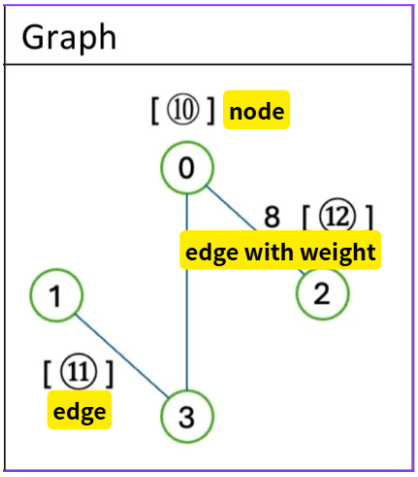
Graphs

一、核心概念

graphs 是由 頂點 (Vertices/Nodes) 和邊 (Edges) 組成的集合，記為 G = (V, E)。

其中 V: 頂點的集，E: 連接頂點的邊的集合。

Definition



Classifications

Type	Definition	Example	p.s.
Undirected Graph (無向圖)	邊沒有方向性 (A-B 等於 B-A)	臉書好友 (雙向關係)	邊是無序對
Directed Graph (有向圖/Digraph)	邊有方向性 (箭頭從 A 指向 B)	IG 追蹤 (單向關注)	邊是有序對 <A, B>
Weighted Graph (加權圖)	邊帶有權重 (數值/Cost)	Google Maps 距離/時間	用於最短路徑計算
Unweighted Graph	All edges equal	Board game map 遊戲版	
Cyclic Graph (有環圖)	路徑可以繞一圈回到原點	城市環狀道路	Tree 是一種 Acyclic (無環) 圖
Connected Graph (連通圖)	任意兩點之間都有路徑相通	道路網	若有孤島則為 Disconnected

頂點的 Degree

- 無向圖：Degree = 連接該點的邊數。

- 有向圖：分為 **In-degree (入度)** 與 **Out-degree (出度)**。

- In-degree: 指向該點的箭頭數量。
- Out-degree: 從該點指出去的箭頭數量。

二、Graph的儲存結構(Representation)

1. Adjacency Matrix (鄰接矩陣)

- 結構：二維陣列 Matrix[i][j]。
- 規則：若 i 和 j 有邊，填 1 (或權重)；否則填 0。
- 優點：查詢兩點是否有邊很快 O(1)，實作簡單。
- 缺點：非常浪費空間 O(V^2)，尤其是在邊很少的「稀疏圖」中。
- 適用：Dense Graph (稠密圖，邊很多時)。

2. Adjacency List (鄰接串列)

- 結構：陣列 + Linked List。每個頂點都有一個 List，紀錄它的鄰居。
- 優點：節省空間 O(V+E)，適合稀疏圖，走訪鄰居效率高。
- 缺點：查詢「A 跟 B 有沒有邊」較慢，需要 traverse list。
- 適用：Sparse Graph (稀疏圖，大部分真實世界應用如 FB 好友、地圖)。

比較

特性	**Adjacency Matrix**	**Adjacency List**
空間複雜度	O(V^2) (空間大)	O(V+E) (空間省)
檢查邊是否存在 (u, v)	O(1) (快)	O(deg(u)) (慢)
走訪所有鄰居	O(V)	O(deg(u))
新增/刪除點	困難 (要重建陣列)	容易

三、Graph Traversal

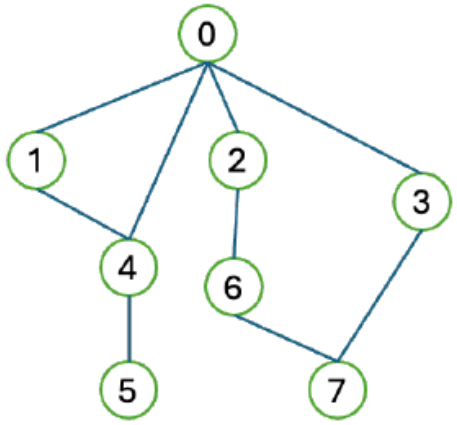
- Depth-First Search (DFS, 深度優先搜尋)

1. **使用堆疊 (Stack)**：此演算法利用「後進先出 (LIFO)」的堆疊（或是程式的遞迴呼叫堆疊）來管理待訪問的節點 。
2. **從選定節點開始**：與 BFS 相同，將起始節點放入堆疊中開始運作 。

3. **深度路徑優先**：不按距離分層，而是選定一條路徑**一條路走到黑**（盡可能往深處鑽，直到無法繼續為止）。
4. **回溯機制 (Backtracking)**：當前節點走到「死胡同」（所有鄰居都已訪問過）時，會**回溯**（Pop 出堆疊）到上一個節點，去探索該節點其他未走過的分支 。
5. **需要已訪問集合**：必須記錄一個「已訪問過的集合 (visited set)」，這點與 BFS 相同，是為了防止在有環 (Cycle) 的圖中重複訪問導致無窮迴圈 。

- Breadth-First Search (BFS, 廣度優先搜尋)

1. **使用佇列 (Queue)**：此演算法利用「先進先出」的佇列來管理待訪問的節點 。
2. **從選定節點開始**：通常從圖形的左上角或被標記為根 (root) 的節點出發 。
3. **逐層訪問**：按節點距離起點的遠近，一層一層地訪問（先訪問距離為 1 的節點，再訪問距離為 2 的，依此類推）。
4. **先廣後深**：確保在往更深層移動之前，當前節點的所有鄰居 (neighbors) 都已先被訪問過 。
5. **需要已訪問集合**：必須記錄一個「已訪問過的集合 (visited set)」，以防止重複訪問節點導致無窮迴圈 。



A : 0,1,4,2,3,5,6,7

Graph vs. Tree 的比較

特性	Tree	Graph
Cycle (環)	無環 (Acyclic)	可能有環
Path	兩點間只有唯一路徑	可能有多條路徑
Root	有唯一的 Root	沒有 Root 的概念
走訪	不需要 Visited 陣列	必須有 Visited 陣列 (防止無窮迴圈)
連通性	必定連通	可能不連通 (Disconnected)