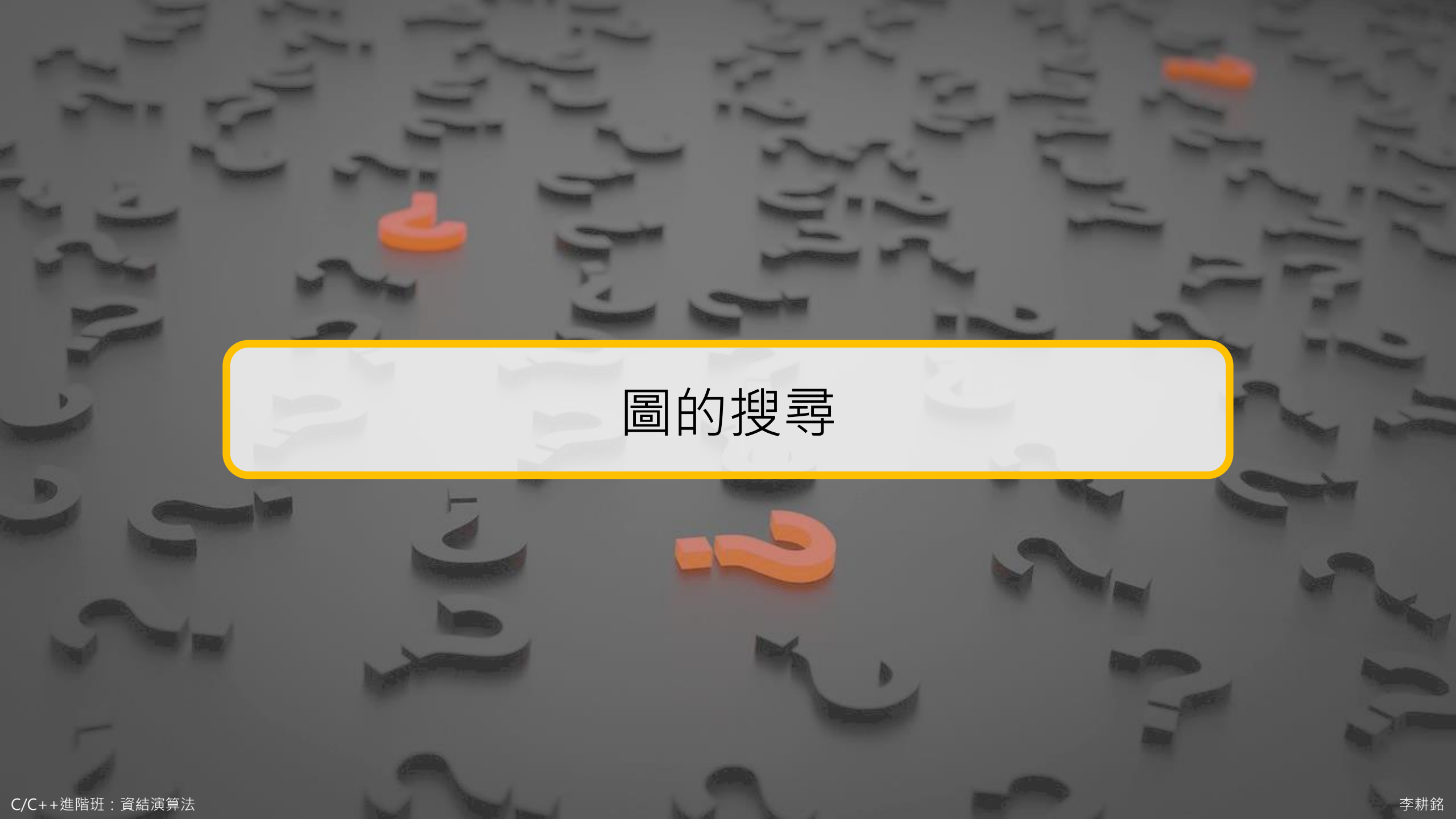
C/C++ 進階班
演算法

廣度優先搜尋
(Breadth-First Search)

李耕銘

課程大綱

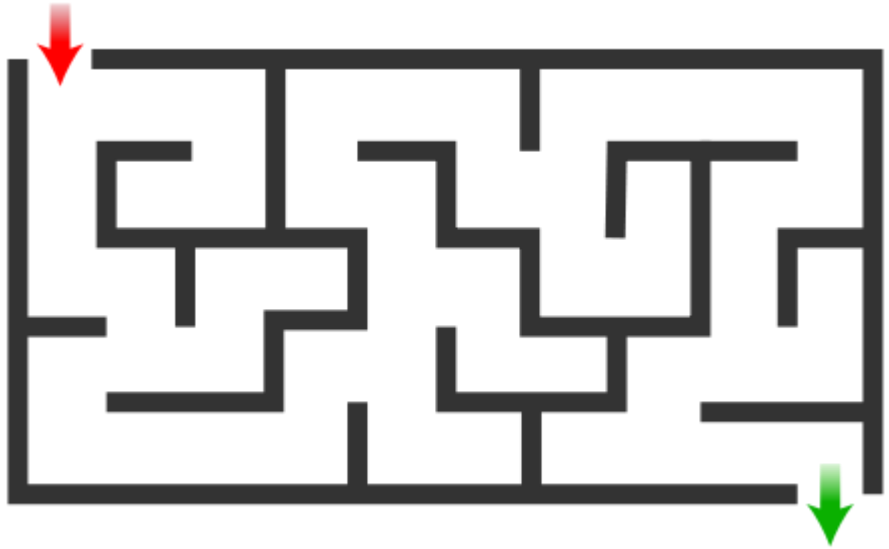
- 圖的搜尋
- 廣度優先搜尋 (BFS)
  - 計算Connected Component
  - 窮舉所有情形
  - 尋找最短路徑
  - 環的存在
- 實戰練習



圖的搜尋

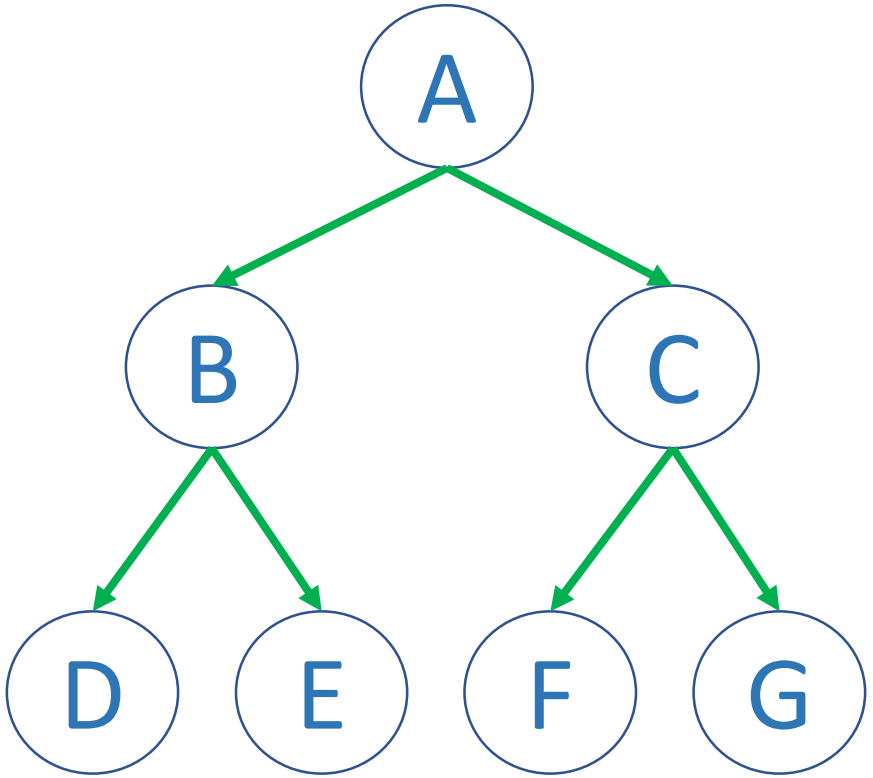
圖的搜尋

- 圖的問題
  1. 兩頂點間是否存在路徑
  2. 兩頂點間的最短路徑
  3. 盡可能把所有的頂點或路徑走過一次
- 回憶：二元樹的走訪
  1. 前序 (Pre-order)
  2. 中序 (In-order)
  3. 後序 (Post-order)
- 需要一個固定的行為準則以供執行
  1. Breadth-First Search (BFS)
  2. Depth-First Search (DFS)



二元樹的尋訪

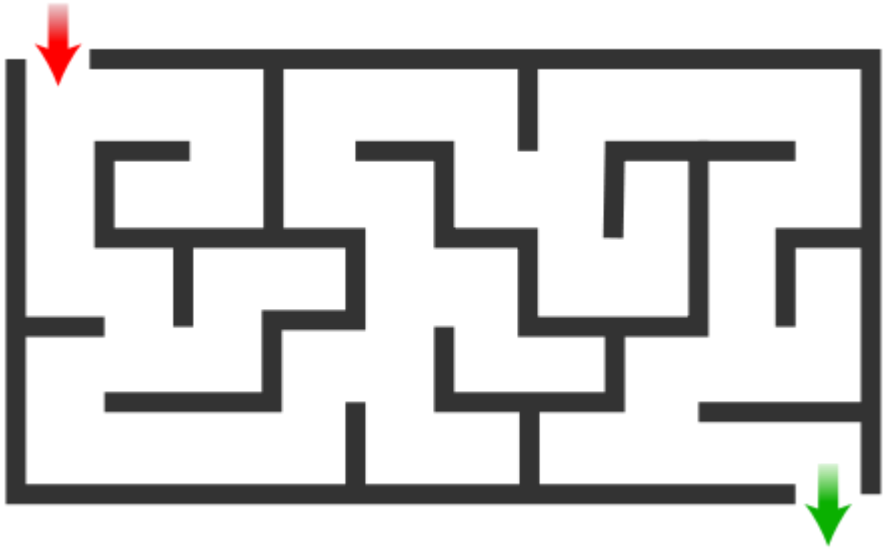
- 前序 (Pre-order)
  - ✓ ABDECFG
  - ✓ 每經過一個新節點就先處理該節點
- 中序 (In-order)
  - ✓ DBEAFCG
  - ✓ 左子樹的資料都處理完再處理該節點
- 後序 (Post-order)
  - ✓ DEBFGCA
  - ✓ 左右子樹的資料都處理完再處理該節點



紅字為根節點

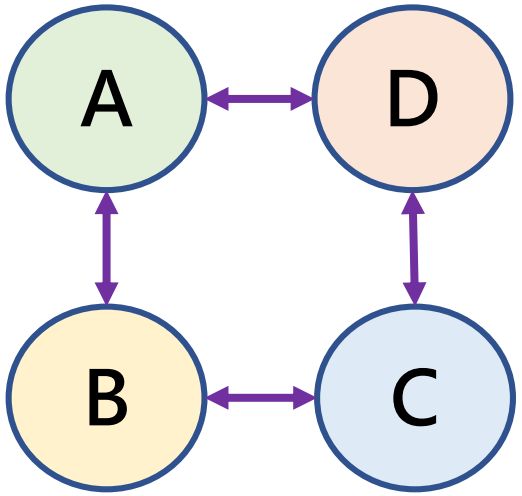
圖的搜尋

- Breadth-First Search (BFS)
  - 先處理完所有的相鄰節點後再往下處理
  - 每條路都淺嘗則止
  - 通常使用 Queue 來達成
- Depth-First Search (DFS)
  - 先往下處理完一相鄰節點後再往下處理
  - 每條路都走到底，不行再回頭
  - 通常使用 Stack 來達成

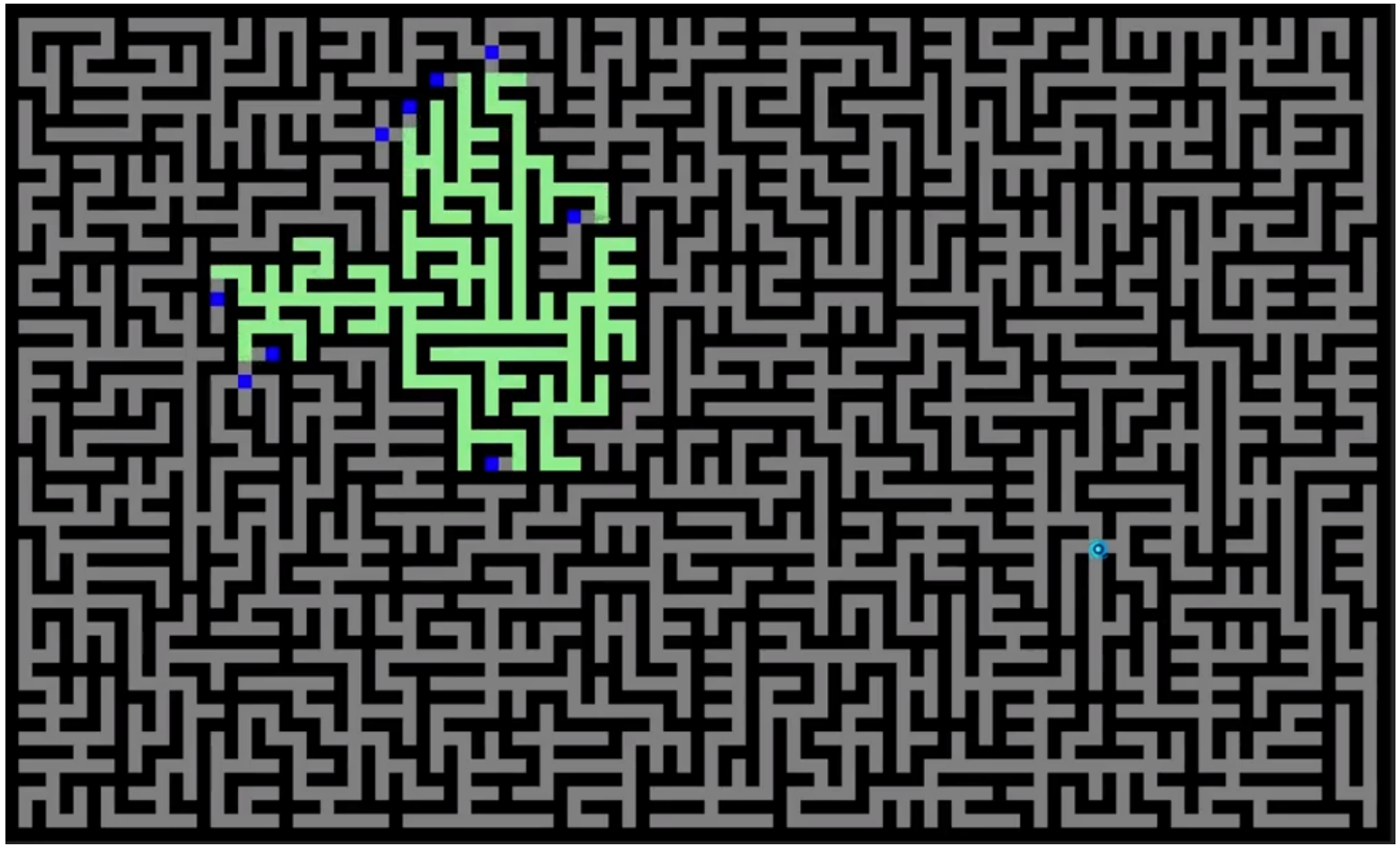


圖的搜尋

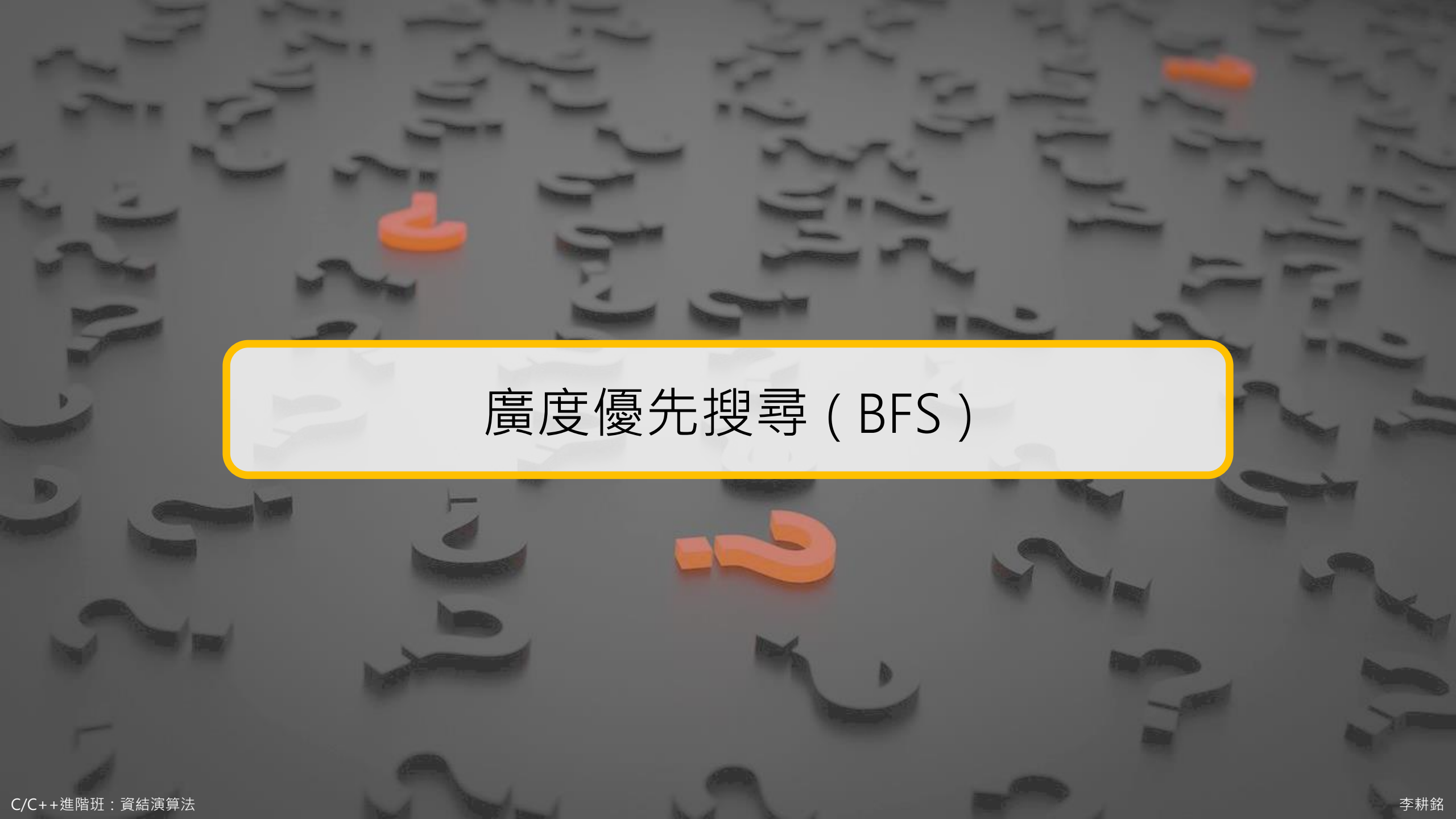
- 為避免無窮迴圈或記憶體浪費
- 把節點分成三種
  1. 白：尚未尋訪過、尚未處理過
  2. 灰：已尋訪過、但尚未處理過
  3. 黑：尋訪而且處理過
- 分成三種狀況
  1. 走到白節點→發現新大陸
  2. 走到灰節點→忽略(BFS)、環(DFS)
  3. 走到黑節點→可以結束該次尋訪



圖的搜尋



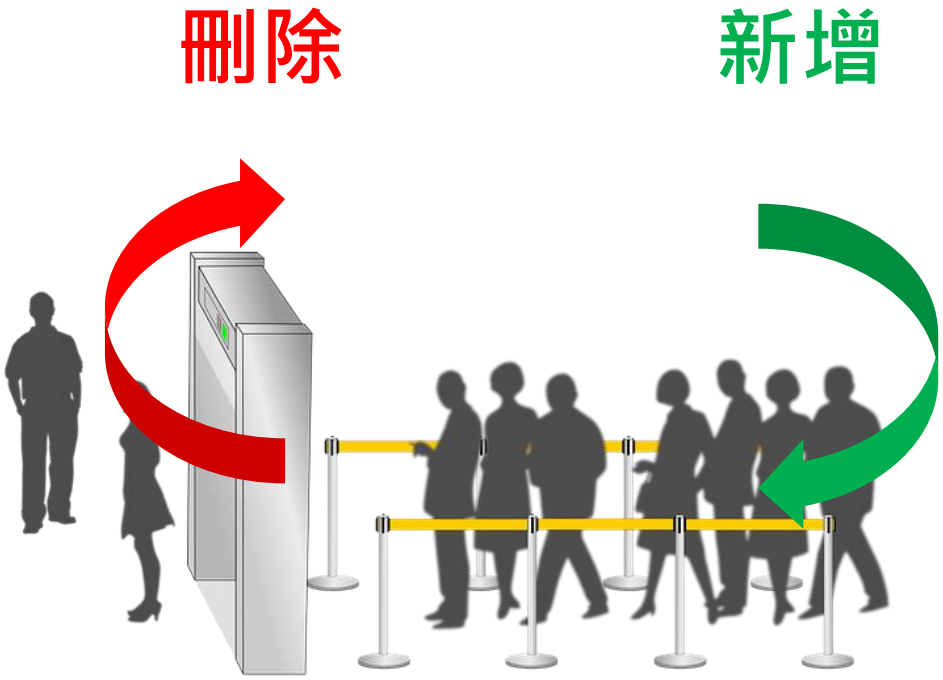
Ref：https://www.youtube.com/watch?v=vf817b882Uw



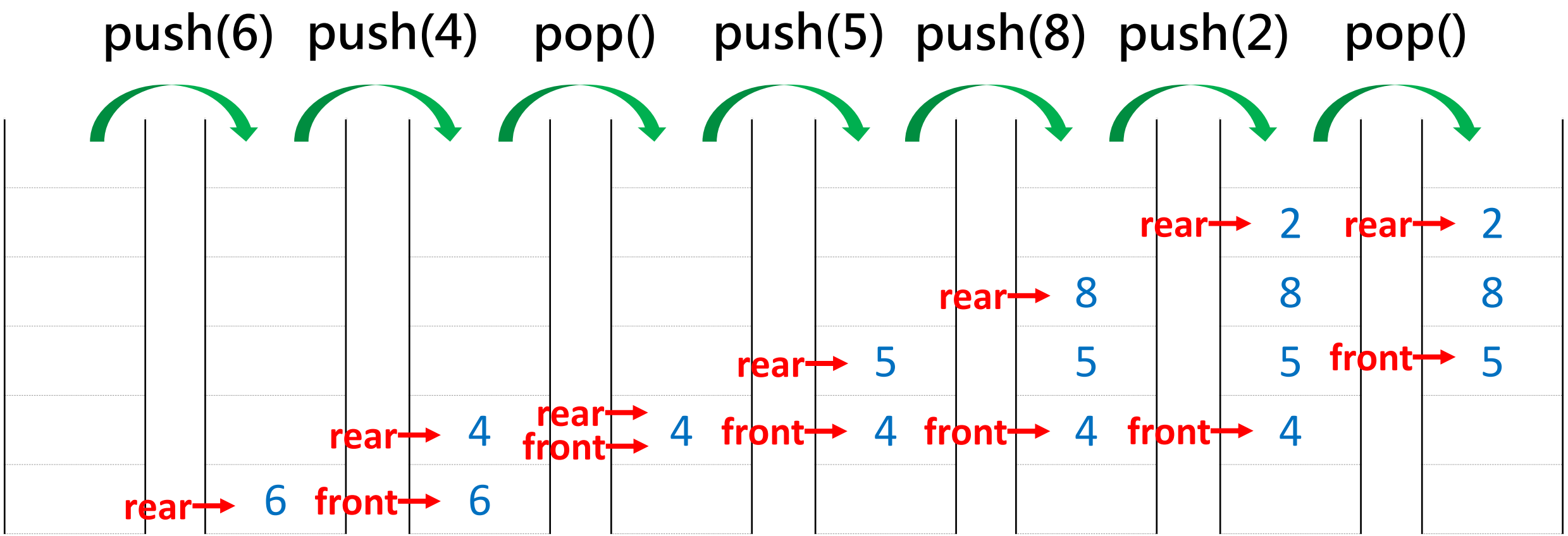
廣度優先搜尋（BFS）

佇列(Queue)

- 佇列(Queue)
  - 插入、刪除在異側
  - first-in-first-out(FIFO)
- 常見的操作
  - push：新增一筆資料
  - pop：刪除一筆資料
  - front：回傳前端的資料。
  - rear：回傳末端的資料
  - empty：確認 queue 裡是否有資料
  - size：回傳 queue 的資料個數



佇列(Queue)

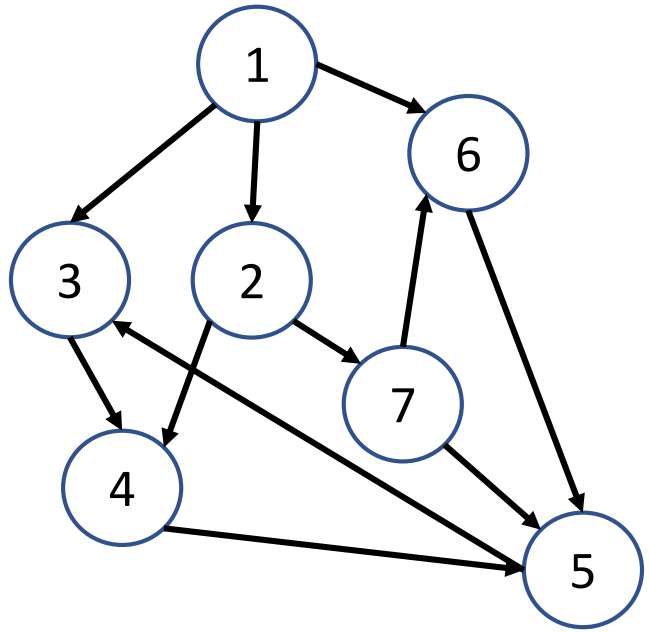


佇列(Queue)的用途

- 依序處理先前的資訊
  - 常用來做資料的緩衝區
    - 記憶體(標準輸出、檔案寫入)
    - 引表機輸出
    - CPU的工作排程
- 迷宮探索、搜尋
  - Breadth-First Search
- 無法得知 queue 裡有哪些資料
  - 只能以 pop() 一個個把資料拿出來

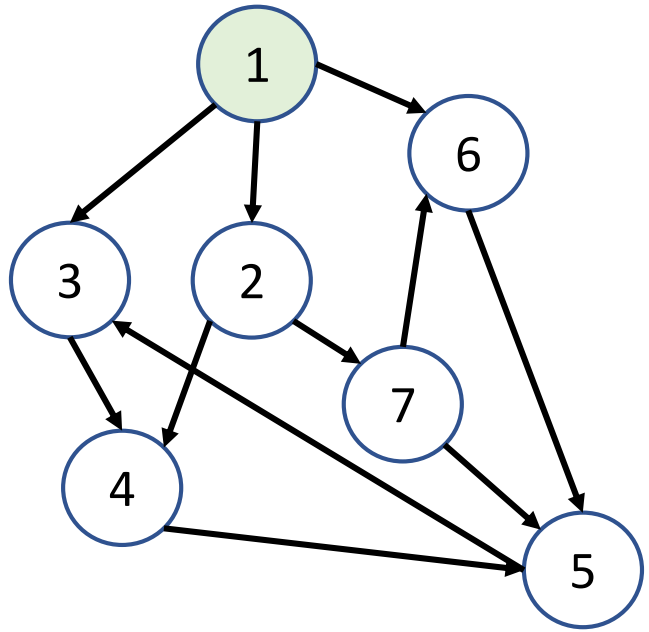
廣度優先搜尋（BFS）

- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



廣度優先搜尋（BFS）

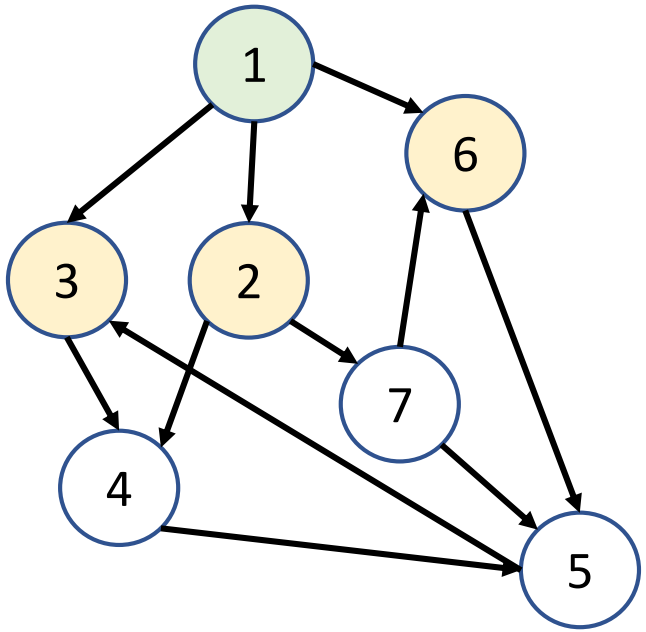
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1																			

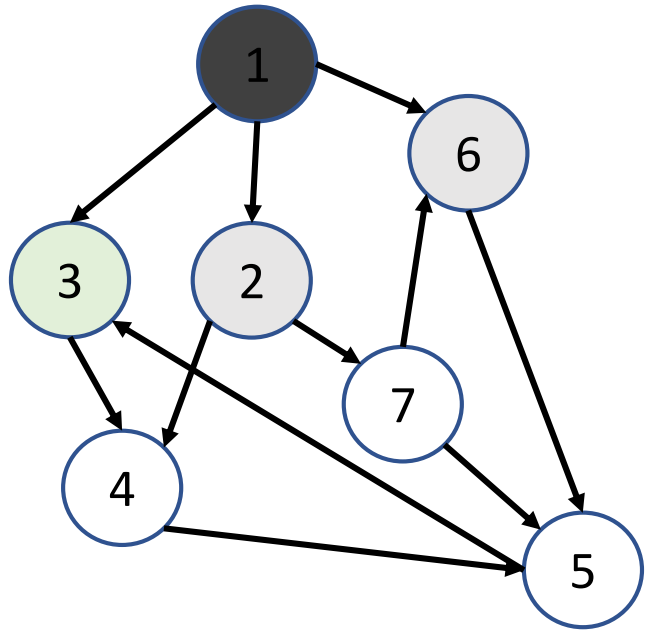
廣度優先搜尋（BFS）

- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



廣度優先搜尋（BFS）

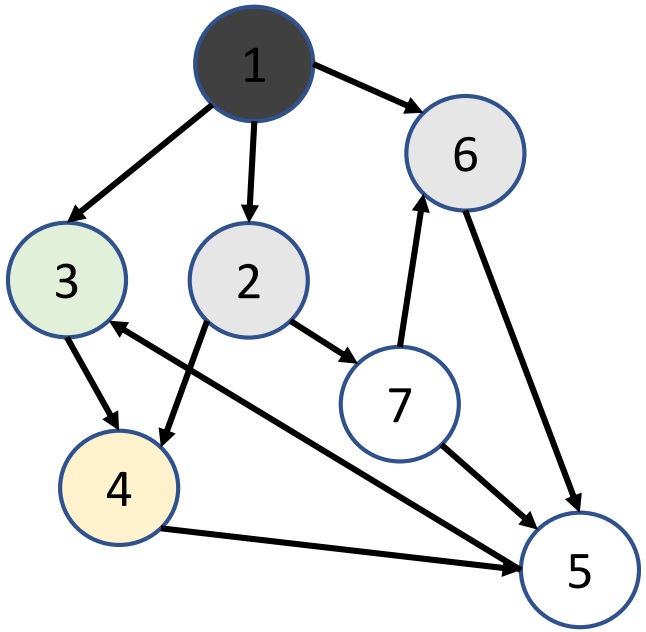
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6																

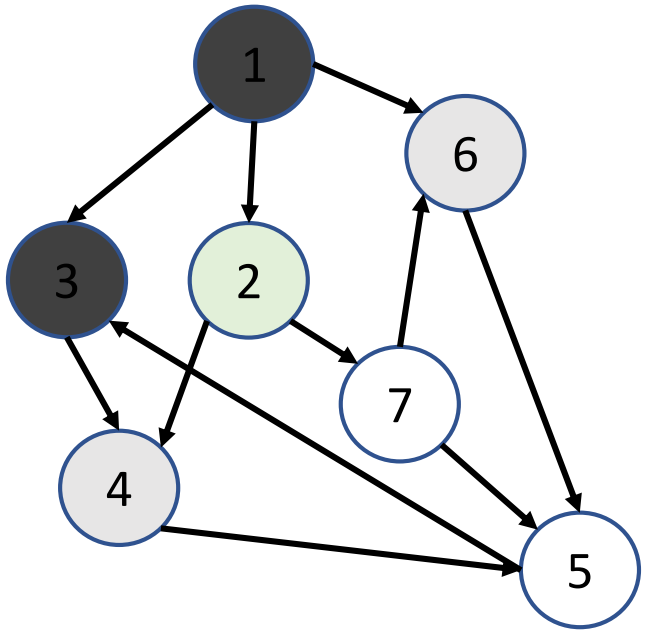
廣度優先搜尋（BFS）

- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



廣度優先搜尋（BFS）

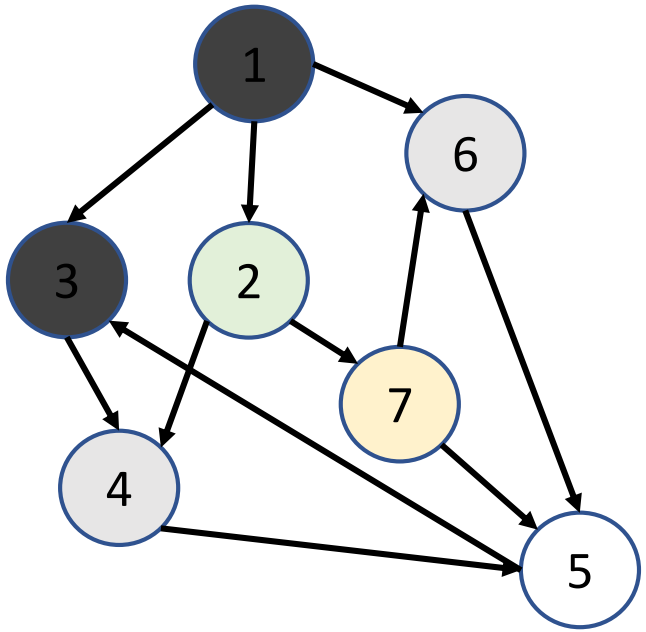
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6	4															

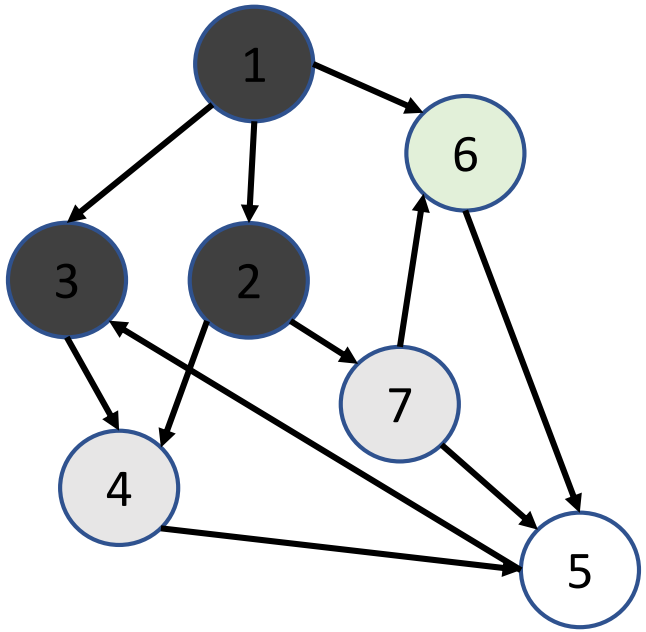
廣度優先搜尋（BFS）

- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



廣度優先搜尋（BFS）

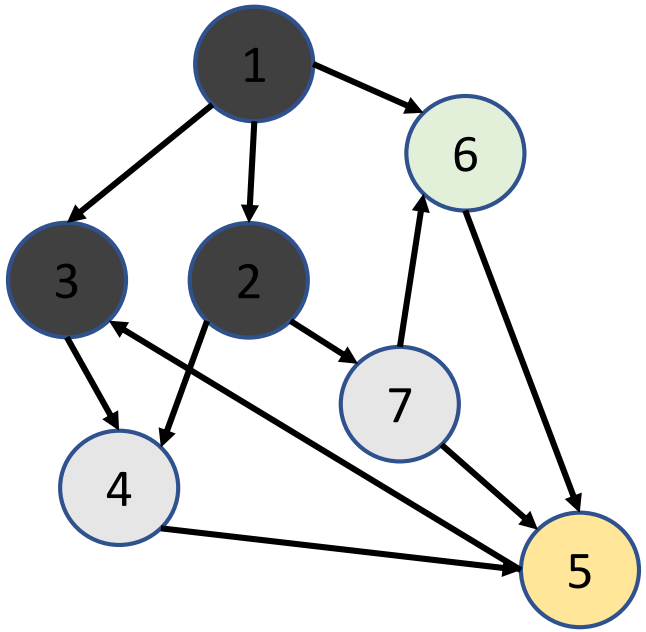
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6	4	7														

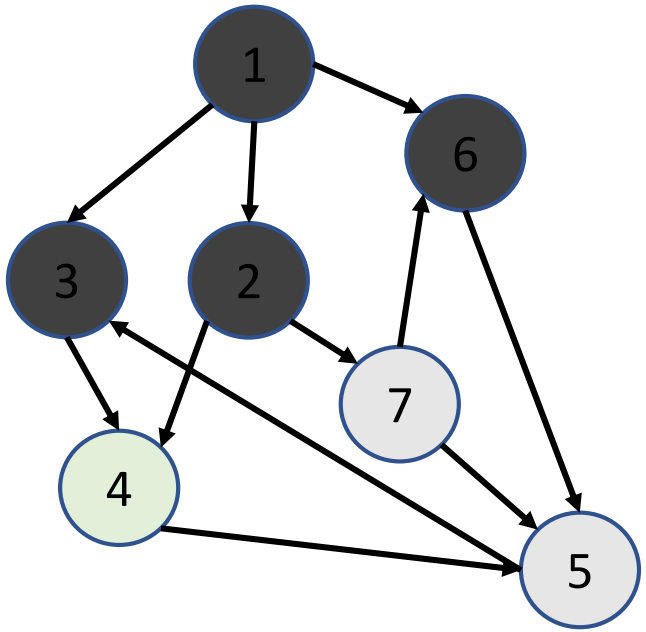
廣度優先搜尋（BFS）

- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



廣度優先搜尋（BFS）

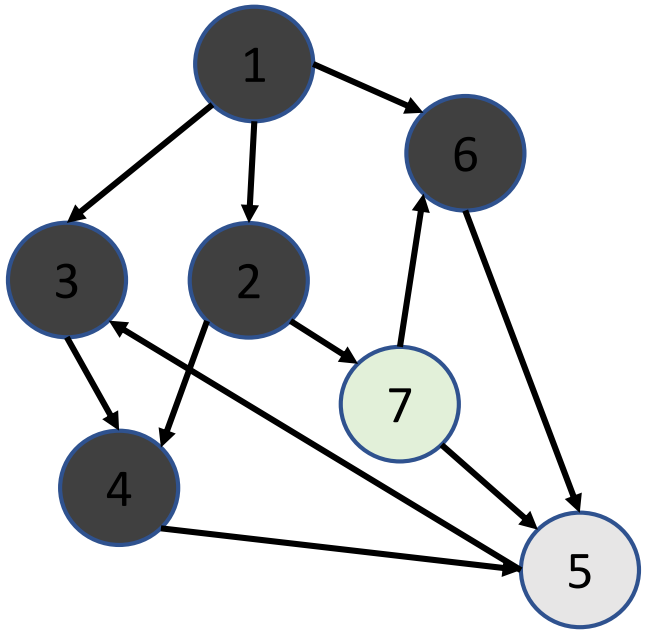
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6	4	7	5												

廣度優先搜尋（BFS）

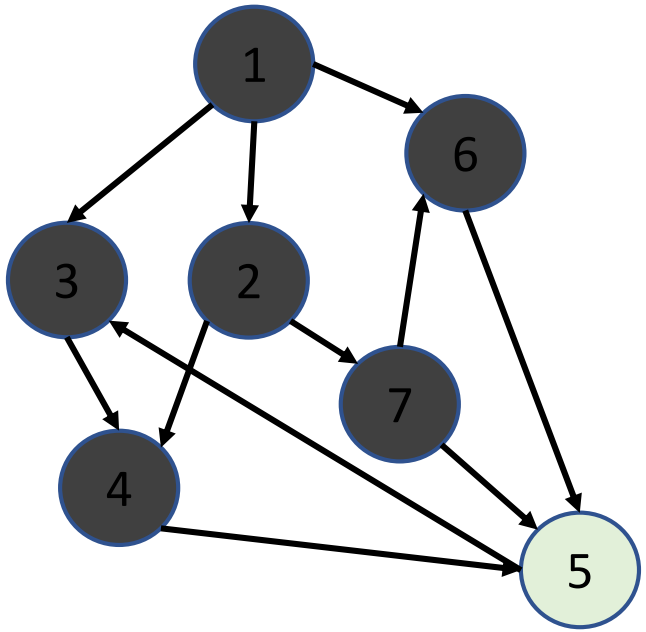
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6	4	7	5												

廣度優先搜尋（BFS）

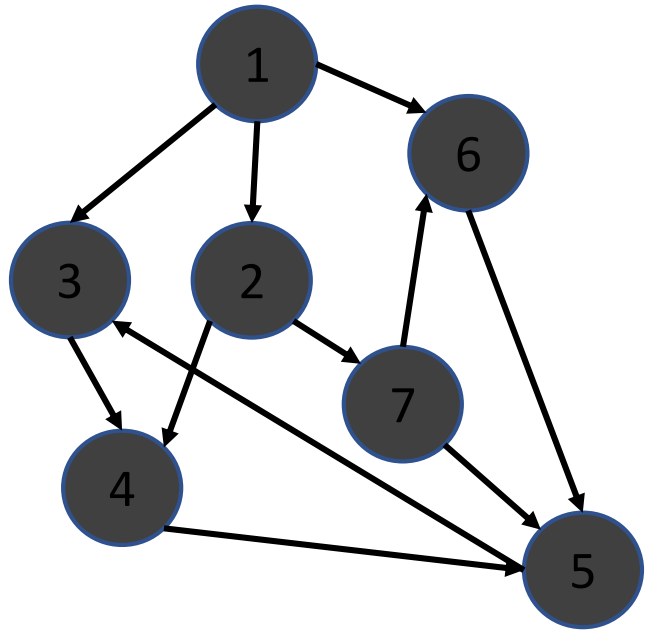
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



1	3	2	6	4	7	5												

廣度優先搜尋（BFS）

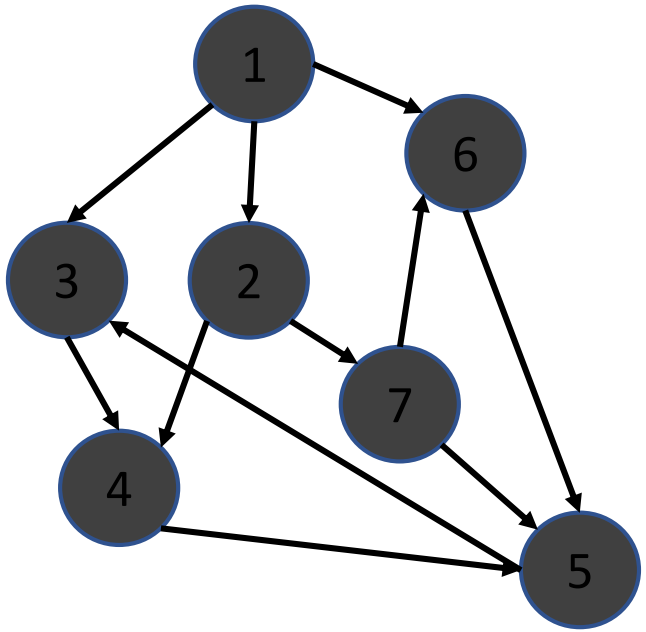
- 先尋訪完該節點所有的相鄰節點
- 尋訪完全部的相鄰節點後再擴散出去
- 通常使用 Queue 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過該節點，尚未尋訪所有相鄰節點
  3. Black：已尋訪過所有相鄰節點



1	3	2	6	4	7	5												

廣度優先搜尋（BFS）

- 結束條件
  - ➢ 當 Queue 裡的資料已經處理完畢
- 顏色代表的意義
  1. White：尚未被放入 Queue
  2. Gray：已被放入 Queue、尚未處理
  3. Black：已被放入 Queue、已處理



1	3	2	6	4	7	5												

廣度優先搜尋（BFS）

BFS(G,s)

```
1  for each vertex(v) in G:
2        color[v] = white
3  path_queue = {s}
4  color[s] = gray
5  while path_queue.size() != 0:
6        vertex_now = path_queue.pop()
7        for each vertex in vertex_now.adjacent():
8              if  color[vertex] == white:
9                    color[vertex] = gray
10                   path_queue.push(vertex)
11       color[vertex_now] = black
```

O(|V|)

O(|E|)

- 效能分析
  - 初始化：O(|V|)
  - 處理所有邊：O(|E|)
  - 總和：O(|V|+|E|)

Example Code

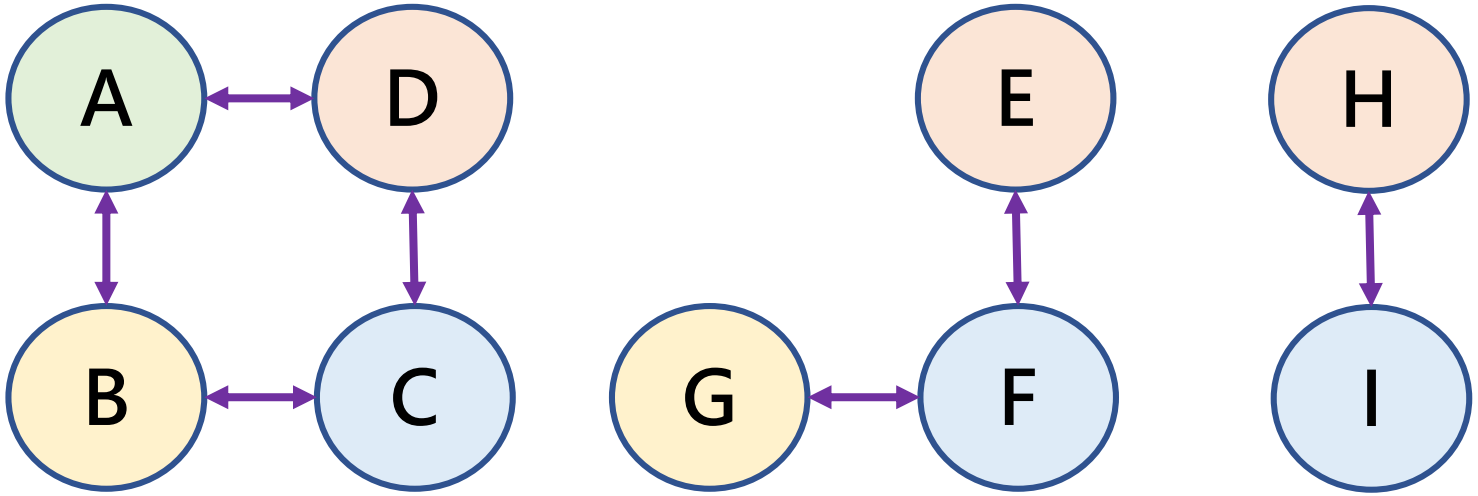
Mission

從給定的頂點與邊中，印出廣度優先搜尋的搜索過程。

Example Code

Mission

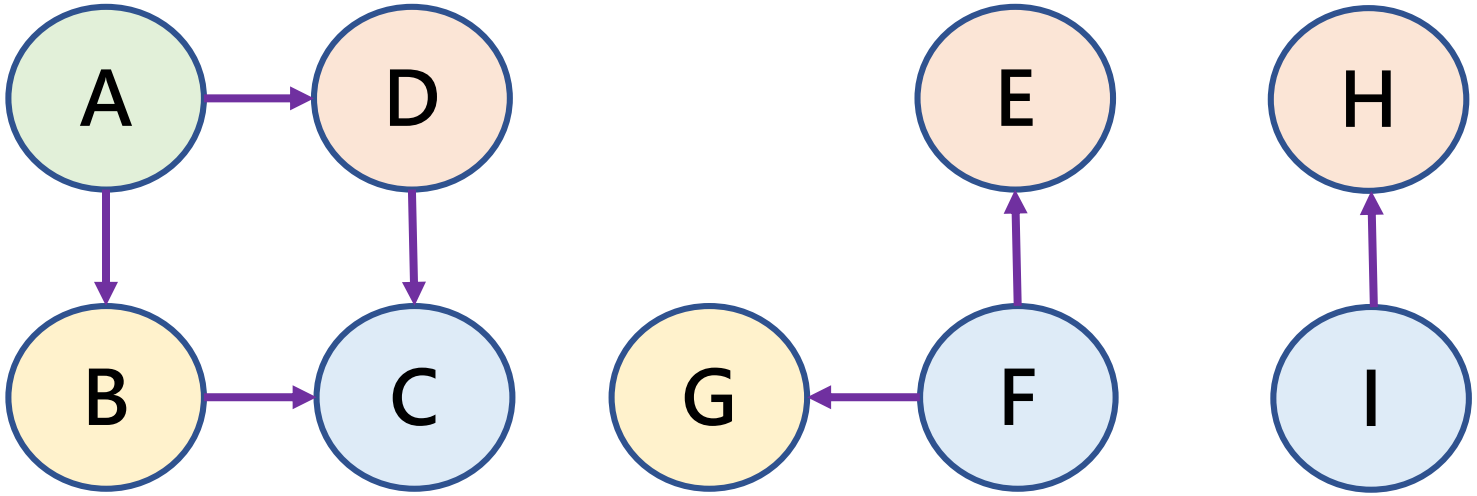
用剛剛寫的廣度優先搜尋計算連通的個數(無向圖)



Example Code

Mission

用剛剛寫的廣度優先搜尋計算弱連通的個數(有向圖)



Practice

Mission

Try LeetCode #733. Flood Fill

An image is represented by an m x n integer grid image where image[i][j] represents the pixel value of the image.

You are also given three integers sr, sc, and newColor. You should perform a flood fill on the image starting from the pixel image[sr][sc].

To perform a flood fill, consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color), and so on. Replace the color of all of the aforementioned pixels with newColor.

Return the modified image after performing the flood fill.

Ref：https://leetcode.com/problems/flood-fill/

Practice

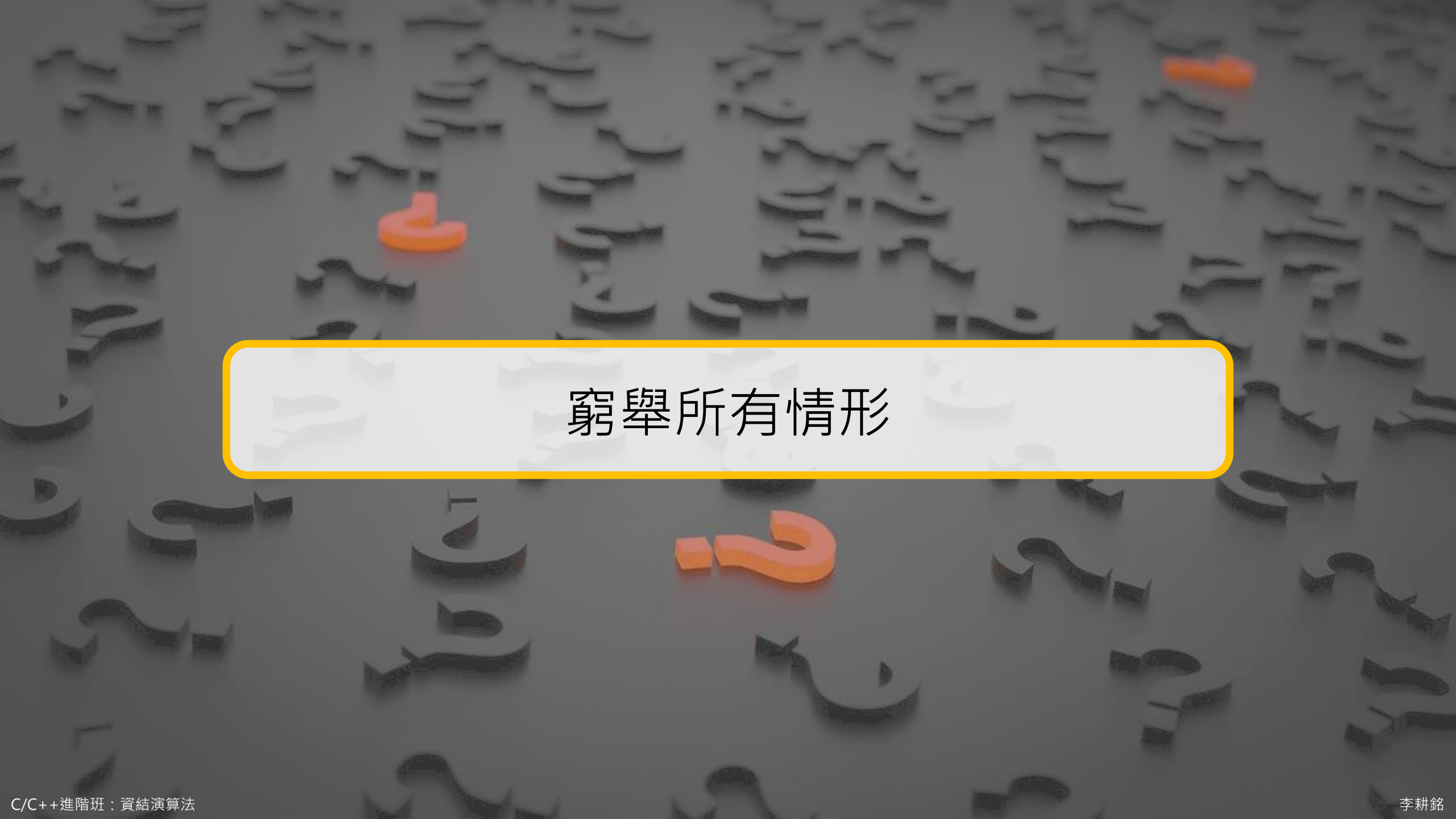
Mission

Try LeetCode #200. Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

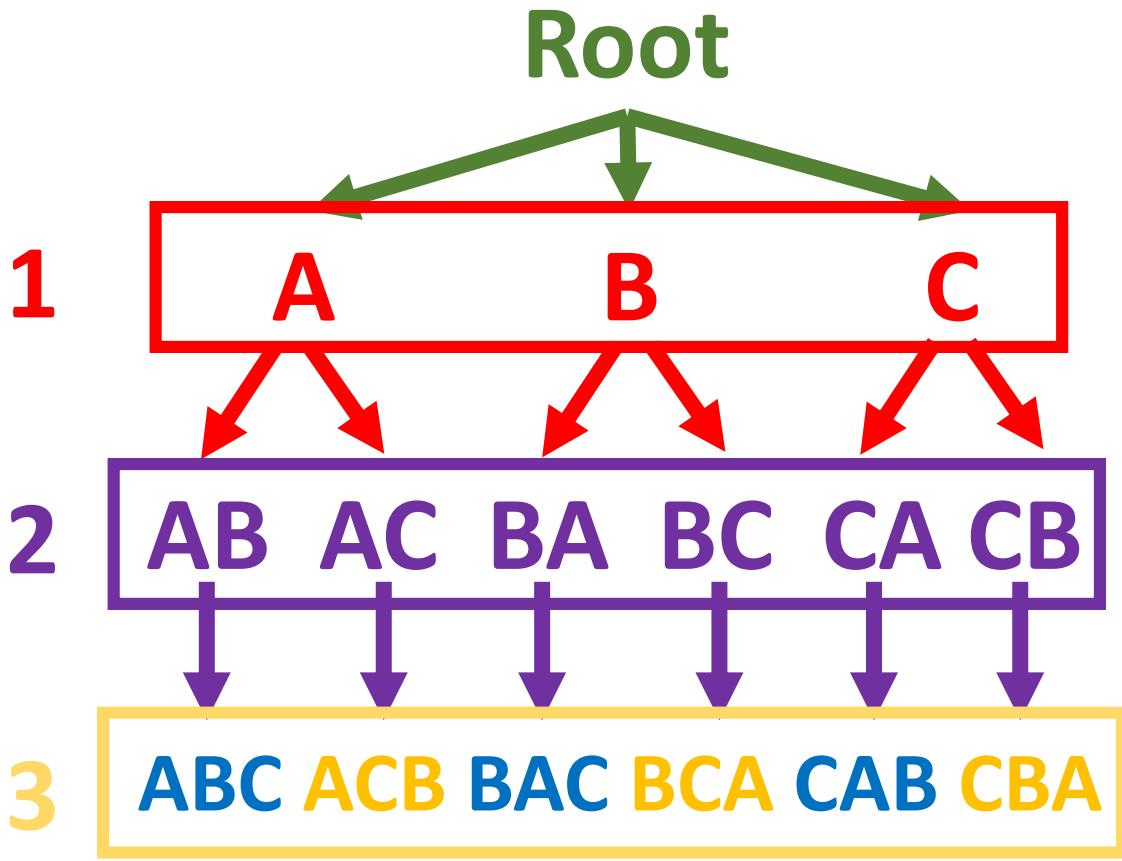
Ref：https://leetcode.com/problems/number-of-islands/



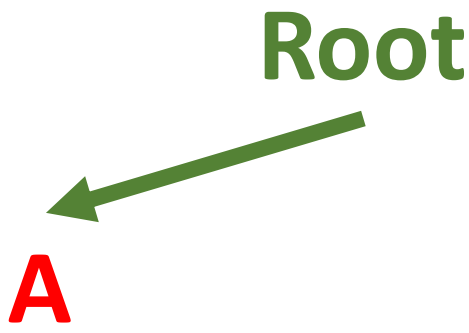
窮舉所有情形

窮舉所有情形

- 窮舉所有狀況
  - 把所有情形及可能發展畫成樹狀圖
  - 一層層依序把樹建立出來
  - 每一層建立後放到 Queue
  - 再依次把資料取出來後窮舉再置入
  - 直到 Queue 為空

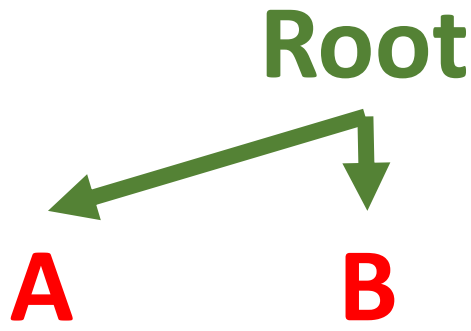


窮舉所有情形



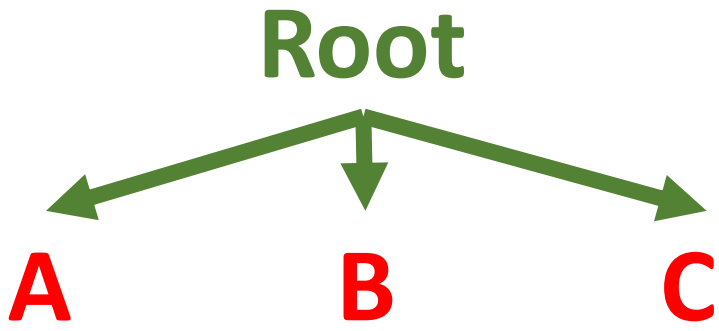
Queue

窮舉所有情形



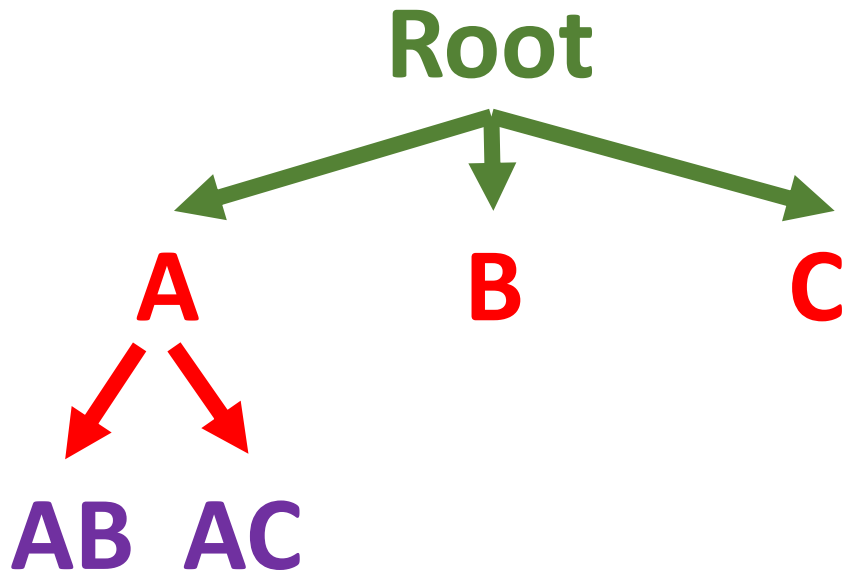
Queue

窮舉所有情形



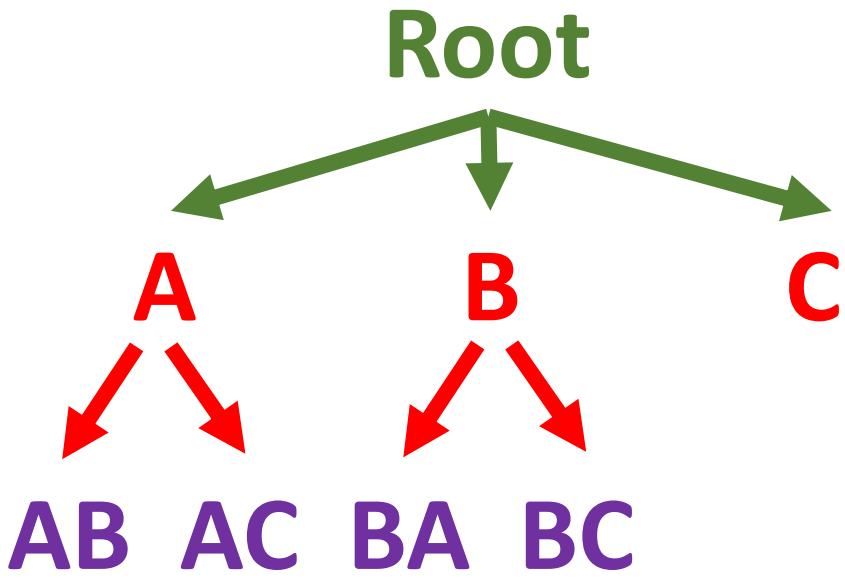
Queue

窮舉所有情形



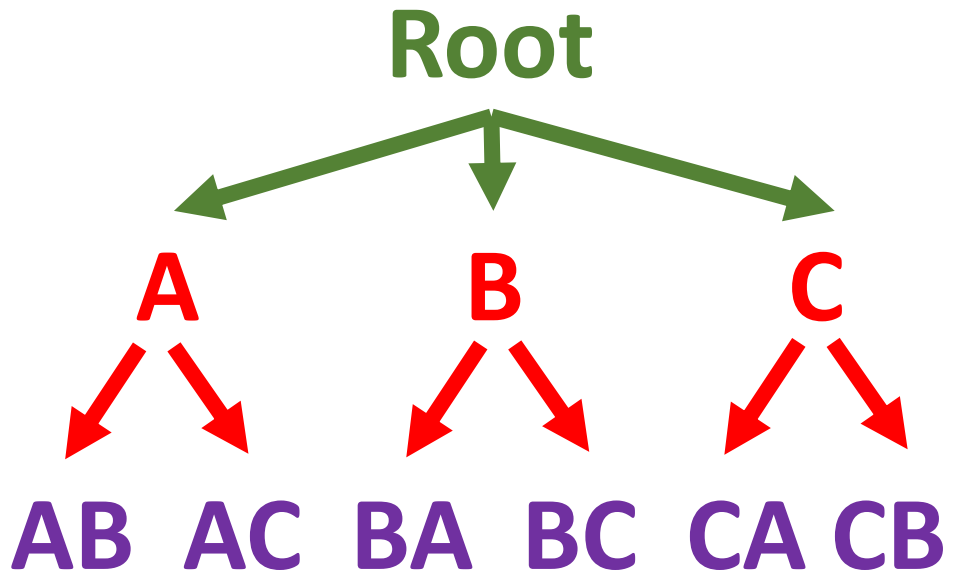
Queue

窮舉所有情形



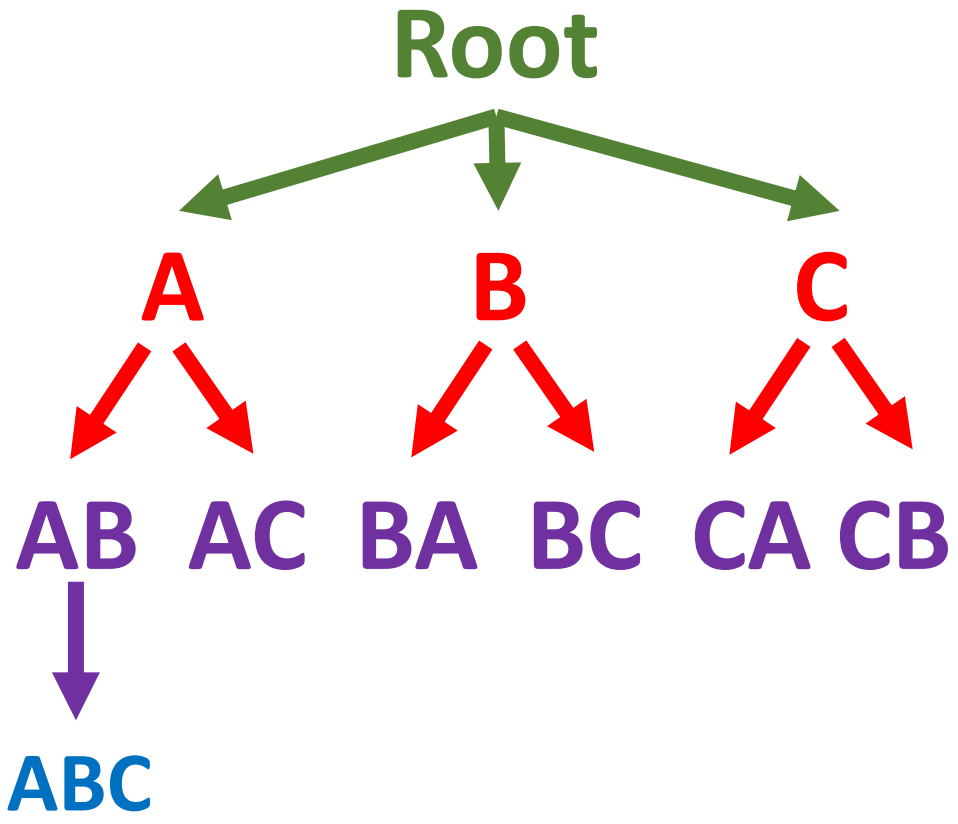
Queue

窮舉所有情形



Queue

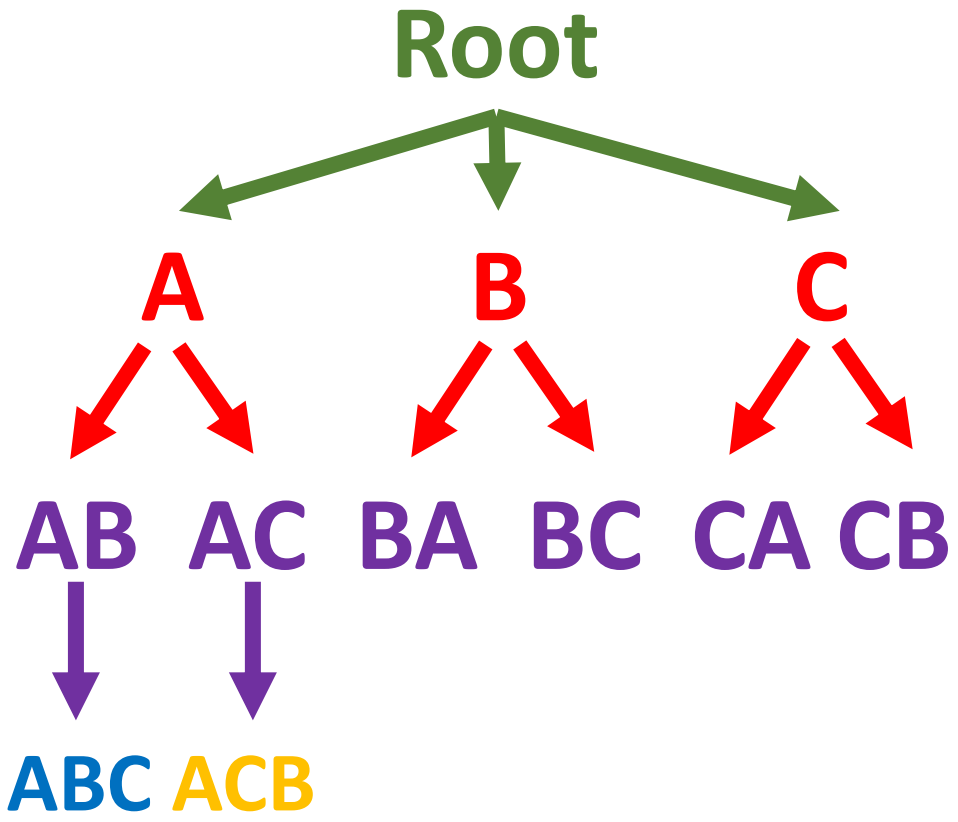
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC					

Queue

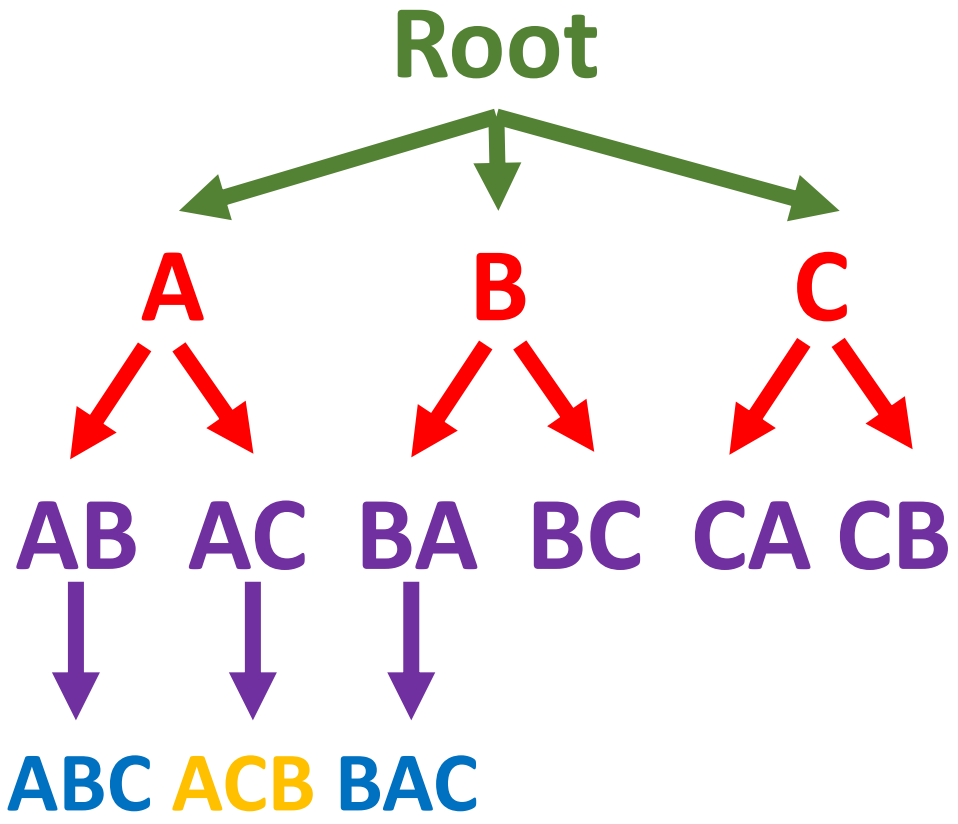
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB				

Queue

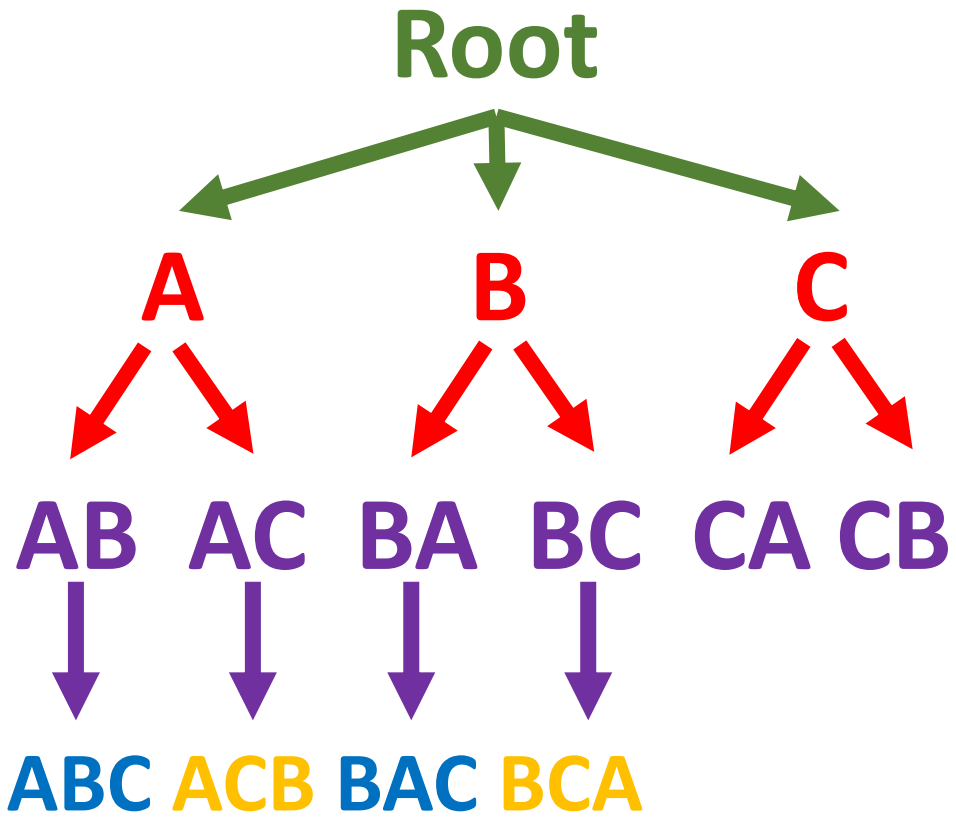
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB	BAC			

Queue

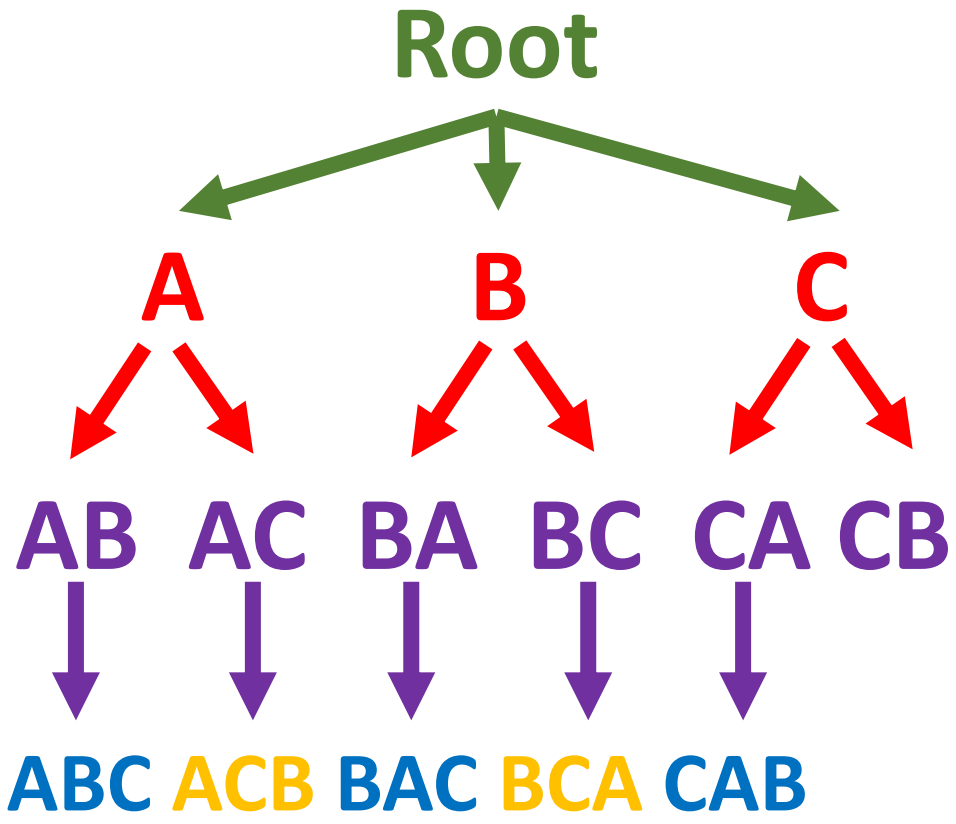
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB	BAC	BCA		

Queue

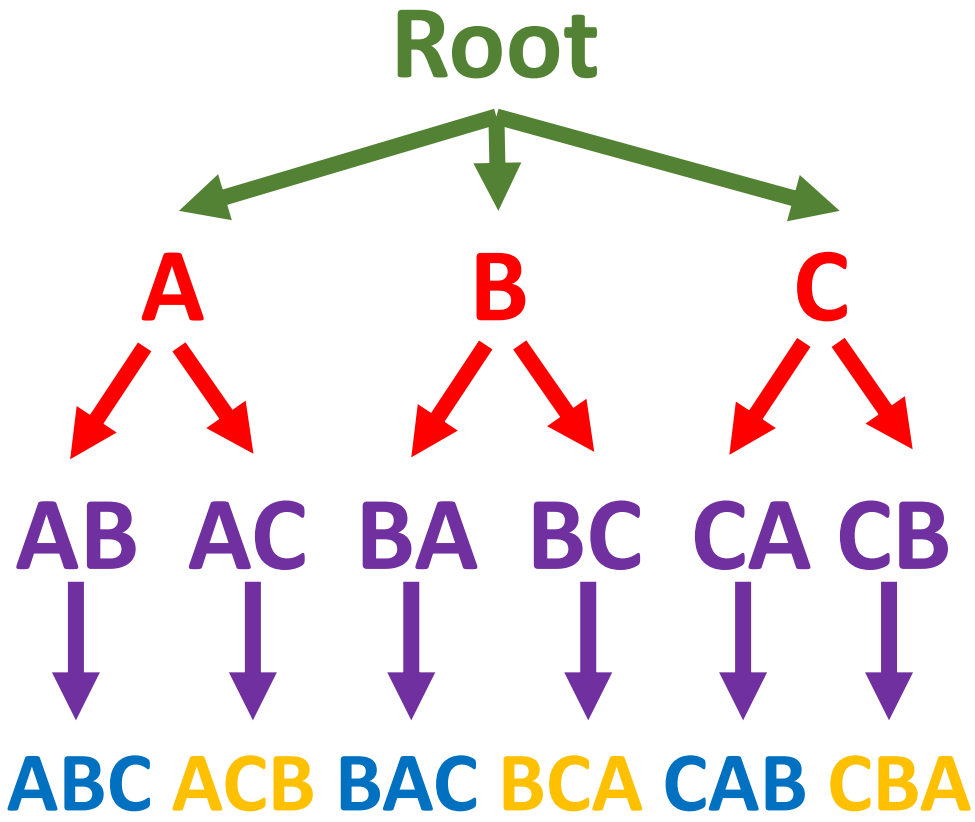
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB	BAC	BCA	CAB	

Queue

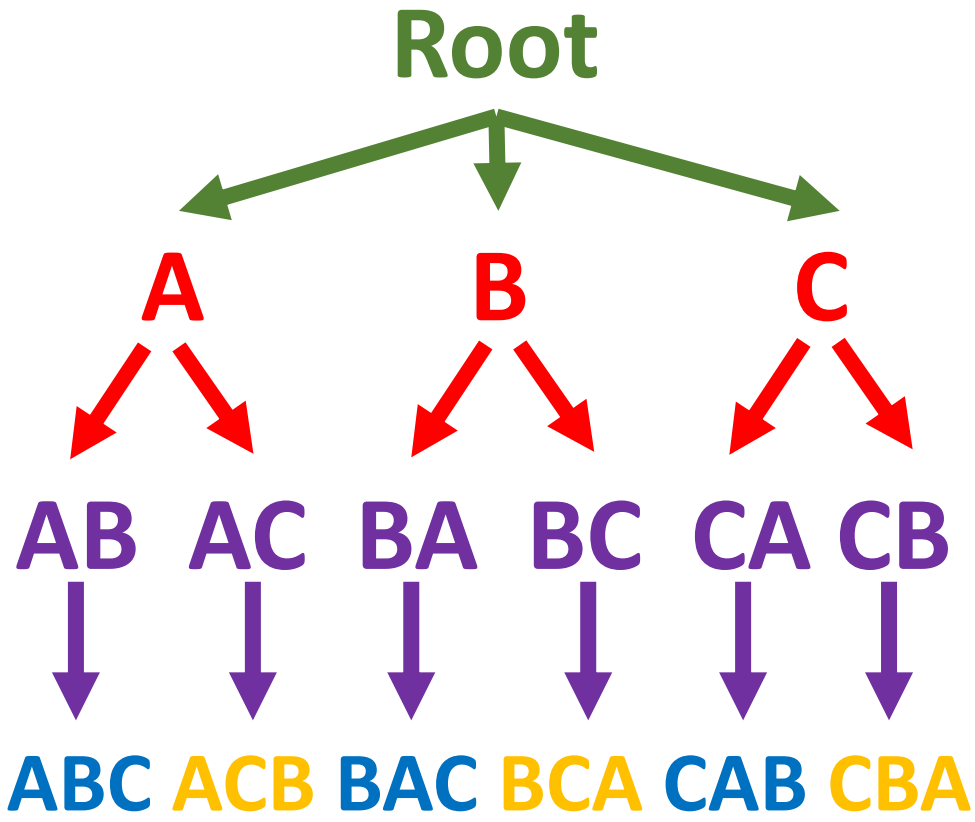
窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB	BAC	BCA	CAB	CBA

Queue

窮舉所有情形



A	B	C	AB	AC	BA	BC	CA	CB	ABC	ACB	BAC	BCA	CAB	CBA

Queue

Practice

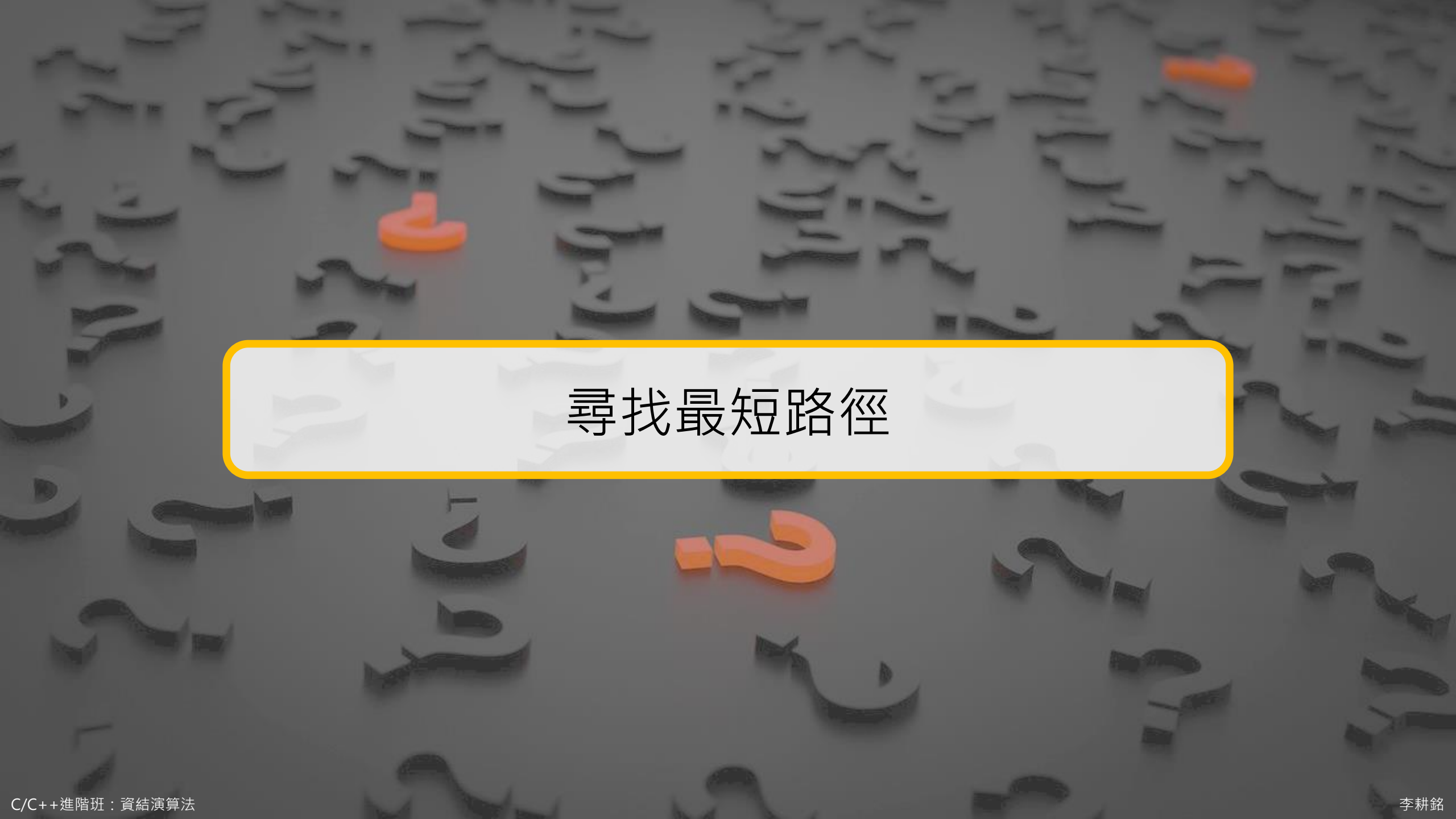
Mission

Try LeetCode #46. Permutations

Given an array nums of distinct integers, return all the possible permutations. You can return the answer in any order.

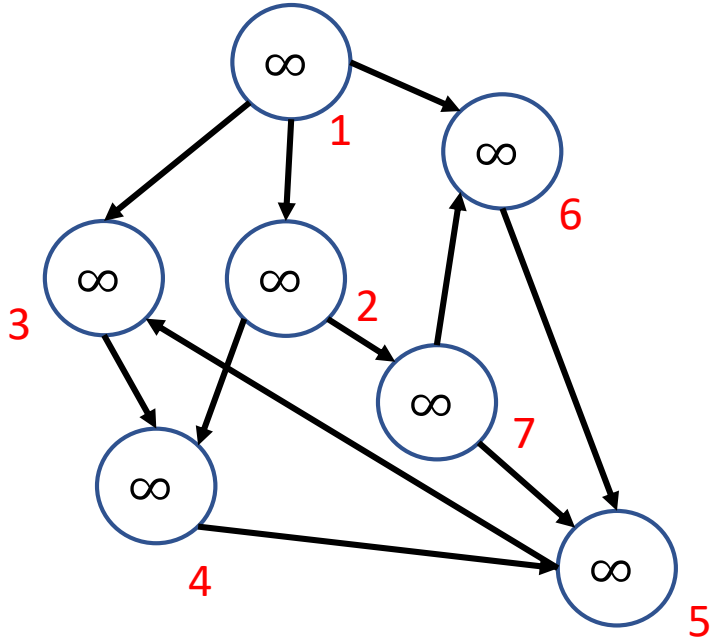
- Input: nums = [1,2,3]
- Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Ref：https://leetcode.com/problems/permutations/

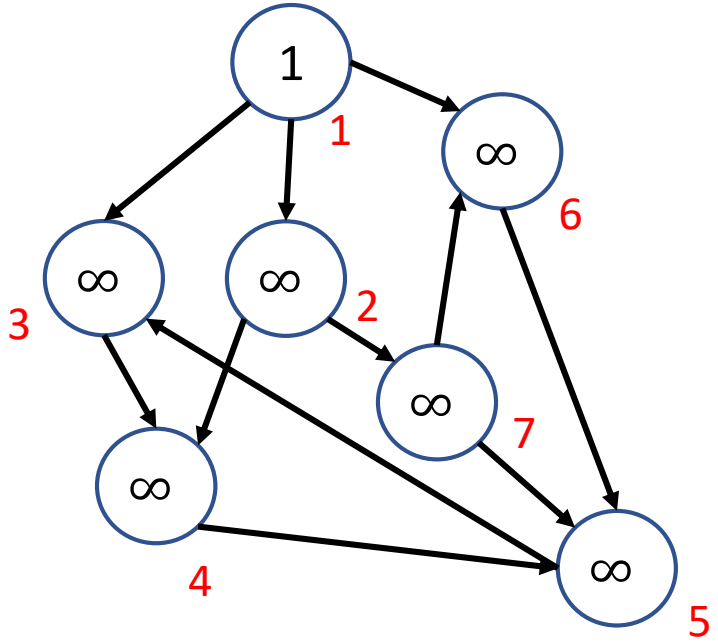


尋找最短路徑

尋找最短路徑

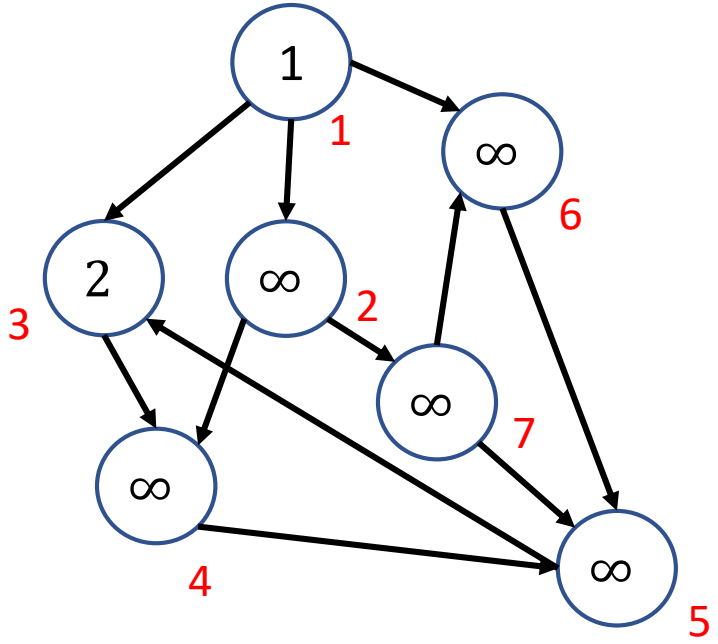


尋找最短路徑



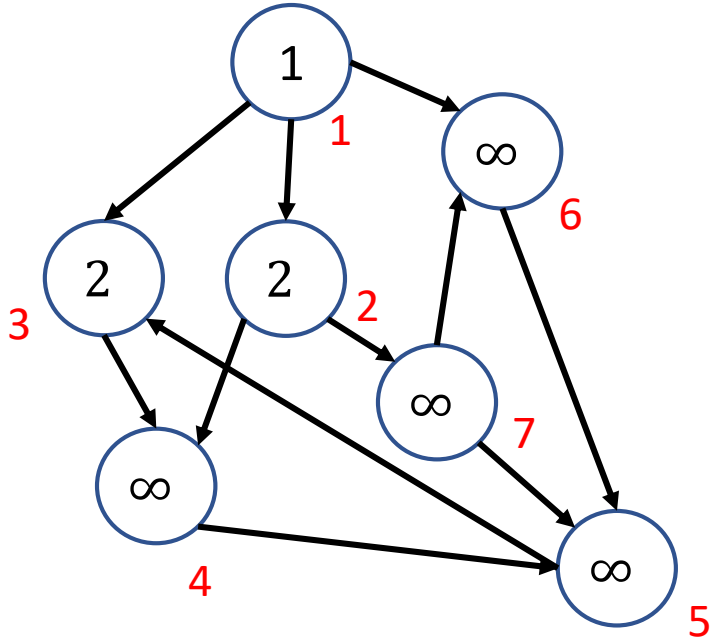
1																			

尋找最短路徑



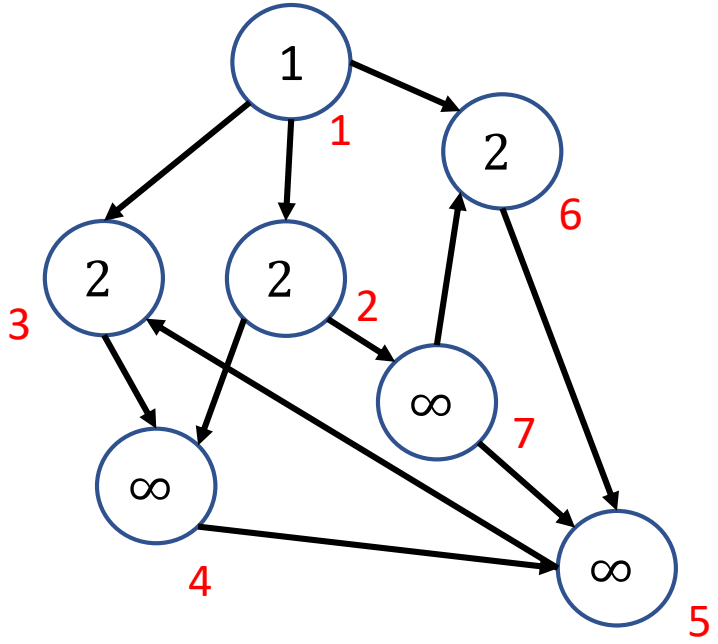
1	3																		

尋找最短路徑



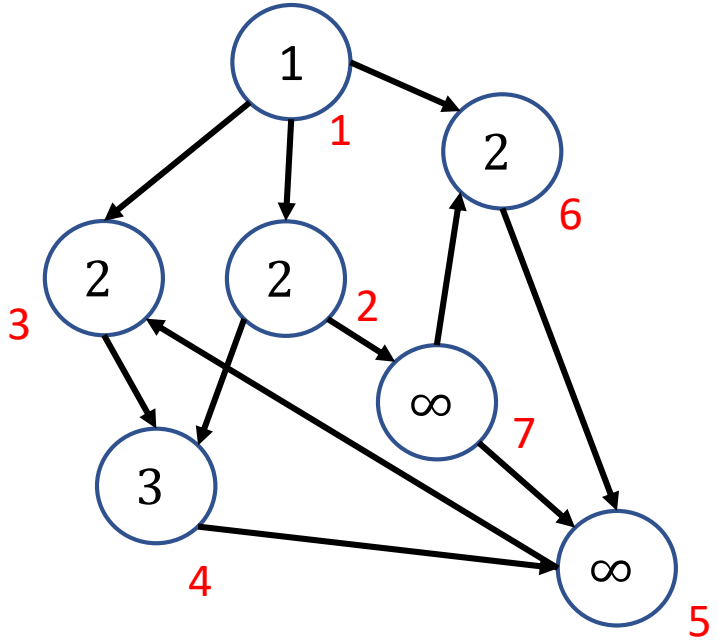
1	3	2																	

尋找最短路徑



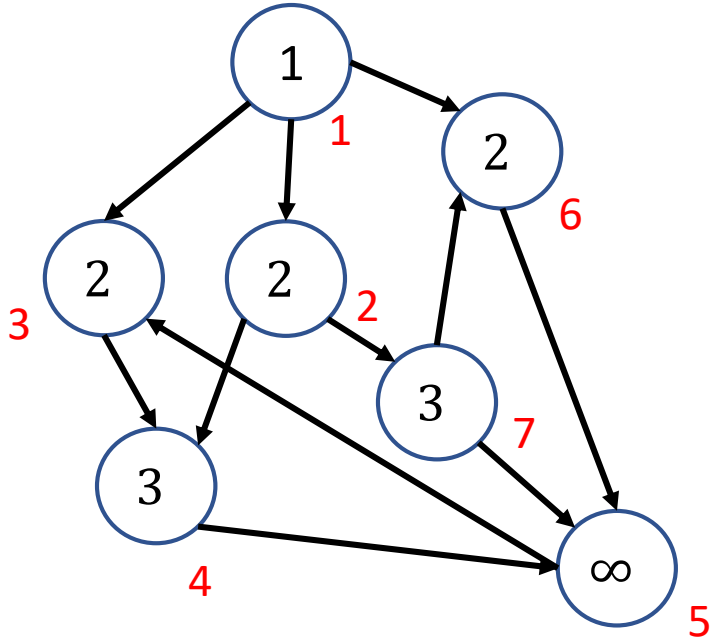
1	3	2	6															

尋找最短路徑



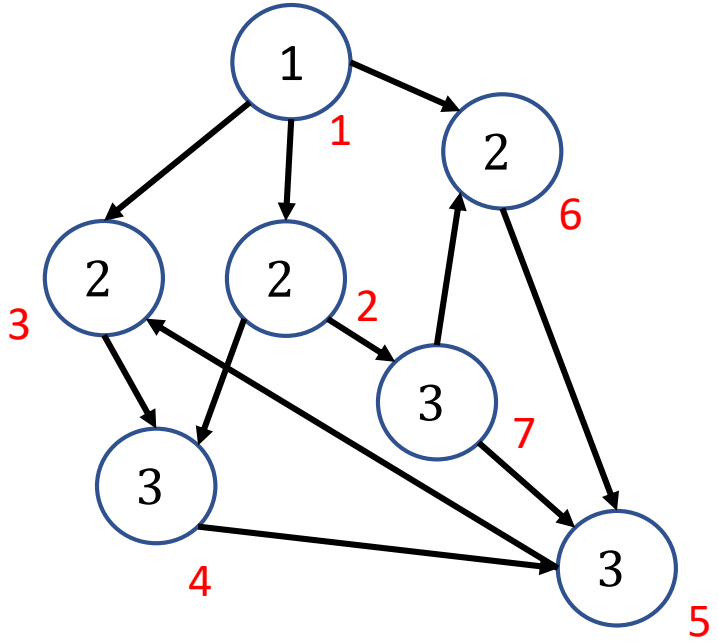
1	3	2	6	4														

尋找最短路徑



1	3	2	6	4	7													

尋找最短路徑



1	3	2	6	4	7	5												

尋找最短路徑

BFS(G,s)

```
1  for each vertex(v) in G:
2        color[v] = white
3  path_queue = {s}
4  color[s] = gray
5  while path_queue.size() != 0:
6        vertex_now = path_queue.pop()
7        for each vertex in vertex_now.adjacent():
8              if  color[vertex] == white:
9                    color[vertex] = gray
10                   path_queue.push(vertex)
11       color[vertex_now] = black
```

O(|V|)

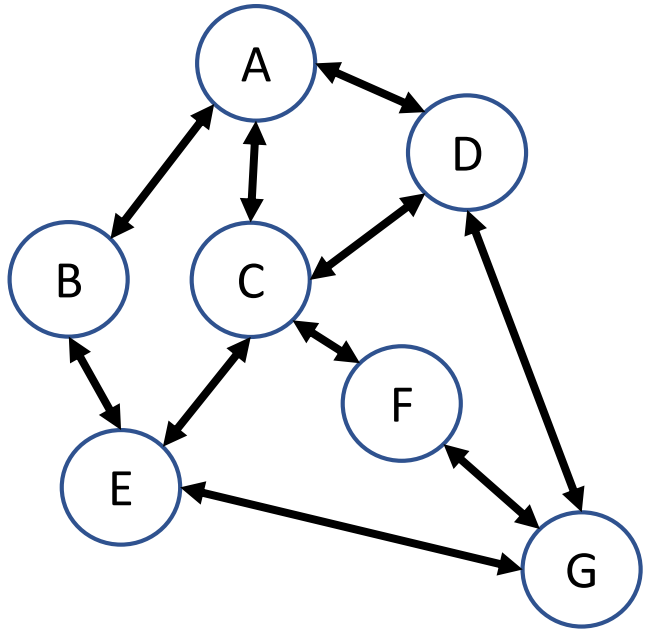
O(|E|)

- 效能分析
  - 初始化：O(|V|)
  - 處理所有邊：O(|E|)
  - 總和：O(|V|+|E|)

Practice

Mission

給定城市的數目與連接的道路，假設每一條道路所需要的交通時間相等，試著找出從 A 到其他城市間的最短路徑。(Unweighed)



Practice

Mission

Try LeetCode #1091. Shortest Path in Binary Matrix

In a town, there are n people labelled from 1 to n. There is a rumor that one of these people is secretly the town judge. If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

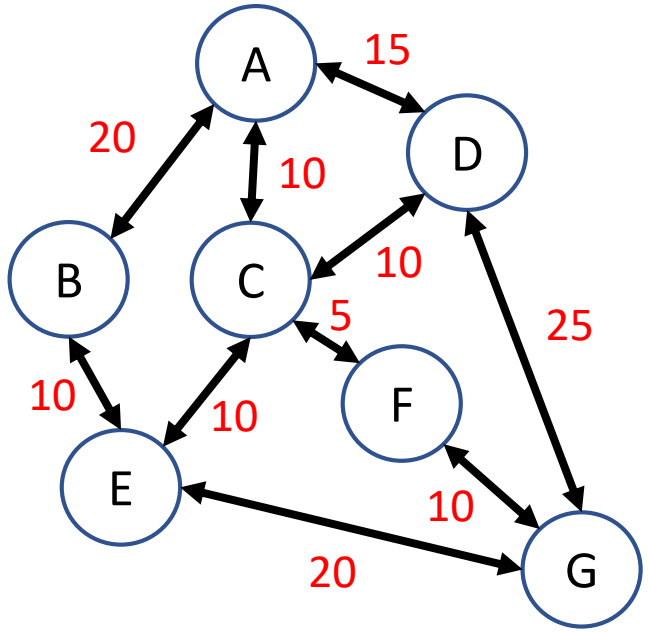
You are given trust, an array of pairs trust[i] = [a, b] representing that the person labelled a trusts the person labelled b. If the town judge exists and can be identified, return the label of the town judge. Otherwise, return -1.

Ref：https://leetcode.com/problems/shortest-path-in-binary-matrix/

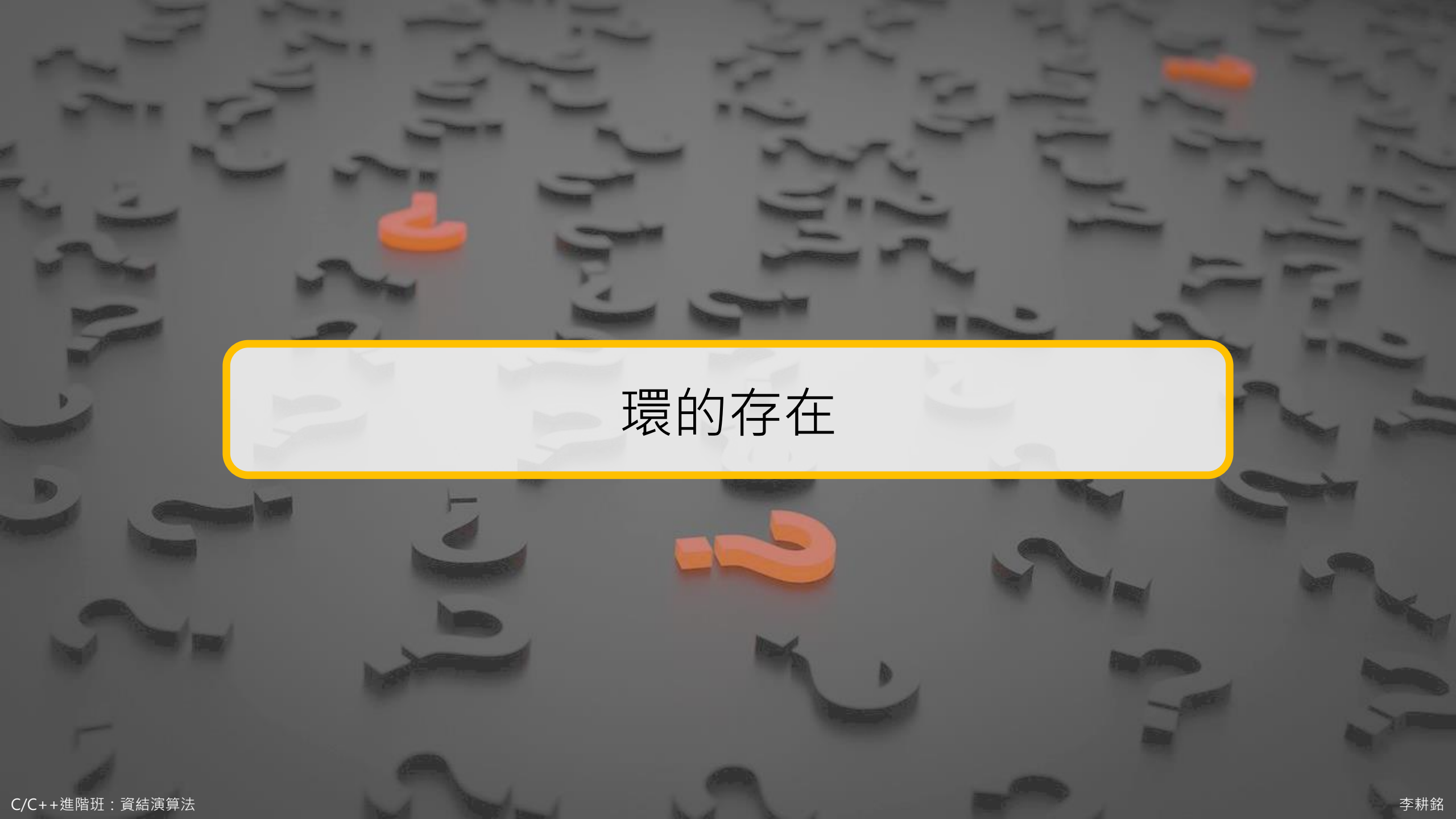
Practice

Mission

給定城市的數目與連接的道路，假設每一條道路所需要的交通時間不相等，試著找出從 A 到其他城市間的最短路徑。(Weighed)



Weighted graph 可以用 BFS 找嗎？



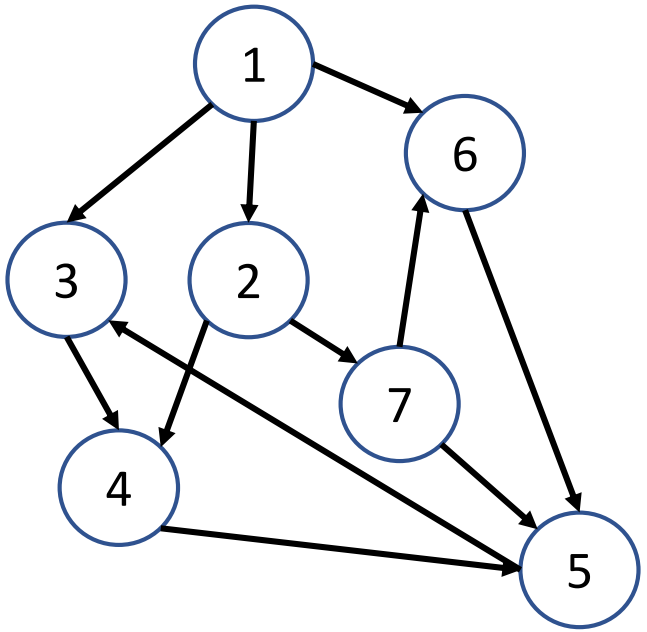
環的存在

環的存在

- 如何判別有沒有環存在？
  1. 無向圖 (Undirected Graph)

     將頂點放入 Queue 前，該頂點已存在 Queue 裏頭
  2. 有向圖 (Directed Graph)

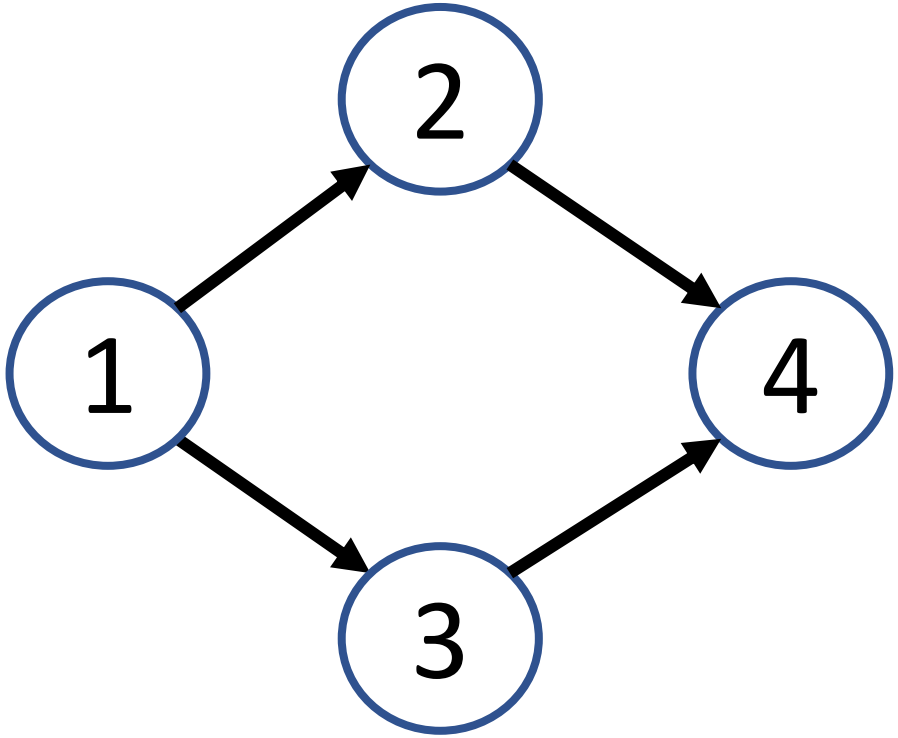
     計算 BFS 過程中的 in-degree 的頂點個數 (麻煩)



為什麼有向圖不能透過該點是否尋訪過來確認環？

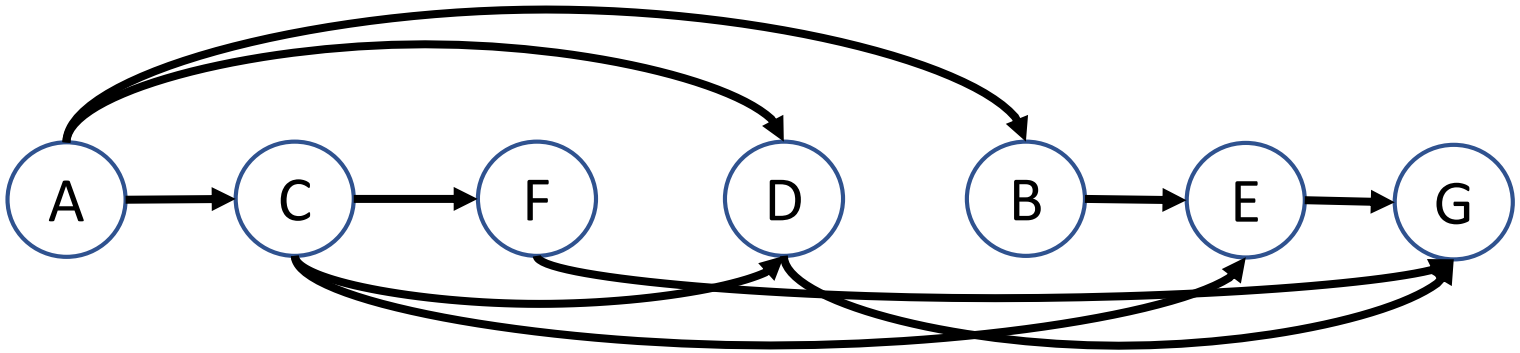
環的存在

- 利用 BFS 確認環
  - Queue：1
  - Queue：~~1~~23
  - Queue：~~12~~34
  - Queue：~~123~~44
  - 4出現過，但沒有環！
  - 有向圖不能透過該點是否尋訪過來確認環



環的存在

- 原理：有向無環圖 (Directed Acyclic Graph) 裡的拓樸排序
  - 若有邊 e(u,v)，則 u 必在 v 之前
- 從入度 (in-degree) 為 0 的節點 (u) 出發拜訪所有相鄰節點
  - 入度為 0 的點不可能形成環，是安全的
  - 一開始沒有入度為 0 節點的話，必有環
- 把該節點 (u) 的相鄰節點入度 -1，因從該方向過來的邊不會形成環
- 若入度降為 0，則再把該節點的相鄰節點入度 -1，不斷重複該步驟
- 看最後所有節點的入度是否都能降成 0



環的存在

- 利用 BFS 確認環的存在
  - 檢查每個節點的出度與入度
  - 原理：Kahn’s algorithm for Topological Sorting
  - 步驟
    1. 計算每個節點的入度 (in-degree)
    2. 已拜訪的節點數目初始化為 0
    3. 把入度為 0 的節點加入 Queue
    4. pop 出一個節點，把已拜訪的節點數目 +1
    5. 把該節點所有的 neighboring node 的入度 -1
    6. 如果入度被降成 0，把該點加入 Queue
    7. 重複步驟 4~6，直到 Queue 為空
    8. 如果已拜訪節點數=總節點數，無環，反之無環

環的存在

BFS_Cycle(G,s)

```
1  for each e(u,v) in G:
2      in_degree[v] += 1
3  for each (vertex,value) in indegree:
4      if in_degree[vertex] == 0
5          queue.push(vertex)
6  visited_node = 0
7  while(queue is not empty):
8      current node = queue.pop()
9      for each vertex in current node.adjacent():
10         in_degree[vertex] -= 1
11         if in_degree[vertex] == 0
12             queue.push(vertex)
13     visited_node += 1
14 if visited_node == |V|:  return false
15 else: return true
```

O(|E|+|V|)

O(|E|+|V|)

- 效能分析
  - 初始化：O(|E|+|V|)
  - 處理所有頂點：O(|V|)
  - 處理所有邊：O(|E|)
  - 總和：O(|E|+|V|)

很麻煩喔！
確認是否有環通常不會用 BFS
改用下一章會講到的 DFS

Practice

Mission

Try LeetCode #207. Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Ref：https://leetcode.com/problems/course-schedule/

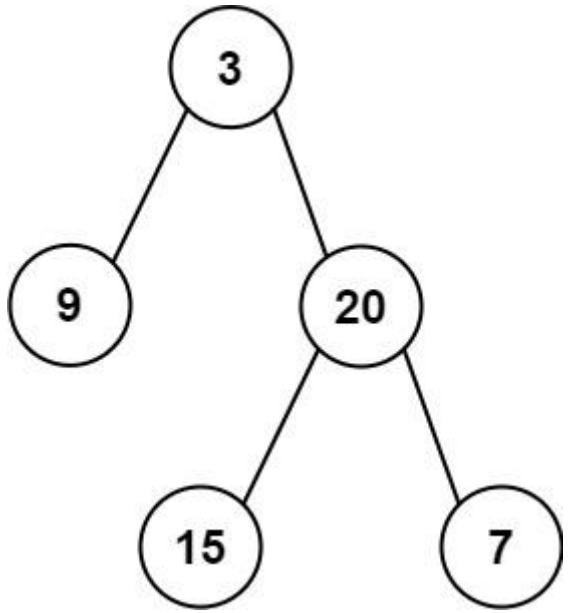
實戰練習

Practice

Mission

Try LeetCode #404. Sum of Left Leaves

Given the root of a binary tree, return the sum of all left leaves.



Ref：https://leetcode.com/problems/sum-of-left-leaves/

Practice

Mission

Try LeetCode #1020. Number of Enclaves

You are given an m x n binary matrix grid, where 0 represents a sea cell and 1 represents a land cell.

A move consists of walking from one land cell to another adjacent (4-directionally) land cell or walking off the boundary of the grid.

Return the number of land cells in grid for which we cannot walk off the boundary of the grid in any number of moves.

Ref : https://leetcode.com/problems/number-of-enclaves/