21. 遞迴&回溯法

2025資訊研究社 算法班
Made with ❤️ by jheanlee

遞迴是什麼

遞迴(recurrsion，中國譯為遞歸)指函式執行過程中重複使用自身的過程

知名的遞迴問題如費波那契數列、河內塔問題等
在資料結構、演算法上亦常使用，如二元樹、併查集 (disjoint-set/union find)

遞迴的重要注意事項

1. 初始狀態 (base case) - 有哪些例子需要額外定義的(如 0! = 1、費波那契數列的首二項)
2. 終止條件 - 程式應在何時終止(不終止就會卡死)
3. 重複內容 - 哪些事情是要重複執行的
4. 資料範圍限制 - 執行過程有沒有可能超過可以處理的資料量(如 int 範圍限制)

遞迴解法較難一次想清所有條件及極端情況(edge case)，解題時務必謹慎作答

為什麼要用遞迴

1. 有重複執行的項目，可以切成小塊解決
   -> 分治法 (divide and conquer)、動態規劃 (dynamic programming)
2. 窮舉所有可能、遍歷所有路徑
   -> 回溯法 (backtracking) 、深度優先搜尋法(depth-first search)
3. 單純這樣想比較簡單

注意 : 由於遞迴極易出現bug及漏洞，軟體設計時應極力避免使用。延伸閱讀: 堆疊溢位 (stack overflow)

遞迴的本質、和迭代的比較

遞迴利用的是電腦記憶體區的呼叫堆疊(call stack)，將待處理的遞迴層次儲存

所有遞迴都可以被迭代(iteration，其實就是迴圈) 取代，反之亦然

在多數主流語言中，迭代方式執行起來會較遞迴有效率(時間和空間都是)，而迭代的解法也往往被視為較好的解法(之後學的動態規劃也是一樣

競程比賽不是數學計算比賽

競程比賽不會要你把數學的通式算出來，算出來也沒有意義

程式中的遞迴是讓程式讀起來更簡潔，並簡化問題

遞迴可以被優化(降次降維、memo等等，下學期動態規劃會再教，但絕對不會是硬求出通式來

Leetcode 509 - Fibonacci Number

費波那契數列是非常經典的遞迴例子，定義如下：

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

給予正整數n，回傳費波那契數列的第n 項 F(n)

Example 1:

```
Input: n = 2
Output: 1
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.
```

Example 2:

```
Input: n = 3
Output: 2
Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.
```

Example 3:

```
Input: n = 4
Output: 3
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
```

Leetcode 509 - Fibonacci Number

1. Base case:
   n == 0 時, F(0) = 0
   n == 1 時, F(1) = 1
2. 重複項目:
   F(n) = F(n - 1) + F(n - 2), for all n > 1
3. 終止條件:
   到 base case 即終止
4. 題目限制 0 <= n <= 30 , 不會有溢位情形

Leetcode 509 - Fibonacci Number

這題有非常多優化方法，這裡先教一種，未來還會再回來看他

```
class Solution {
public:
  int fib(int n) {
    if (n == 0) {
      return 0;
    } else if (n == 1) {
      return 1;
    } else {
      return fib(n - 1) + fib(n - 2);
    }
  }
};
```

GCD 最大公因數 - 輾轉相除法

輾轉相除法又叫做歐幾里得演算法(你未來還會聽到他的名字)

簡單來說就是一直 a %= b 、b %= a 、a%=b，直到一個數變0

GCD 最大公因數 - 輾轉相除法

```cpp
int gcd_recursive(int n, int m) {
  if (n < m) return gcd_recursive(m, n);
  if (m == 0) return n;

  return gcd_recursive(m, n % m);
}

int gcd_iterative(int n, int m) {
  int tmp = 1;

  if (n < m) std::swap(n, m);

  while (tmp != 0) {
    tmp = m % n;
    m = n;
    n = tmp;
  }

  return m;
}
```

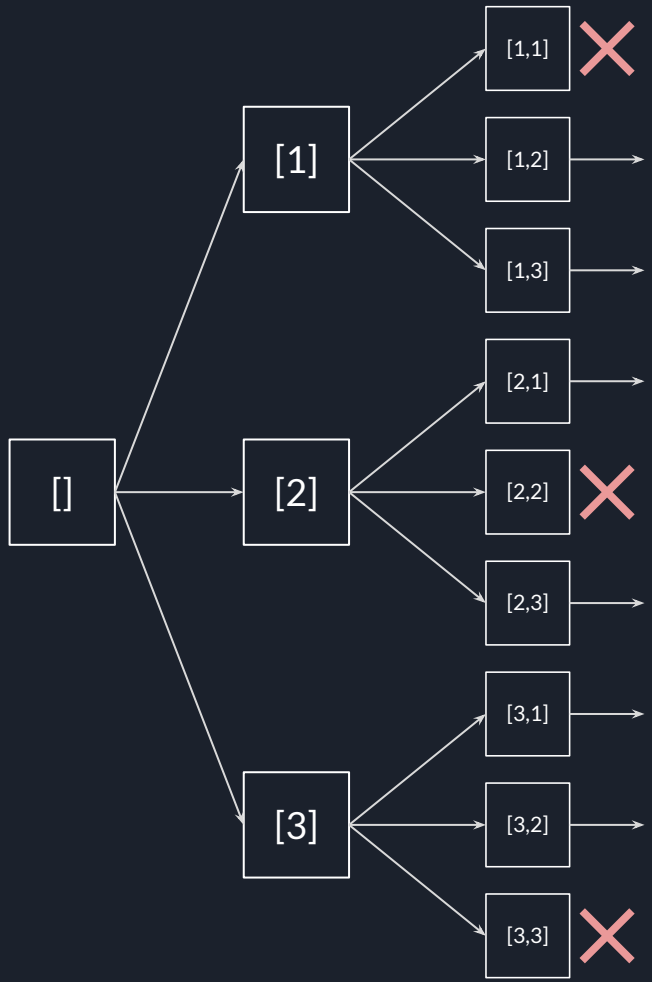
回溯法是什麼

回溯法 (backtracking) 是一種暴力破解法

尋找所有候選解的分枝，並在確定一個解不滿足條件時放棄該分枝剪枝)

回溯法是什麼

例: 1, 2, 3 的不重複排列



Leetcode 46 - Permutations

給予一個不含重複元入的陣列nums，回傳所有重新排列的可能結果。回傳順序不限

Example 1:

> **Input:** nums = [1,2,3]
> **Output:** [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Example 2:

> **Input:** nums = [0,1]
> **Output:** [[0,1],[1,0]]

Example 3:

> **Input:** nums = [1]
> **Output:** [[1]]

Leetcode 46 - Permutations

```cpp
class Solution {
private:
  void backtrack(vector<vector<int>> &output, vector<int> &nums, vector<bool> &used, vector<int> &curr) {
    if (curr.size() == nums.size()) {
      output.push_back(curr);
      return;
    }

    for (int i = 0; i < nums.size(); i++) {
      if (!used[i]) {
        curr.push_back(nums[i]);
        used[i] = true;
        backtrack(output, nums, used, curr);
        used[i] = false;
        curr.pop_back();
      }
    }
  }

public:
  vector<vector<int>> permute(vector<int>& nums) {
    vector<bool> used(nums.size(), false);
    vector<int> curr;
    vector<vector<int>> output;
    backtrack(output, nums, used, curr);
    return output;
  }
};
```

Leetcode 77 - Combinations

給予正整數n 及 k，回傳以 1 ~ n 之間的正整數組合、長度為k 的所有組合(不顧順序 [1, 2] == [2, 1])。回傳順序不限

Example 1:

> **Input:** n = 4, k = 2
> **Output:** [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
> **Explanation:** There are 4 choose 2 = 6 total combinations.
> Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.

Example 2:

> **Input:** n = 1, k = 1
> **Output:** [[1]]
> **Explanation:** There is 1 choose 1 = 1 total combination.

Leetcode 77 - Combinations

```cpp
class Solution {
private:
  void backtrack(vector<vector<int>> &output, int n, int k, vector<int> &curr, int idx) {
    if (curr.size() == k) {
      output.push_back(curr);
      return;
    }

    for (int i = idx; i < n; i++) {
      curr.push_back(i + 1);
      backtrack(output, n, k, curr, i + 1);
      curr.pop_back();
    }
  }
public:
  vector<vector<int>> combine(int n, int k) {
    vector<vector<int>> output;
    vector<int> curr;

    backtrack(output, n, k, curr, 0);
    return output;
  }
};
```

Leetcode 51 - N-Queens

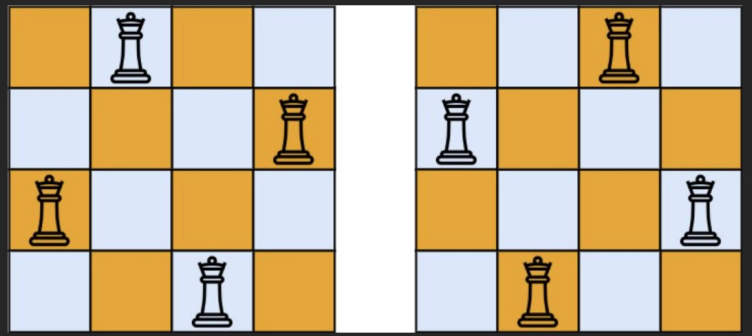
這題是著名的八皇后問題(eight queens puzzle) 的變體

西洋棋中，皇后可以攻擊同一行、同一列或同一(正反) 斜線上的棋子。如何在n x n 的棋盤上擺放n隻皇后，才能使任兩隻皇后無法互相攻擊

給予正整數n，以 'Q' 代表皇后、 '.' 代表空格，回傳所有可能的解法

Leetcode 51 - N-Queens

Example 1:



> **Input:** n = 4
> **Output:** [[".Q..","...Q","Q...","..Q."],
> ["..Q.","Q...","...Q",".Q.."]]
> **Explanation:** There exist two distinct solutions to the 4-queens puzzle as shown above

Example 2:

> **Input:** n = 1
> **Output:** [["Q"]]

Leetcode 51 - N-Queens

```
class Solution {
private:
  bool check_square(vector<string> &board, int qi, int qj) {
    for (int i = 0; i < board.size(); i++) {
      if (board[i][qj] == 'Q') {
        return false;
      }
    }

    for (int i = qi - 1, j = qj - 1; i >= 0 && j >= 0; i--, j--) {
      if (board[i][j] == 'Q') {
        return false;
      }
    }

    for (int i = qi - 1, j = qj + 1; i >= 0 && j < board.size(); i--, j++) {
      if (board[i][j] == 'Q') {
        return false;
      }
    }

    return true;
  }
```

Leetcode 51 - N-Queens

```
25    void backtrack(vector<vector<string>> &output, vector<string> &board, int i) {
26      if (i == board.size()) {
27        output.push_back(board);
28        return;
29      }
30
31      for (int j = 0; j < board.size(); j++) {
32        if (check_square(board, i, j)) {
33          board[i][j] = 'Q';
34          backtrack(output, board, i + 1);
35          board[i][j] = '.';
36        }
37      }
38    }
39
40  public:
41    vector<vector<string>> solveNQueens(int n) {
42      vector<vector<string>> output;
43      vector<string> board(n, string(n, '.'));
44      backtrack(output, board, 0);
45
46      return output;
47    }
48  };
```