Graph

Graph traversal

BFS VS DFS

項目	BFS（廣度優先搜尋）	DFS（深度優先搜尋）
搜尋策略	逐層搜尋（Level by Level）	一條路走到底（Backtrack）
使用資料結構	Queue（佇列）	Stack（堆疊）或遞迴
適用情境	最短路徑、層級搜尋	拓樸排序、迷宮路徑、組合問題
時間複雜度	O(V + E)	O(V + E)
空間複雜度	O(V)（Queue 大小）	O(V)（遞迴 Stack 大小）

🌳 BFS vs DFS 簡介

項目	BFS（廣度優先搜尋）	DFS（深度優先搜尋）
搜尋策略	逐層搜尋（Level by Level）	一條路走到底（Backtrack）
使用資料結構	Queue（佇列）	Stack（堆疊）或遞迴
適用情境	最短路徑、層級搜尋	拓樸排序、迷宮路徑、組合問題
時間複雜度	O(V + E)	O(V + E)
空間複雜度	O(V)（Queue 大小）	O(V)（遞迴 Stack 大小）

📘 BFS 概念（以無向圖為例）

BFS 是從起點出發，把相鄰節點一層一層加入 queue，再逐一訪問。

例如圖：

```
  1
 / \
```

```
  2  3
 / \
4  5
```

BFS 順序：1 → 2 → 3 → 4 → 5

🔧 C++ 實作（使用 adjacency list）

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void BFS(int start, const vector<vector<int>>& graph, vector<bool>& visited) {
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        cout << cur << " ";

        for (int neighbor : graph[cur]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}
```

📙 DFS 概念（以無向圖為例）

DFS 是從起點出發，沿著一條路走到底，若遇到已拜訪過的節點就回頭。

例如圖：

```
   1
  / \
 2   3
/ \
4   5
```

DFS 順序（可能之一）：1 → 2 → 4 → 5 → 3

🔧 C++ 實作（使用 adjacency list）

```cpp
#include <iostream>
#include <vector>
using namespace std;

void DFS(int node, const vector<vector<int>>& graph, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            DFS(neighbor, graph, visited);
        }
    }
}
```

🎯 圖解（ASCII 概念）

圖：

```
1 - 2 - 4
|   |
3   5
```

BFS（起點1）：1 → 2 → 3 → 4 → 5

DFS（起點1）：1 → 2 → 4 → 5 → 3（或其他順序，依實作順序不同）

🧠 重點比較記憶法

- **BFS**：像**波浪一圈圈擴散出去** → 使用 `queue`
- **DFS**：像**挖地道一直往下鑽** → 使用 `stack` 或**遞迴**

延伸:

- 圖的輸入（例如建圖方式）？
- BFS 與 DFS 的應用場景（例如最短路徑或迷宮）？
- 用 stack 實作 DFS（非遞迴版本）？