute Force）：把所有可能的情況都試一遍
- 是最直覺、最通用的解法
- 適合問題空間不大的場合

優點	實作簡單 一定能找出正確答案（若有） 適用於所有問題類型
缺點	效率低，無法處理太大輸入 計算量可能爆炸性成長 缺乏最佳化

範例一：索尼克比賽示範關卡安排

- 關卡列表：[Green Hill, Seaside Hill, City Escape]
- 玩家列表：[SonicBoss, Jerry, Zexas]
- 從中挑一個人來玩一關，列出所有可能的組合：

```
vector<string> stages = {"Green Hill", "Seaside Hill", "
    City Escape"};
vector<string> players = {"SonicBoss", "Jerry", "Zexas"};

for (string stage : stages) {
    for (string player : players) {
        cout << stage << " - " << player << endl;
    }
}
```

- 共列出 $3 \times 3 = 9$ 種組合
- 時間複雜度：$O(n \times m)$，其中 n 是關卡數，m 是玩家數

範例二：找出陣列中兩個數的和為指定值

- 題目：給定一個整數陣列和一個目標值，找出陣列中是否存在兩個數字相加等於目標值。
- 這是一個典型的窮舉法適用問題（暴力解法）：

```cpp
vector<int> nums = {1, 3, 5, 7, 9};
int target = 10;

bool isFound = false;
for (int i = 0; i < nums.size(); i++) {
    for (int j = i + 1; j < nums.size(); j++) {
        if (nums[i] + nums[j] == target) {
            isFound = true;
        }
    }
}
```

- 時間複雜度：$O(n^2)$，需要兩層迴圈比較所有配對
- 空間複雜度：$O(1)$，僅使用常數額外變數

加速方法：使用 target - num 的概念

- 窮舉法效率低，能不能更快？
- 我們可以「邊走邊記錄」，用一個集合記住看過的數字
- 每次檢查是否存在目標值減去目前數字的差值

```
vector<int> nums = {1, 3, 5, 7, 9};
int target = 10;
unordered_set<int> seen;

for (int num : nums) {
    if (seen.find(target - num) != seen.end()) {
        // 找到了 num + (target - num) = target
        found = true;
    }
    seen.insert(num);
}
```

- 時間複雜度：$O(n)$，每個數字最多操作一次
- 空間複雜度：$O(n)$，額外使用一個集合儲存已看過的數字

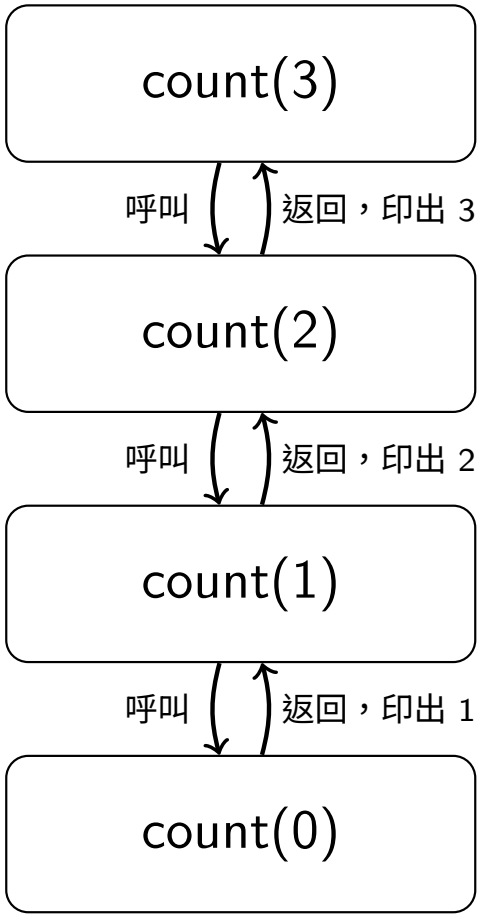
什麼是遞迴？

- 遞迴（Recursion）是一種函式「呼叫自己」的寫法
- 一個遞迴函式必須要有：
  - **基底情況（Base Case）**：不再呼叫自己，直接回傳
  - **遞迴情況（Recursive Case）**：將問題縮小，呼叫自己來解更小的子問題
- 遞迴常用於：
  - 重複的結構（如樹、圖、排列組合）
  - 難以用迴圈表達的邏輯

舉例：從 1 數到 n

```
1 void count(int n) {
2     if (n == 0) return;
3     count(n - 1);
4     cout << n << endl;
5 }
```

遞迴流程圖：count(3)



說明：遞迴先呼叫到底，再從 `count(0)` 返回並印出數字

遞迴寫法（上）：列出每個關卡

- 第一層遞迴：逐一處理每個關卡
- 先不處理玩家部分，只列出關卡

```
vector<string> stages = {"Green Hill", "Seaside Hill", "
    City Escape"};

void listStages(int si) {
    if (si == stages.size()) return;
    cout << stages[si] << endl;
    listStages(si + 1);
}

listStages(0);
```

- 遞迴終止條件為索引超出範圍
- 時間複雜度：$O(n)$，其中 n 是關卡數

遞迴寫法（下）：列出所有關卡與玩家的組合

- 第二層遞迴：處理每個關卡下的所有玩家

```
vector<string> players = {"SonicBoss", "Jerry", "Zexas"};

void listPlayers(int si, int pi) {
    if (pi == players.size()) return;
    cout << stages[si] << " - " << players[pi] << endl;
    listPlayers(si, pi + 1);
}

void listCombinations(int si) {
    if (si == stages.size()) return;
    listPlayers(si, 0);
    listCombinations(si + 1);
}

listCombinations(0);
```

- 雙層遞迴對應雙層迴圈，總時間複雜度為 $O(n \times m)$

遞迴寫法：找出兩數和為目標值（窮舉）

- 使用兩層遞迴枚舉所有組合

```cpp
bool findSumBrute(vector<int>& nums, int i, int j, int target) {
    if (i >= nums.size()) return false;
    if (j >= nums.size()) return findSumBrute(nums, i + 1, i + 2, target);
    if (nums[i] + nums[j] == target) return true;
    return findSumBrute(nums, i, j + 1, target);
}

bool found = findSumBrute(nums, 0, 1, 10);
```

- 每對 (i, j) 都嘗試一次，時間複雜度為 $O(n^2)$

後續待補.....

待續.....