C/C++ 進階班

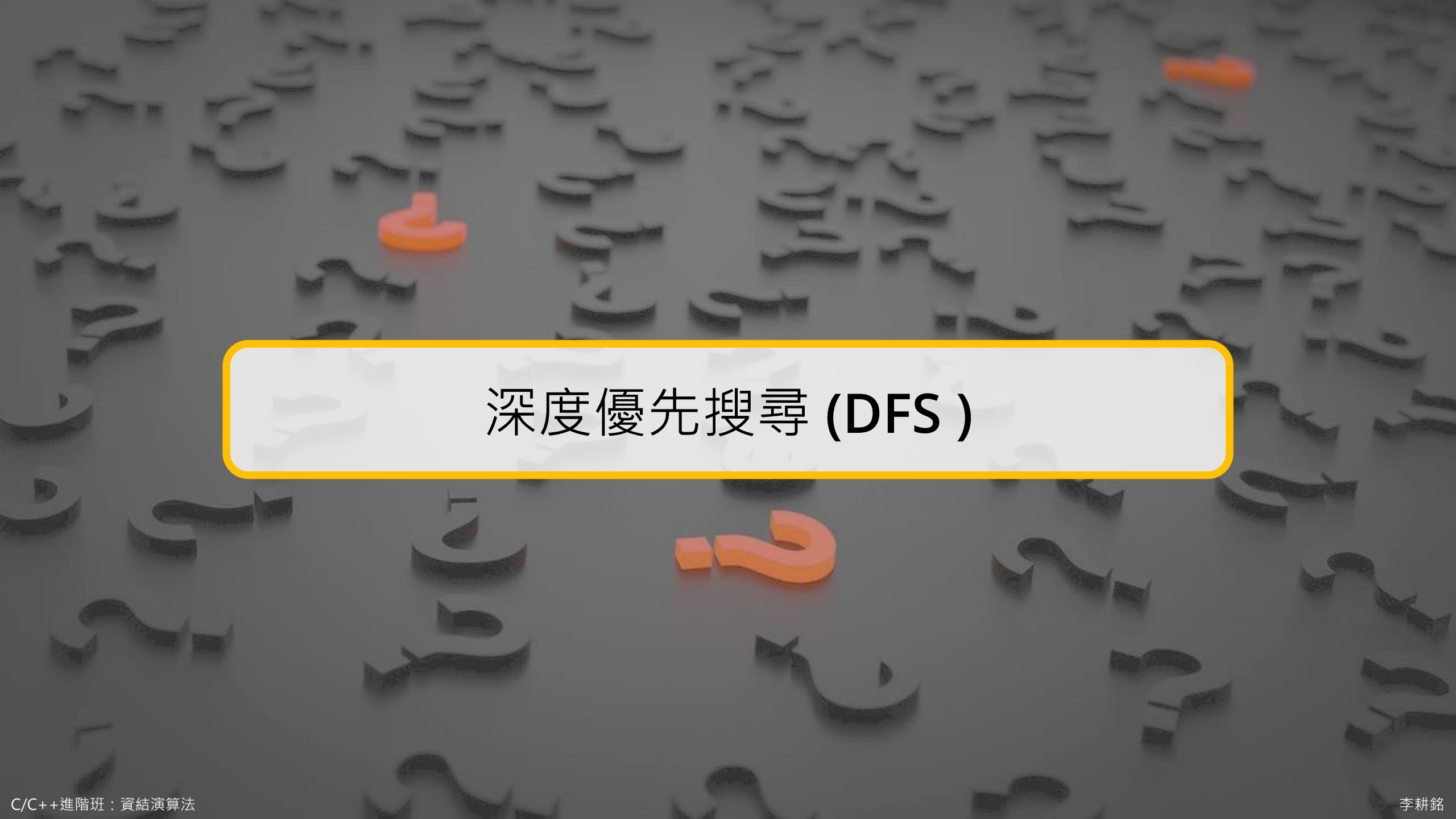
演算法

深度優先搜尋
(Depth-First Search)

李耕銘

課程大綱

- 深度優先搜尋 (DFS)
  - 環的確認
  - 二分圖的判別
  - 拓樸排序
  - 找強連通元件
  - 八皇后問題
- BFS 與 DFS 比較
- 實戰練習

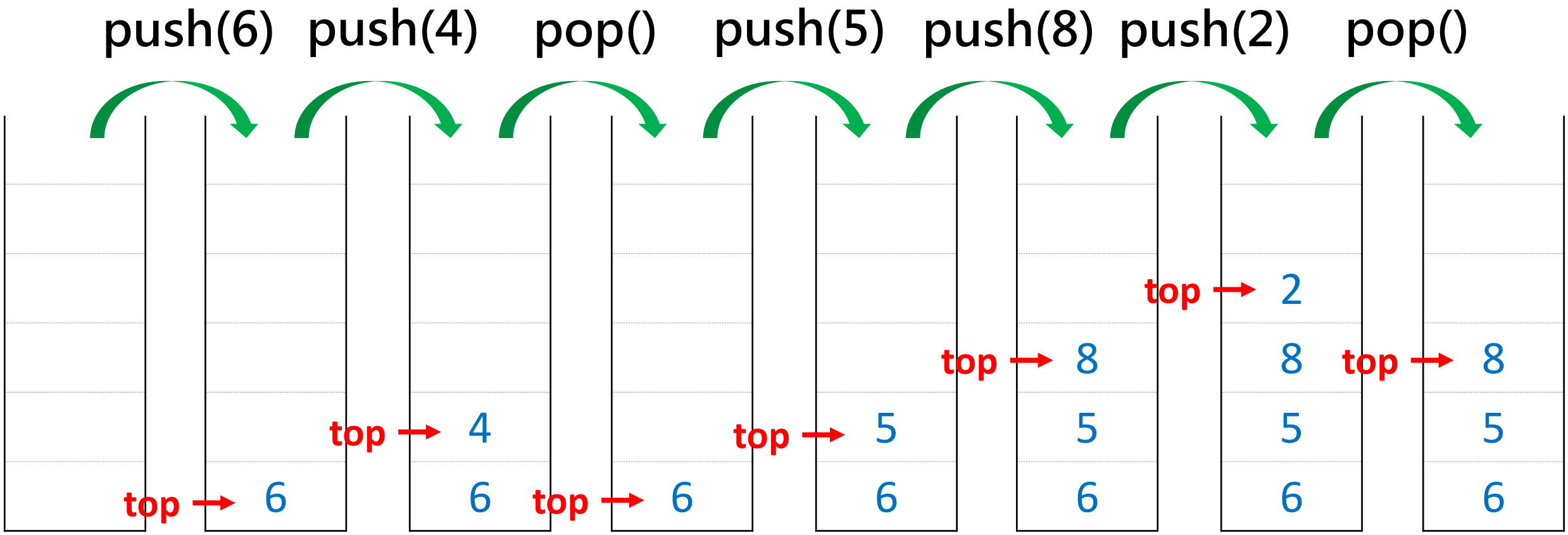


深度優先搜尋 (DFS)

堆疊(Stack)

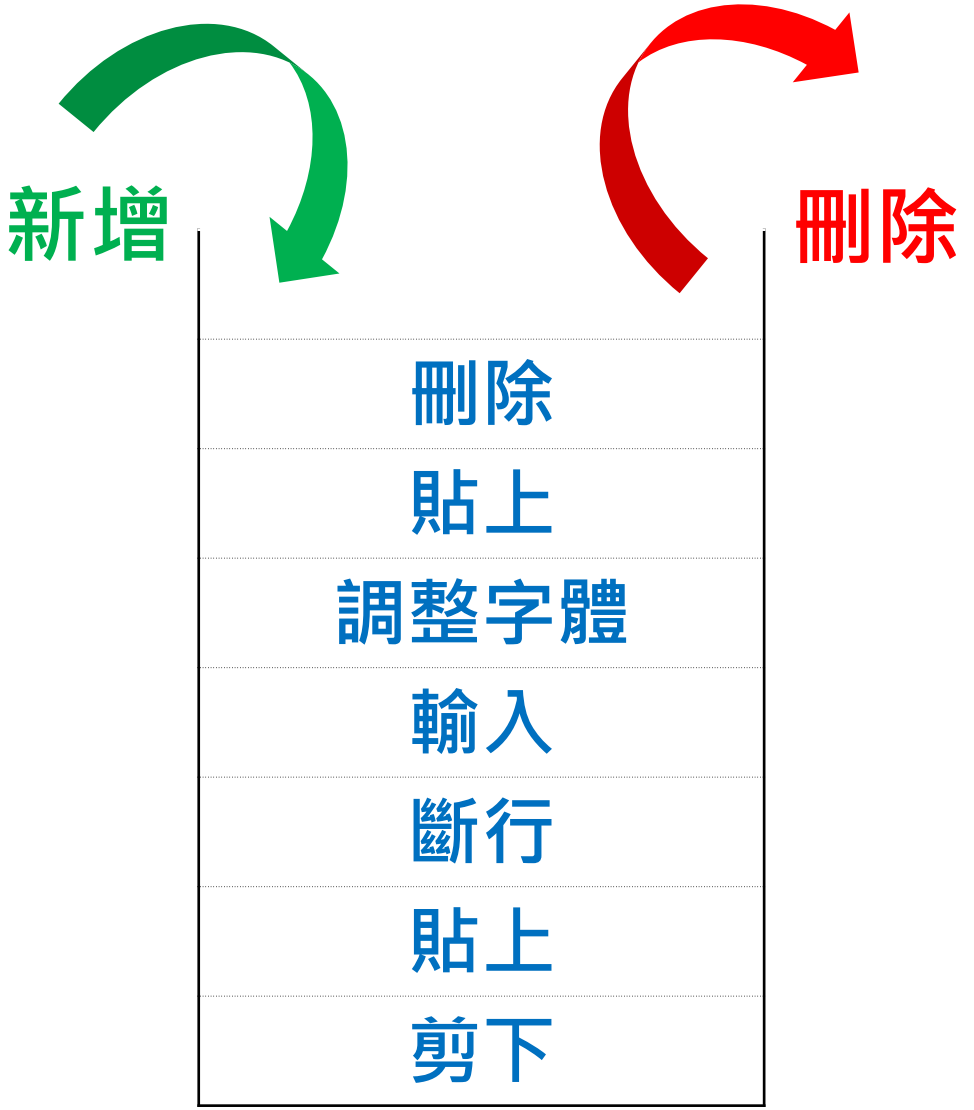
- 堆疊(Stack)
  - 插入、刪除在同側
  - last-in-first-out(LIFO)
- 常見的操作
  - push：新增一筆資料
  - pop：刪除一筆資料
  - top：回傳最末端的資料
  - empty：確認stack裡是否有資料
  - size：回傳 stack 的資料個數

堆疊(Stack)


push(6)
push(4)
pop()
push(5)
push(8)
push(2)
pop()
top
6
top
4
6
top
6
top
5
6
top
8
5
6
top
2
8
5
6
top
8
5
6

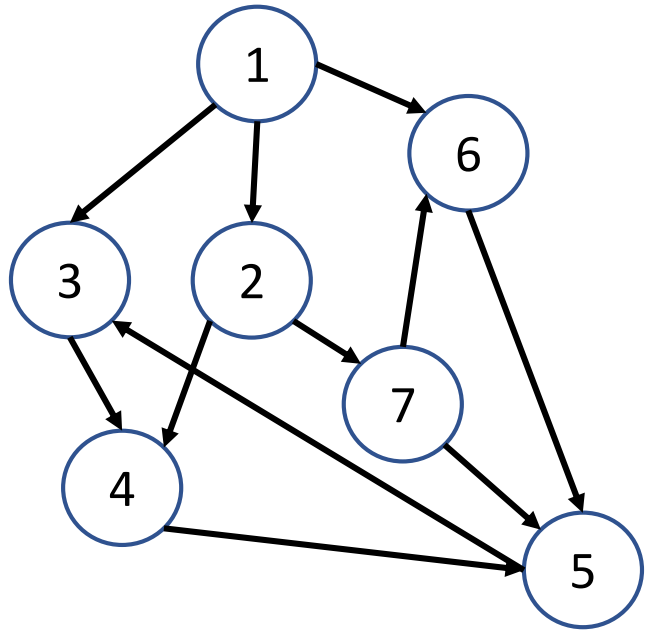
堆疊(Stack)的用途

- 依序紀錄先前的資訊
  - 常用來回復到先前的狀態
    - 瀏覽器回到上一頁
    - 編輯器復原
    - 編譯器的解析(parse)
    - 函式呼叫(遞迴)
- 迷宮探索、河內塔、發牌
  - Depth-First Search (演算法)
- 無法得知 stack 裡有哪些資料
  - 只能以 pop() 一個個把資料拿出來



深度優先搜尋（DFS）

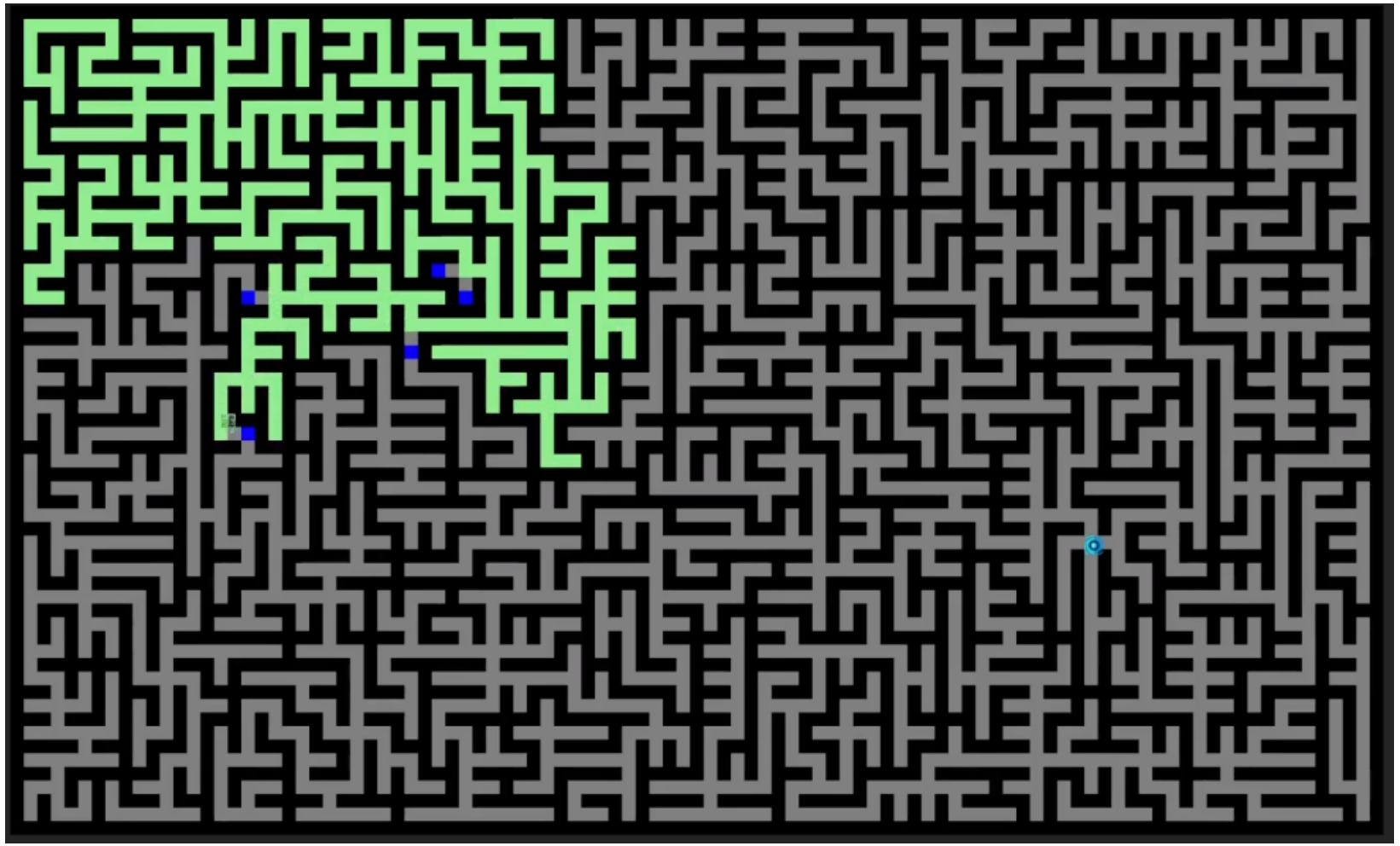
- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 來完成
- 避免陷入無窮迴圈
  1. White：尚未尋訪過
  2. Gray：已尋訪過，尚未處理
  3. Black：已尋訪過，已處理



Path Stack

Vertex

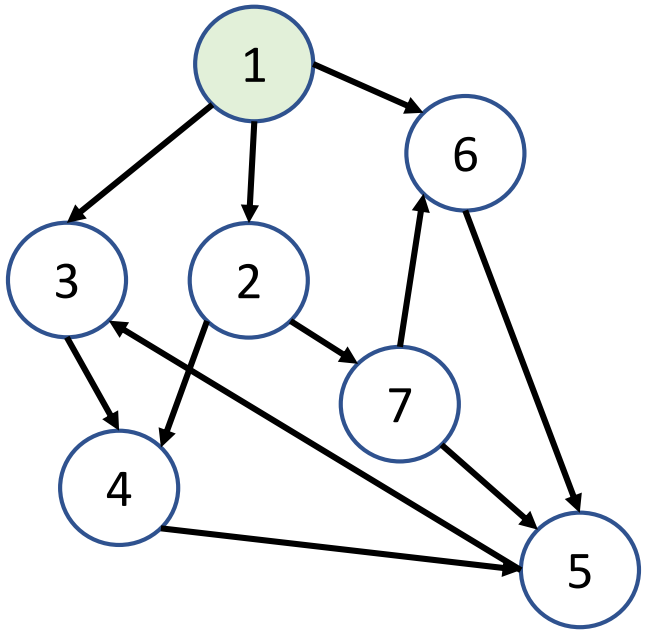
圖的搜尋



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

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

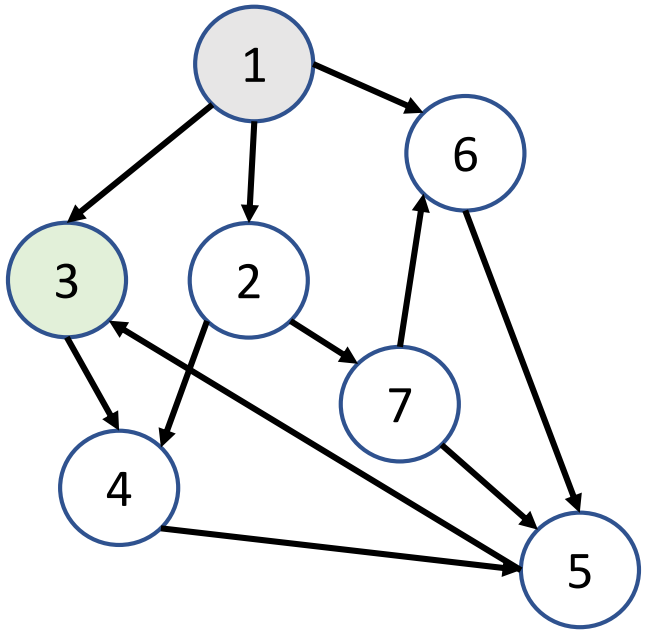
1																		

Vertex

1																		

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

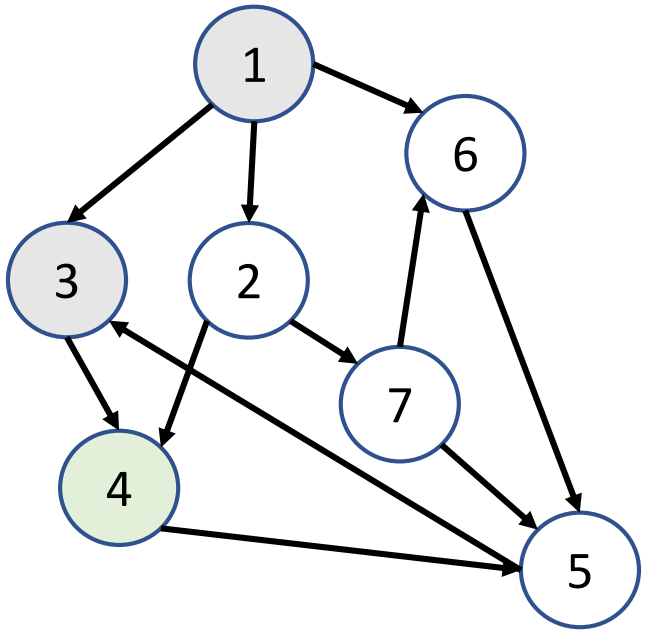
1	3																	

Vertex

1	3																	

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

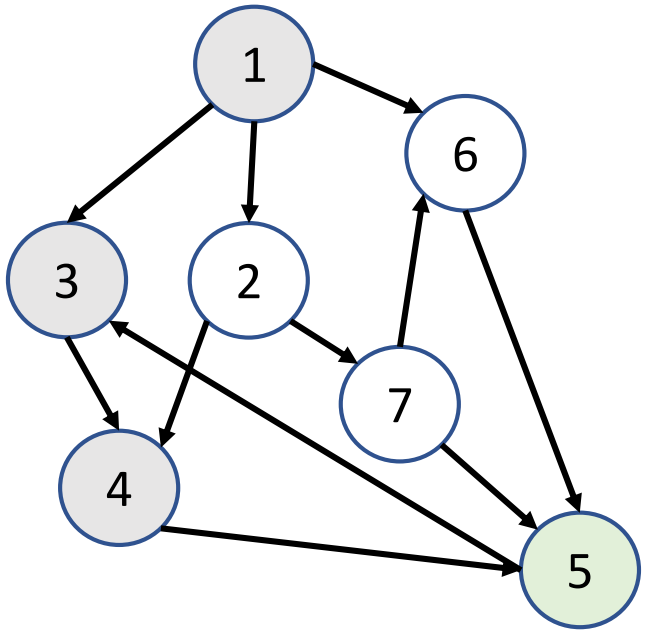
1	3	4																	

Vertex

1	3	4																	

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

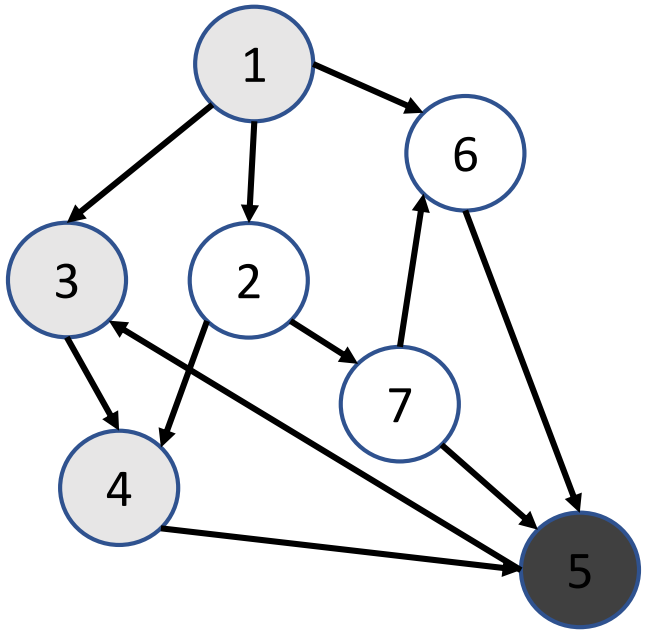
1	3	4	5															

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

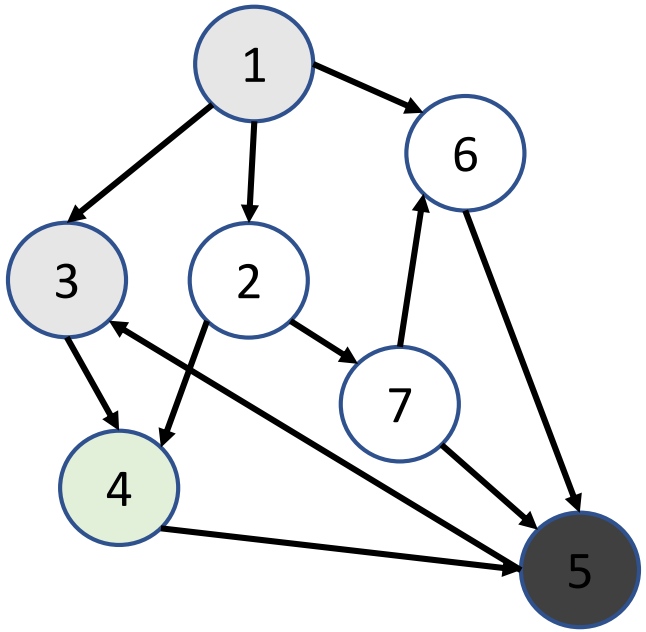
1	3	4																

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

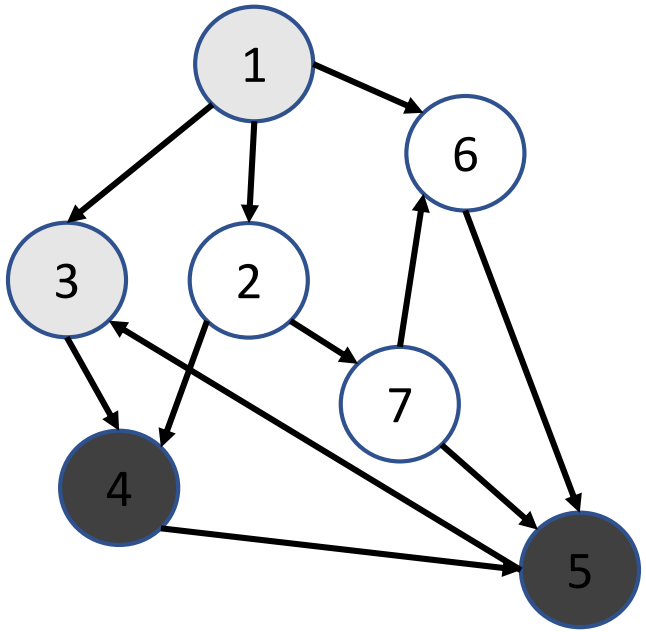
1	3	4																

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

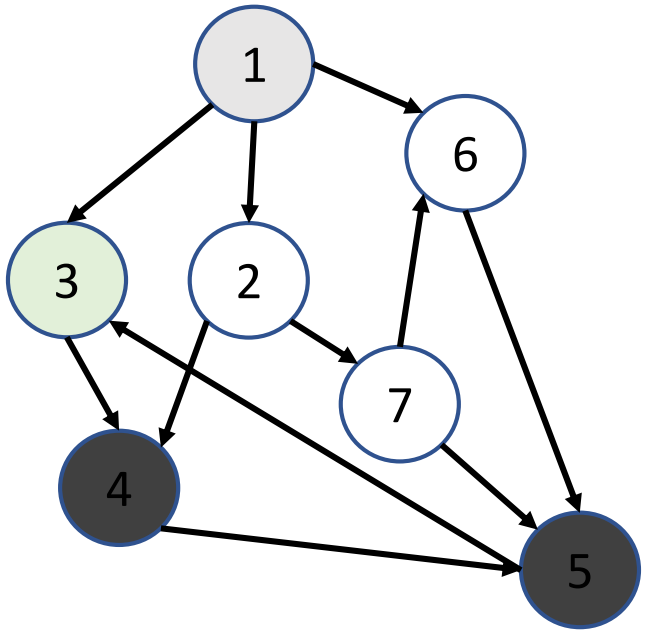
1	3																	

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

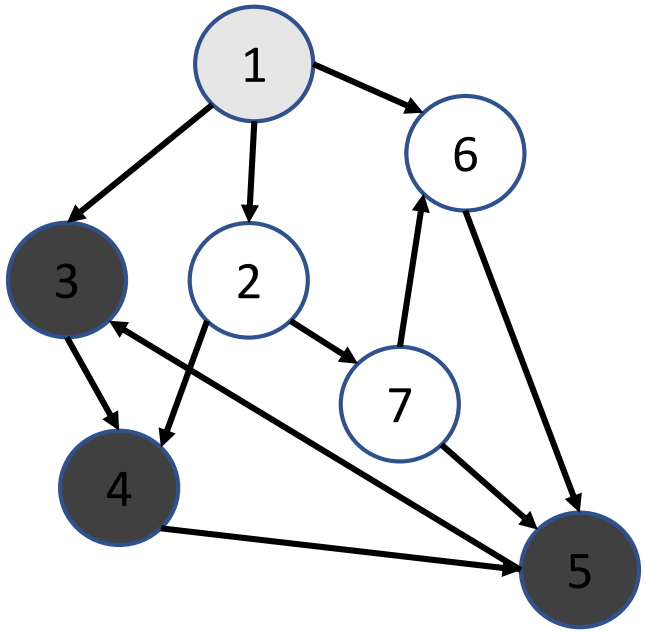
1	3																	

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

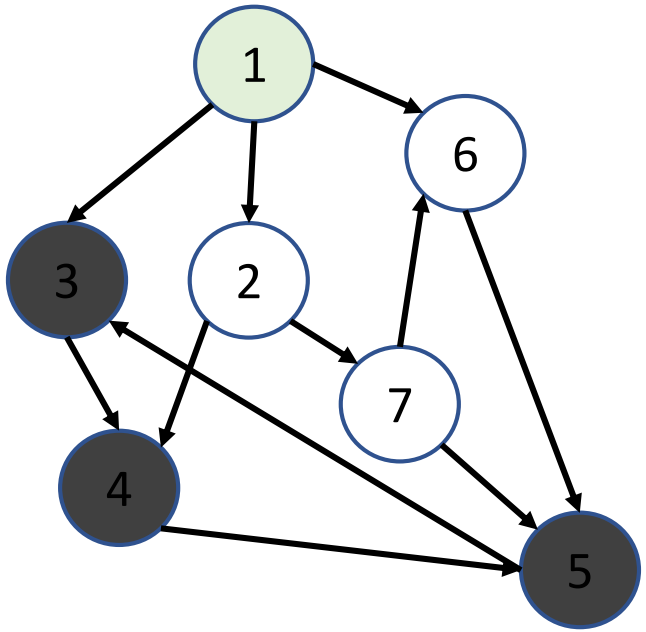
1																		

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

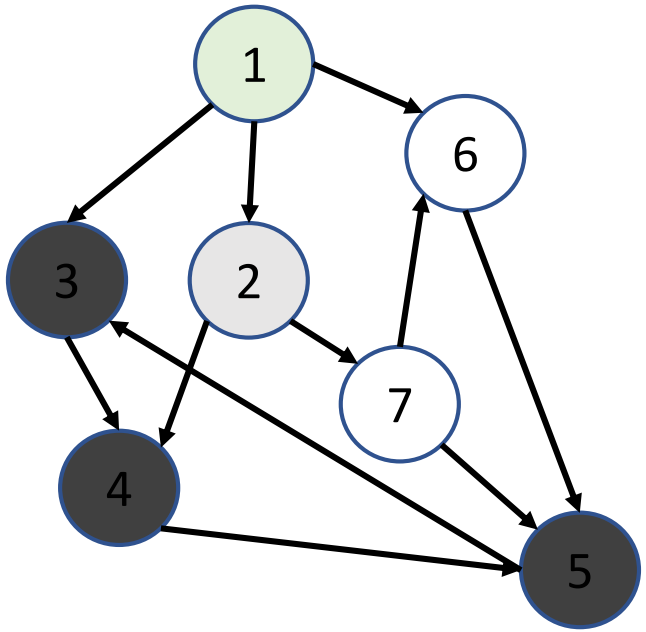
1																		

Vertex

1	3	4	5															

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

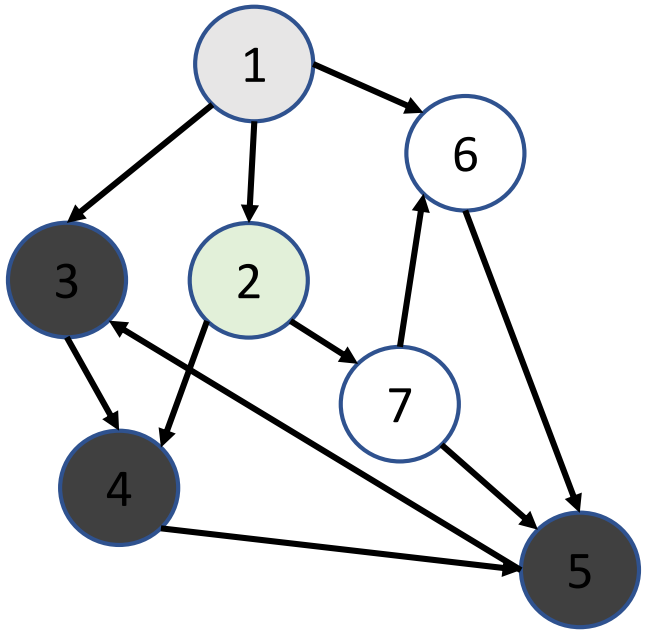
1	2																	

Vertex

1	3	4	5	2														

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

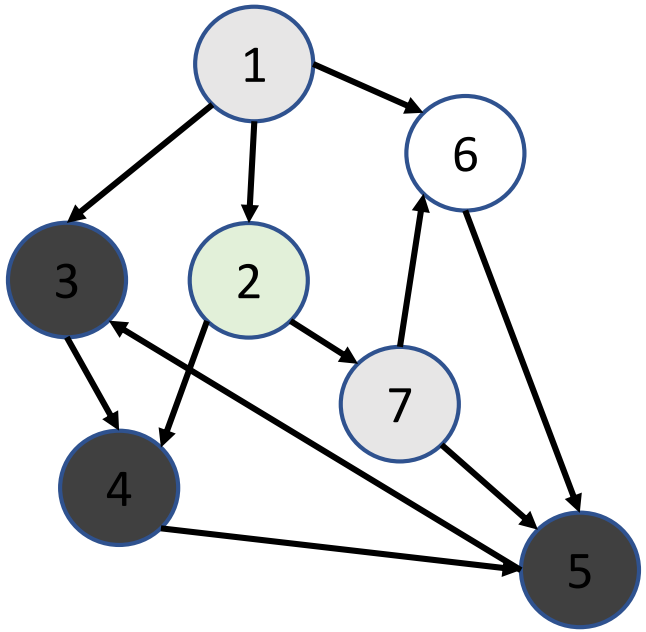
1	2																	

Vertex

1	3	4	5	2														

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

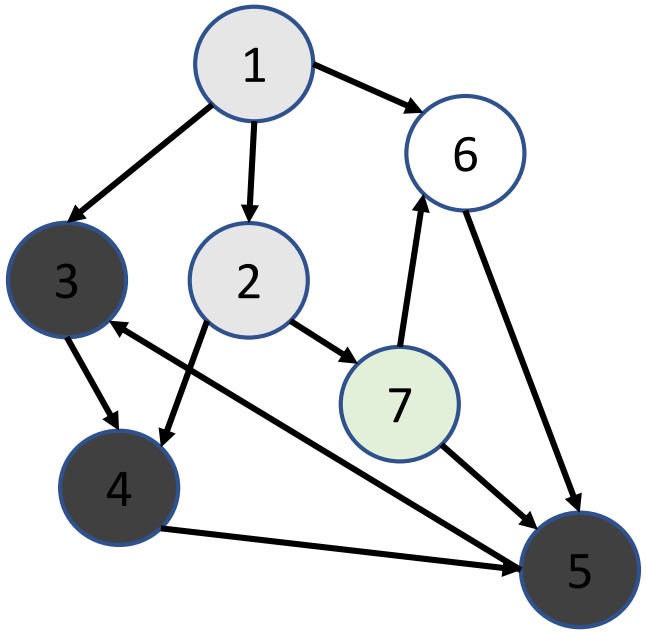
1	2	7																

Vertex

1	3	4	5	2	7													

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

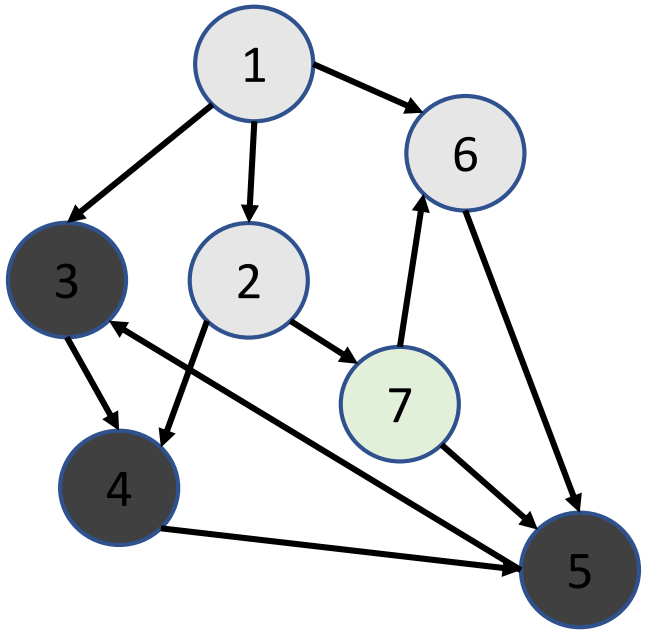
1	2	7																

Vertex

1	3	4	5	2	7													

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

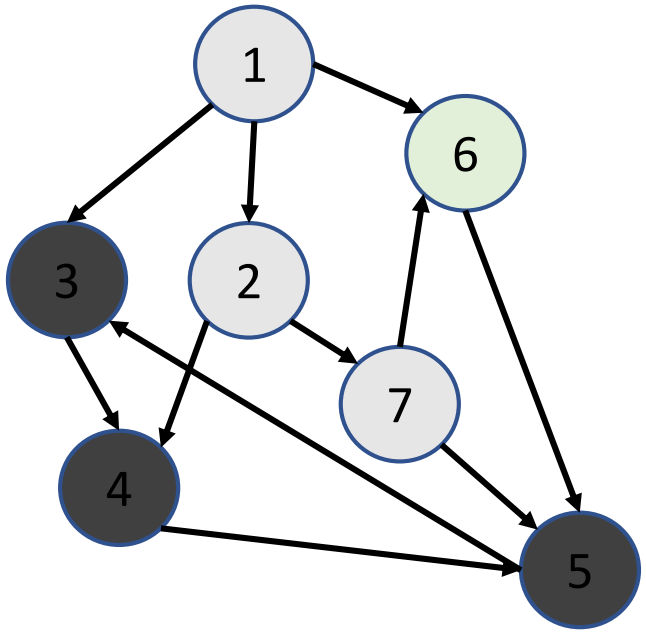
1	2	7	6															

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

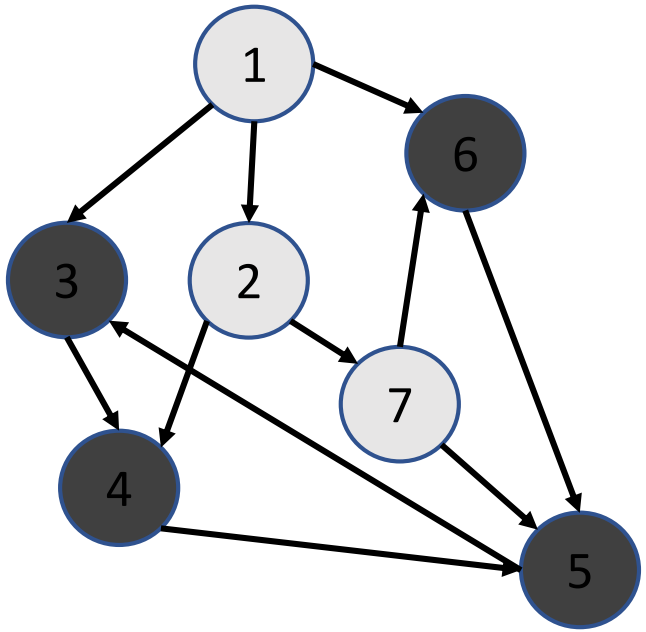
1	2	7	6															

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

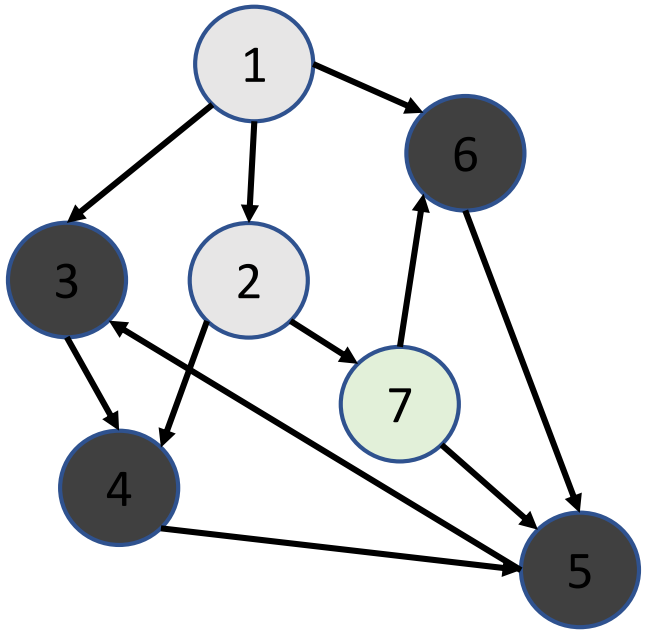
1	2	7																

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

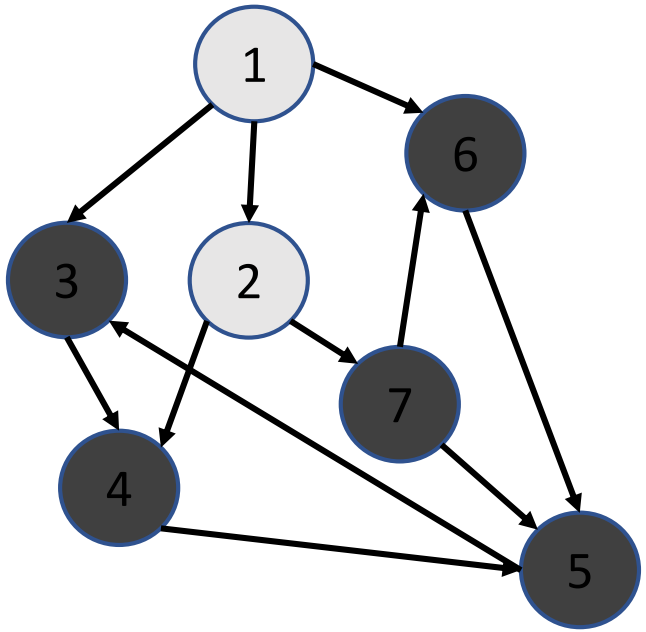
1	2	7																

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的**單一**相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

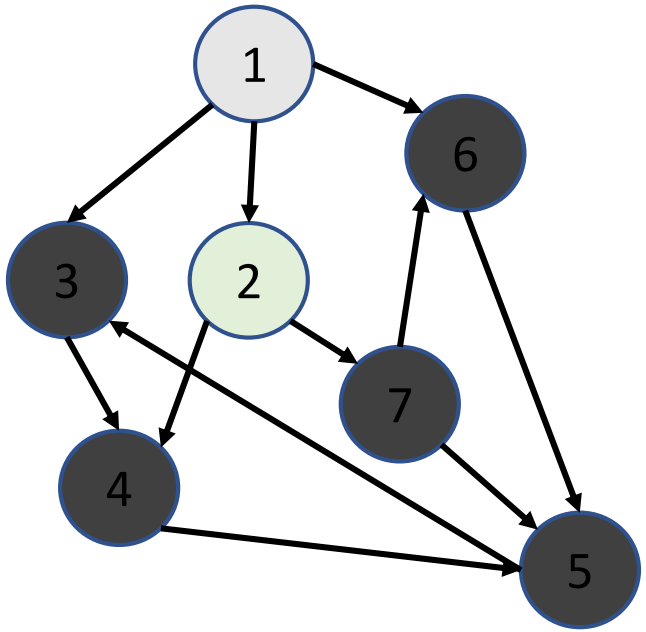
1	2																	

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

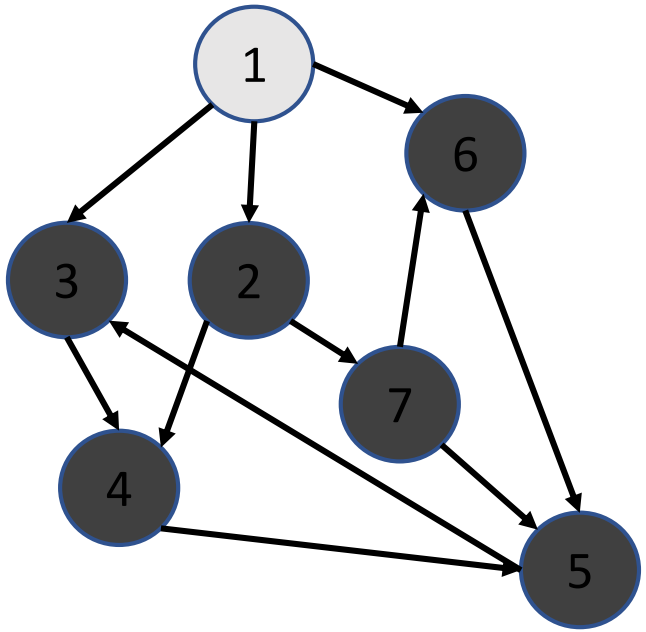
1	2																	

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

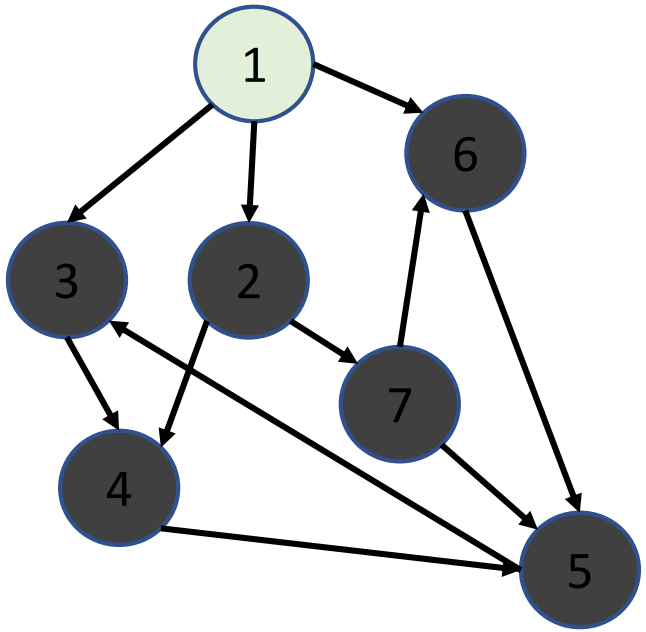
1																		

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

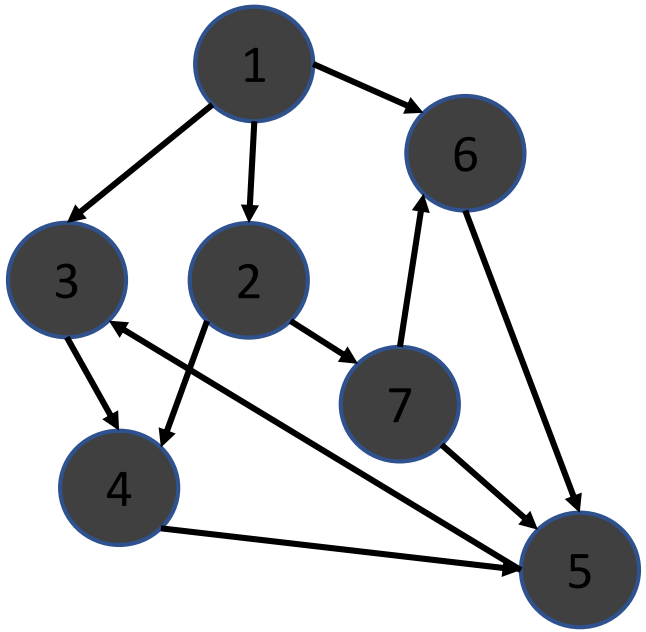
1																		

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



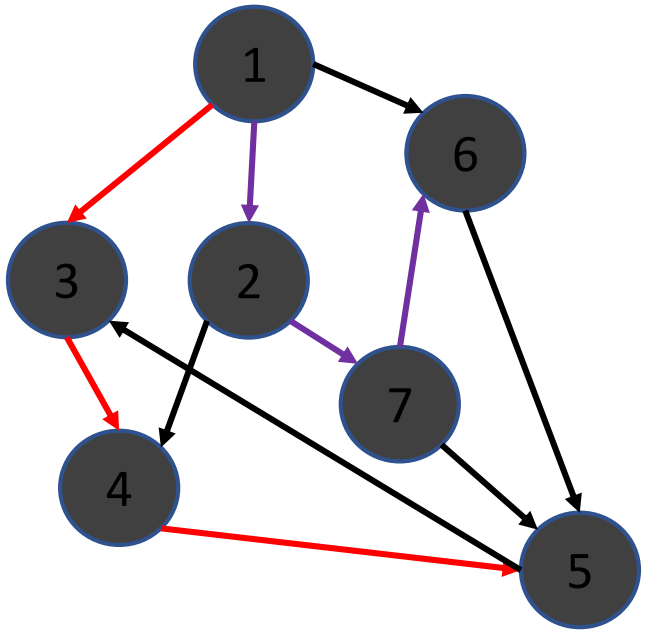
Path Stack

Vertex

1	3	4	5	2	7	6												

深度優先搜尋（DFS）

- 先不斷尋訪該頂點的單一相鄰節點
- 直至無法繼續尋訪就返回可繼續尋訪的頂點
- 通常使用 Stack 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



Path Stack

Vertex

1	3	4	5	2	7	6												

深度優先搜尋 (BFS)

```
                    DFS(G,s)
 1  for each vertex(v) in G:
 2        color[v] = white
 3  path_stack = {s}
 4  DFS_visit(s)
 5
 6  DFS_visit(vertex)
 7        color[vertex] = gray
 8        for each vertex(v) in vertex.adjacent():
 9              if color[v] == white
10                    DFS_visit(v)
11        color[vertex] = black
```

O(|V|)

O(|E|)

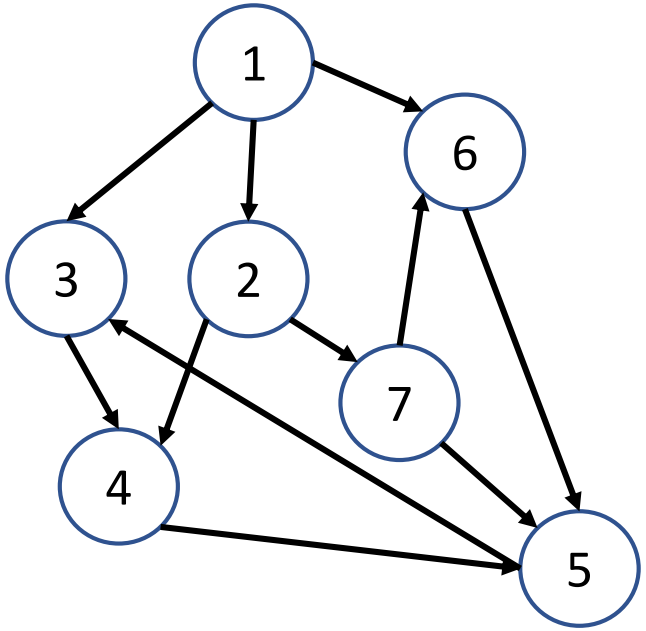
- 效能分析
  - 初始化：O(|V|)
  - 處理所有邊：O(|E|)
  - 總和：O(|V|+|E|)

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 Stack 前，該頂點已存在 Stack 裏頭
  2. 有向圖 (Directed Graph)

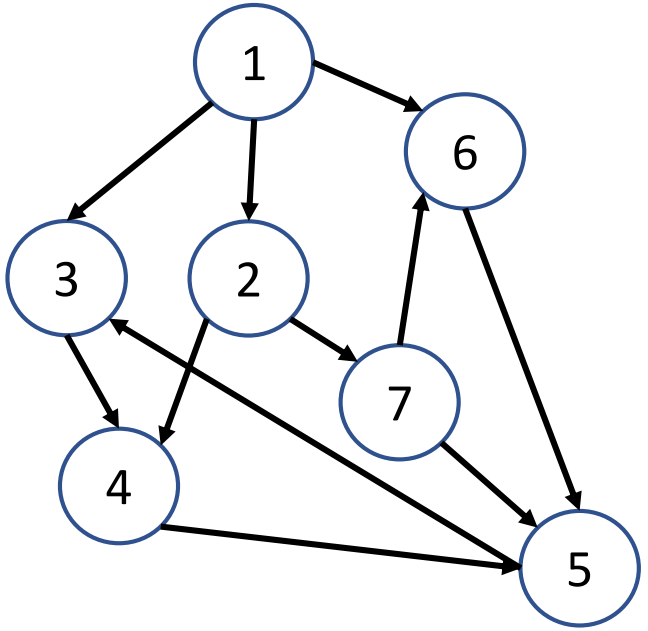
     計算 DFS 過程中，是否曾經過灰色的頂點



Example Code

Mission

從給定的頂點與邊中，印出深度優先搜尋的搜索過程。



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.

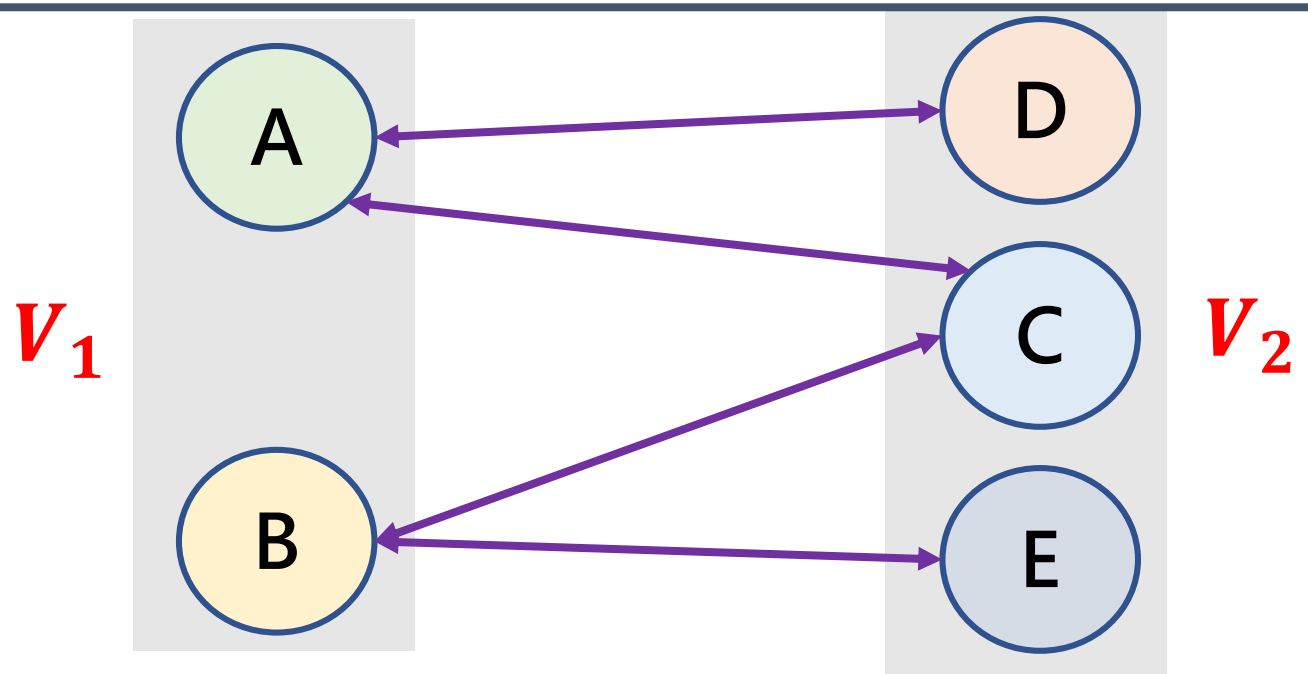
- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

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

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

二分圖 (Bipartite Graph)

- 二分圖 (Bipartite Graph)
  - 無向圖滿足可以把頂點分成兩集合 V_1, V_2
  - 使得所有邊 $e(v_1, v_2)$ 中 v_1, v_2 必不在同一集合
  - 或無向圖進行黑白染色後，所有黑白點不相鄰



Practice

Mission

Try LeetCode #785. Is Graph Bipartite?

There is an undirected graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v.

A graph is bipartite if the nodes can be partitioned into two independent sets A and B such that every edge in the graph connects a node in set A and a node in set B.

Return true if and only if it is bipartite.

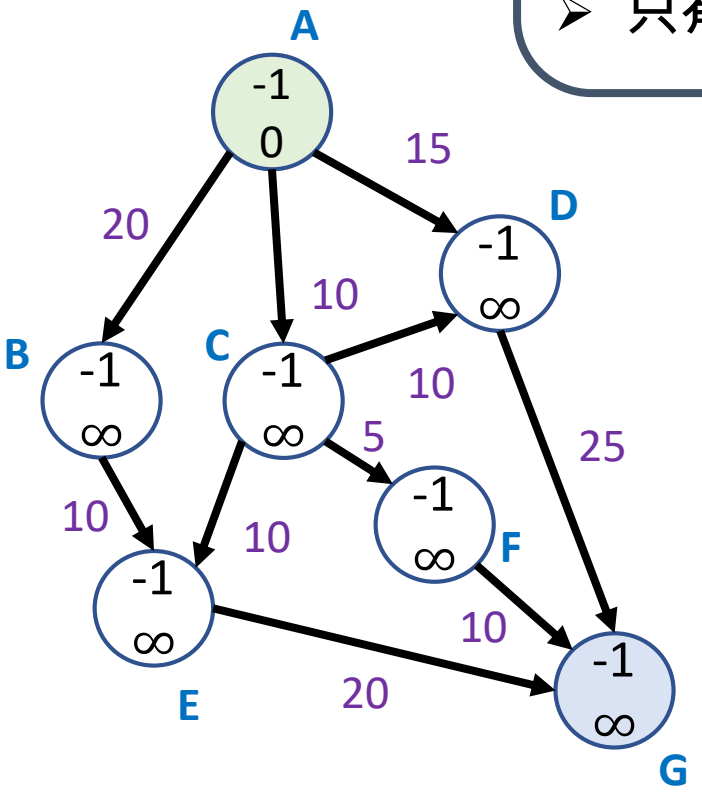
Ref：https://leetcode.com/problems/is-graph-bipartite/

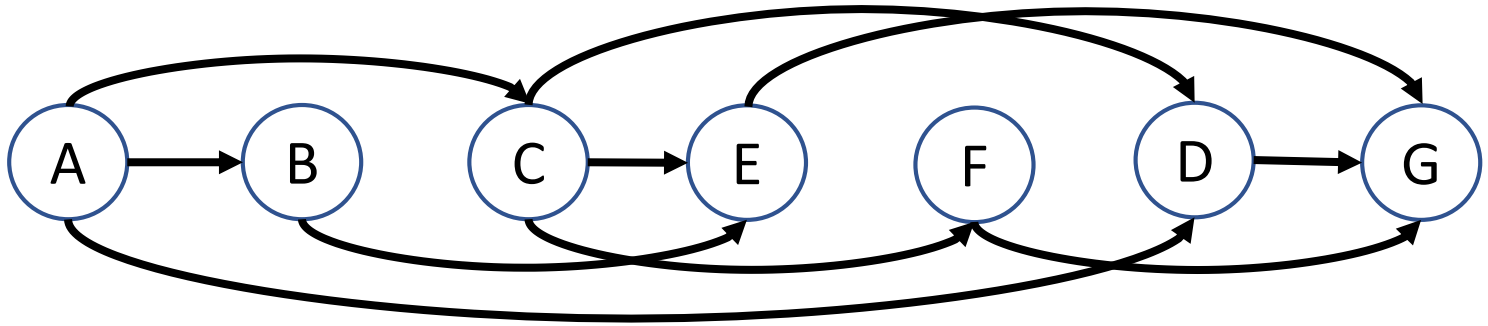
拓樸排序

Topological Sort

Topological Sort(拓撲排序)

- 每條在有向無環圖的 Edge(A,B)
  - ✓ 拓樸排序必是 Vertex(A) 在 Vertex(B) 之前
- 只有有向無環圖的拓樸排序才有意義

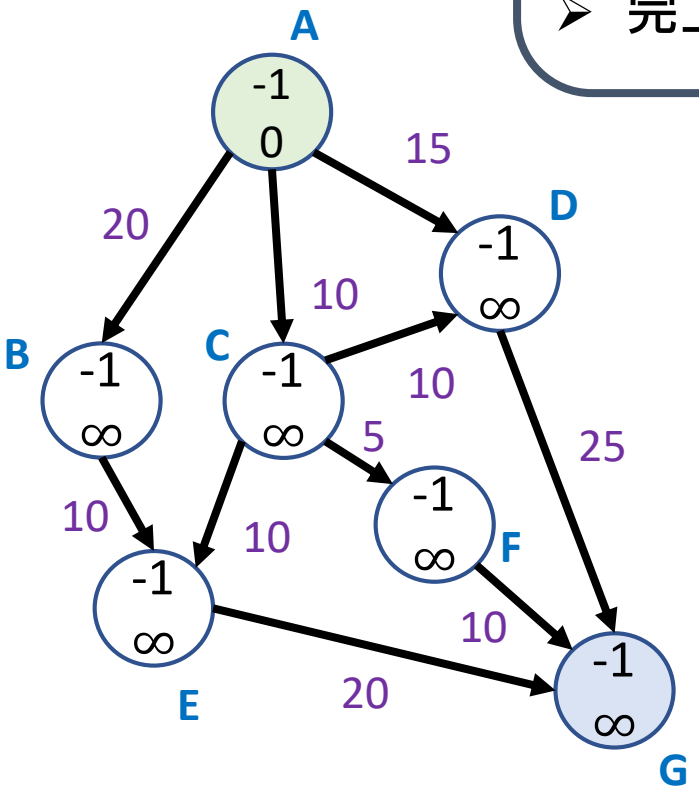


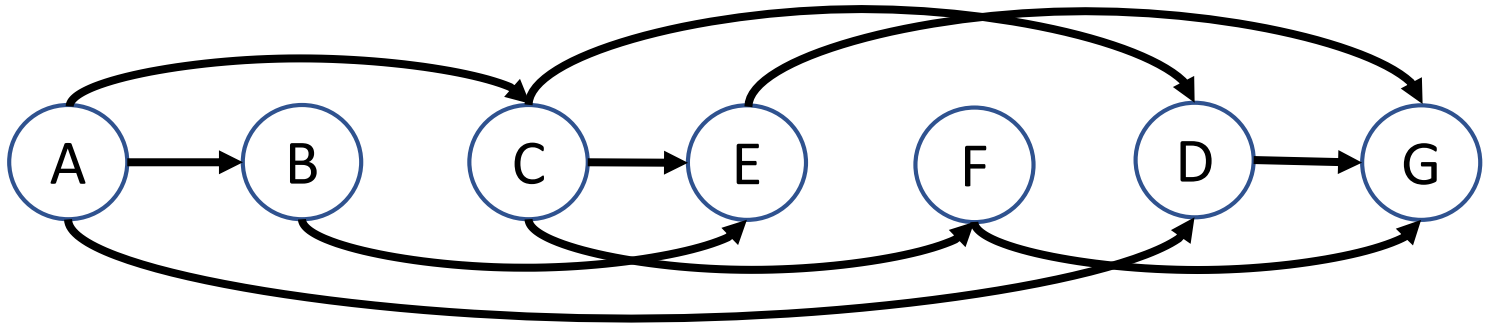


Topological Sort

Topological Sort(拓撲排序) 應用

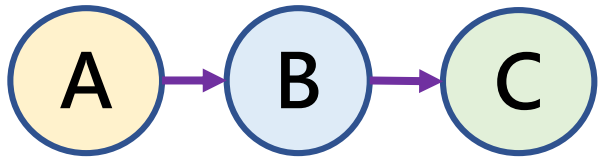
- 修課地圖
- 編譯順序
- 完工順序





Topological Sort

- Topological Sort(拓撲排序)
  - ➢ DFS 離開順序大小就是 DAG 順序
    - ◘ A→B→C

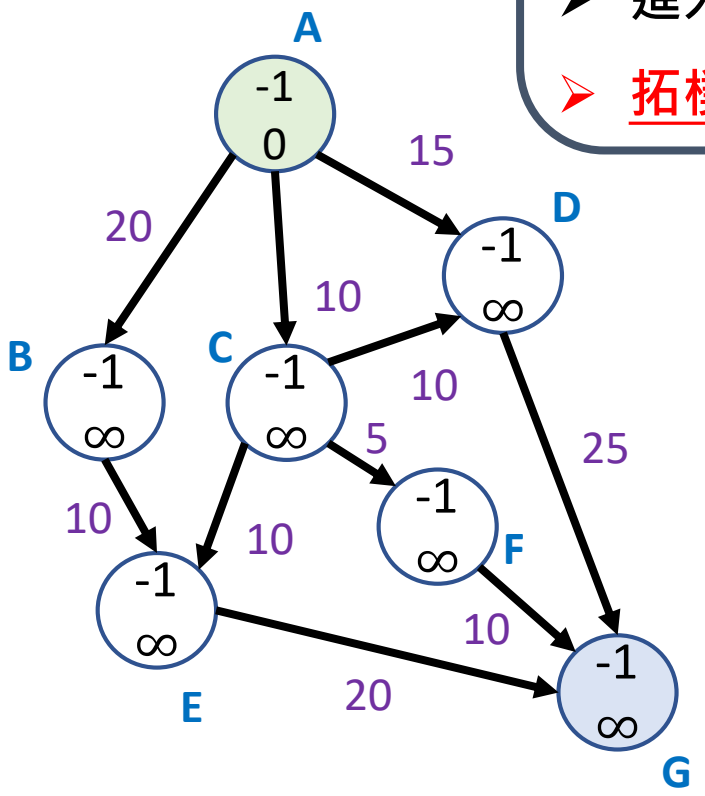


Start from A					
A					
	B				
		C			
Start from B					
B				A	
	C				
Start from C					
C		B		A	

Topological Sort

如何產生 Topological Sort(拓撲排序)

- 進行一次 DFS 便可以把路過次序記錄下來
- 依照離開的時間戳記就可以進行拓樸排序
- 進入→塗成灰色的瞬間；離開→塗成黑色的瞬間
- 拓樸排序不是唯一解！

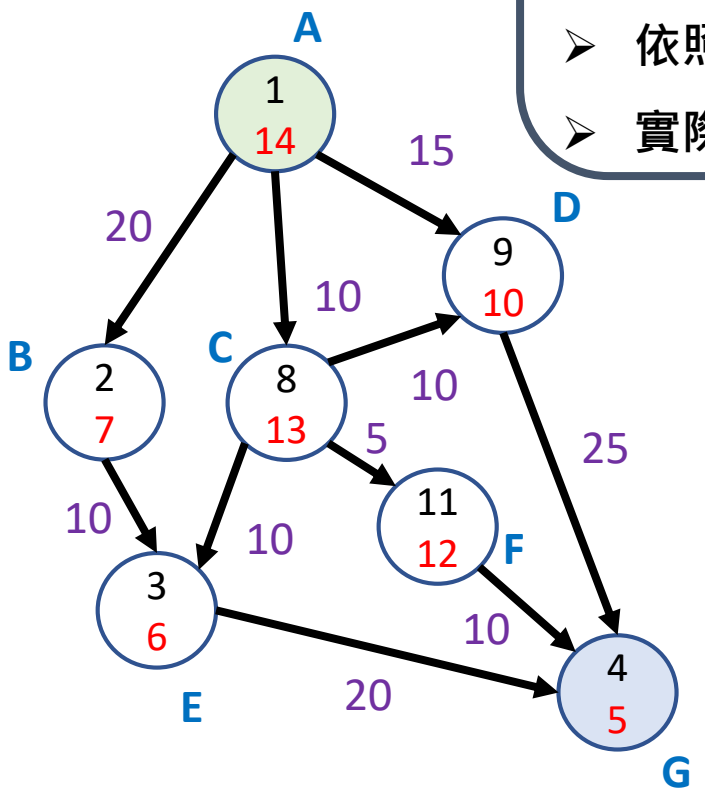


Vertex	進入	離開
A	1	14
B	2	7
C	8	13
D	9	10
E	3	6
F	11	12
G	4	5

Topological Sort

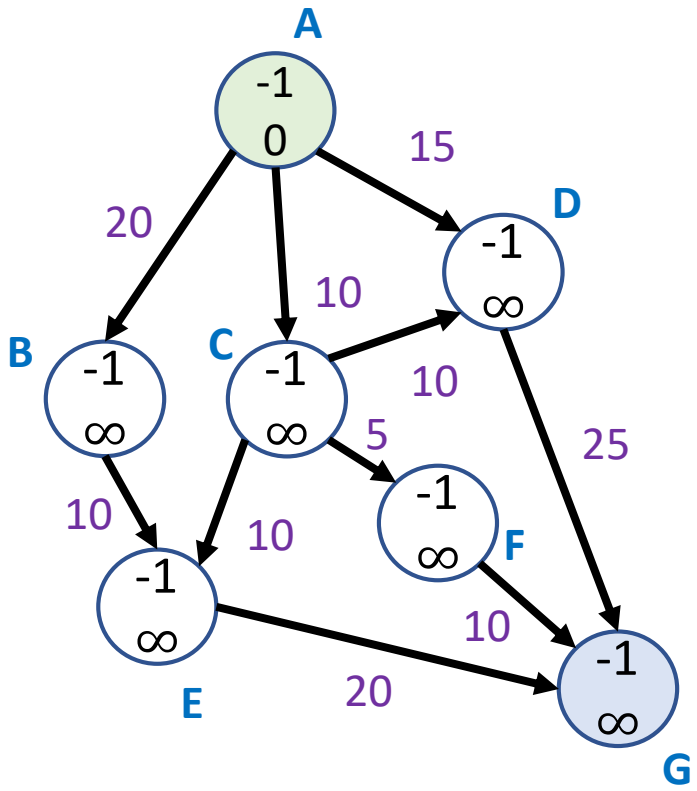
如何產生 Topological Sort(拓撲排序)

- 離開的時間戳記代表處理完該頂點的時間點
  - ✓ 也就是塗成黑色的時間點
- 越上游的頂點，離開時間戳記越大
- 依照離開時間戳記的大小便可以得到拓樸排序
- 實際上不用記錄數字，依照離開順序放到 Stack即可



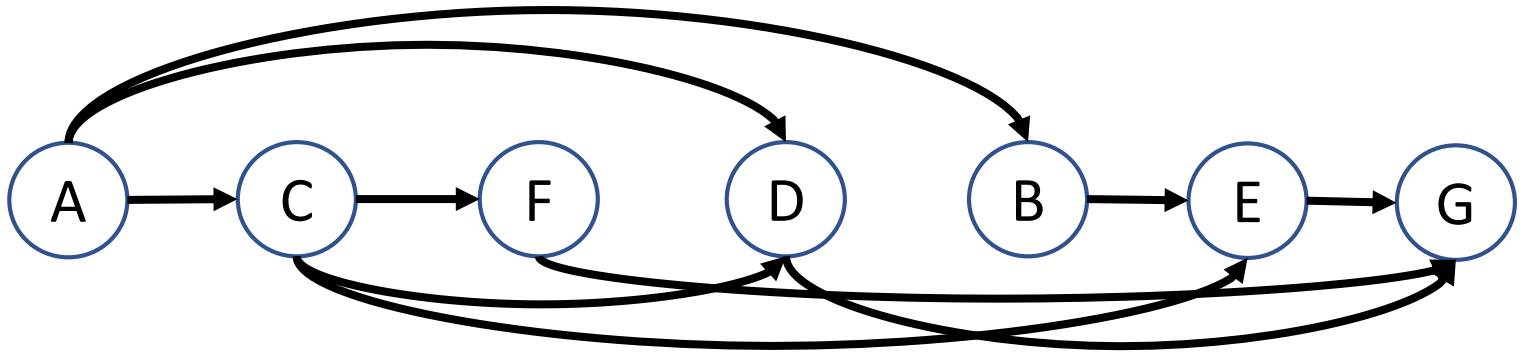
Vertex	進入	離開
A	1	14
B	2	7
C	8	13
D	9	10
E	3	6
F	11	12
G	4	5

Topological Sort



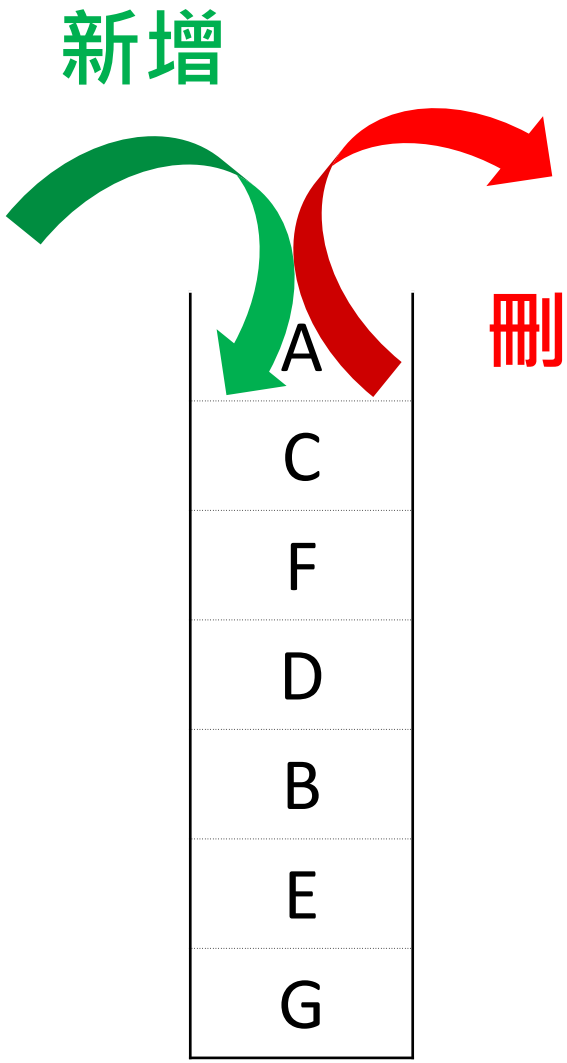
Vertex	進入	離開
A	1	14
B	2	7
C	8	13
D	9	10
E	3	6
F	11	12
G	4	5

ACFDBEG



Topological Sort

Vertex	進入	離開
A	1	14
B	2	7
C	8	13
D	9	10
E	3	6
F	11	12
G	4	5
ACFDBEG		

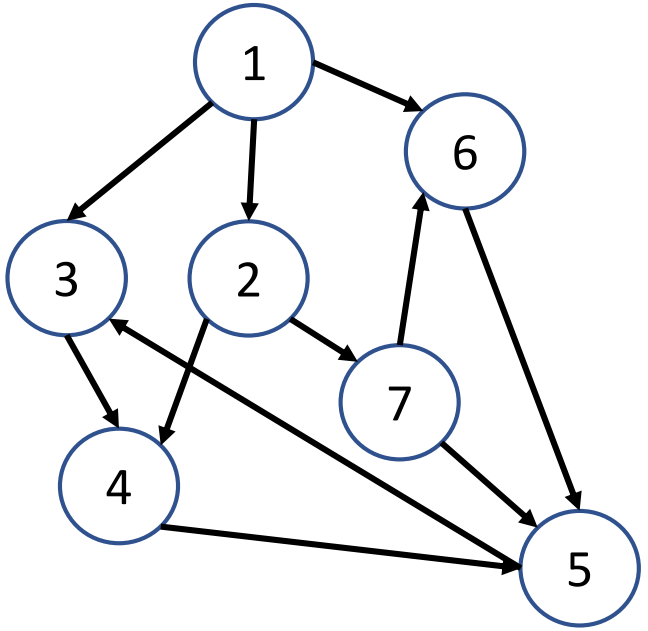


A C F D B E G

Example Code

Mission

利用深度優先搜尋演算法列出拓樸排序



Practice

Mission

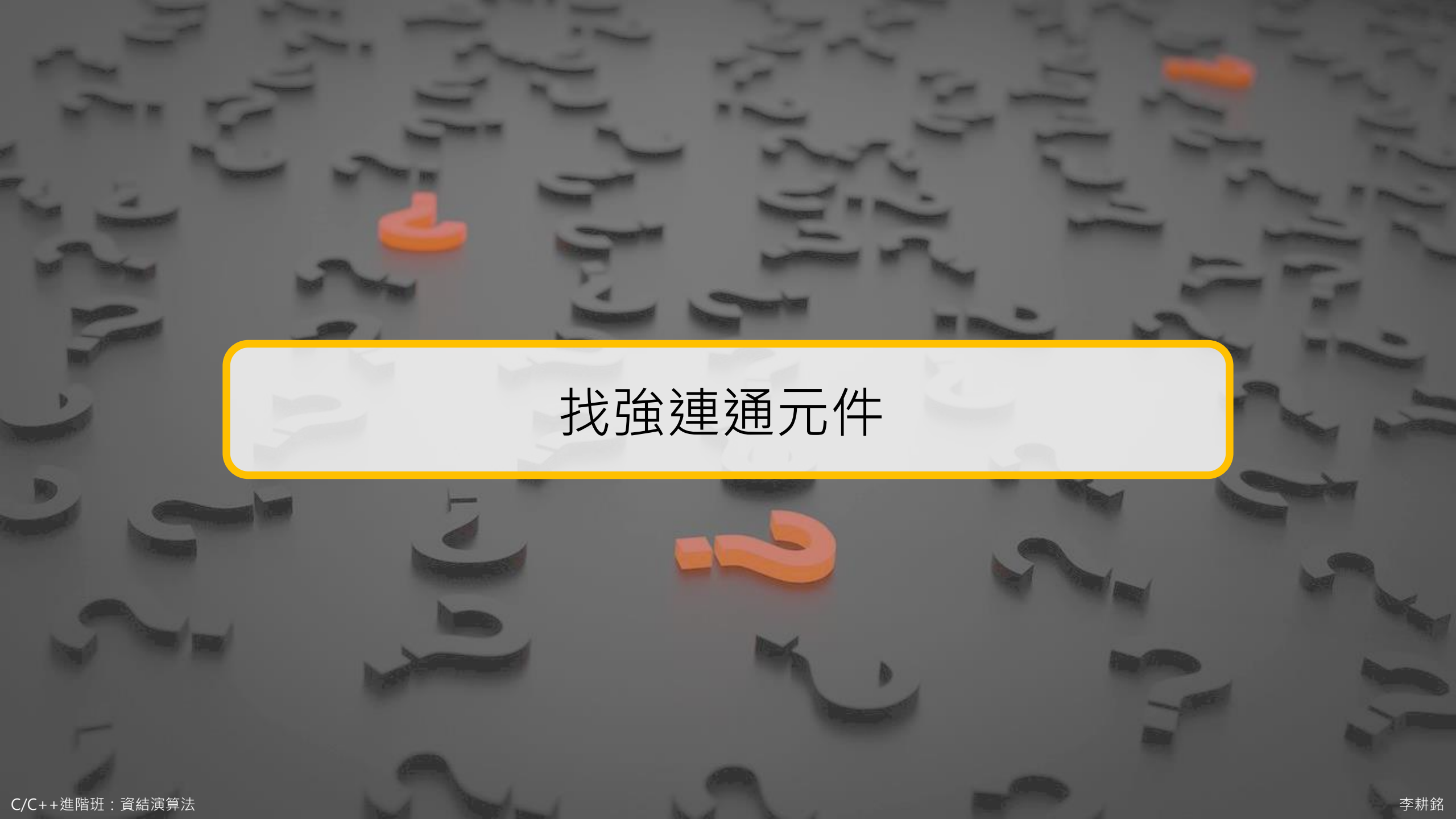
Try LeetCode #210. Course Schedule II

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 the ordering of courses you should take to finish all courses. If there are many valid answers, return any of them. If it is impossible to finish all courses, return an empty array.

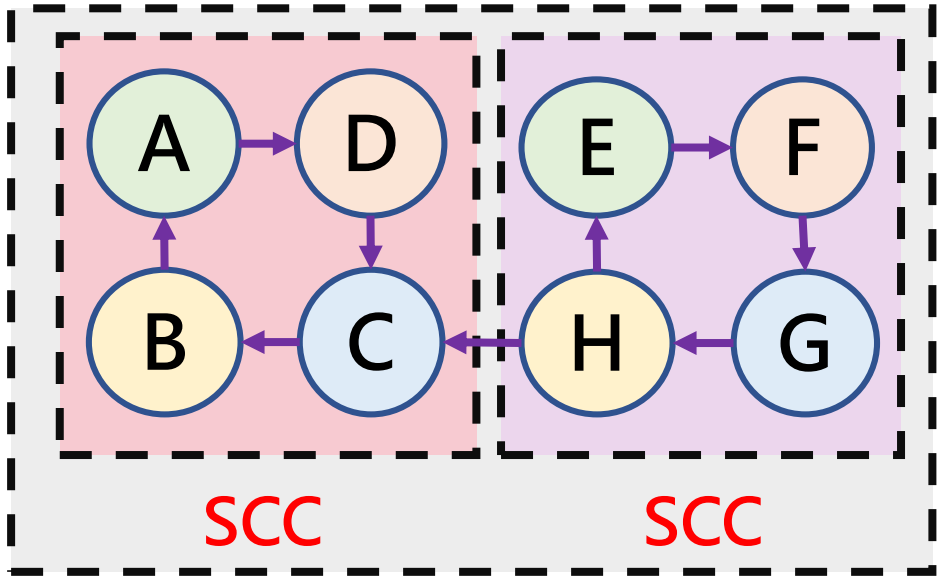
Ref：https://leetcode.com/problems/course-schedule-ii/



找強連通元件

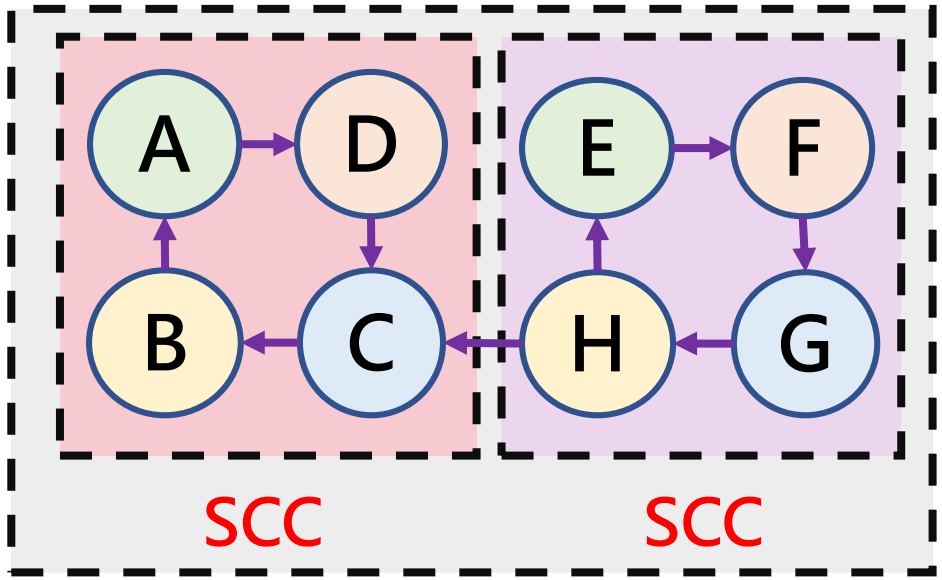
連通圖 (Connected Graph)

- 強連通元件 (Strong Connected Component，SCC)
  - 任兩點 (u,v) 間必存在 u→v 及 v→u 的路徑
  - SCC 可以組成極大強連通子圖



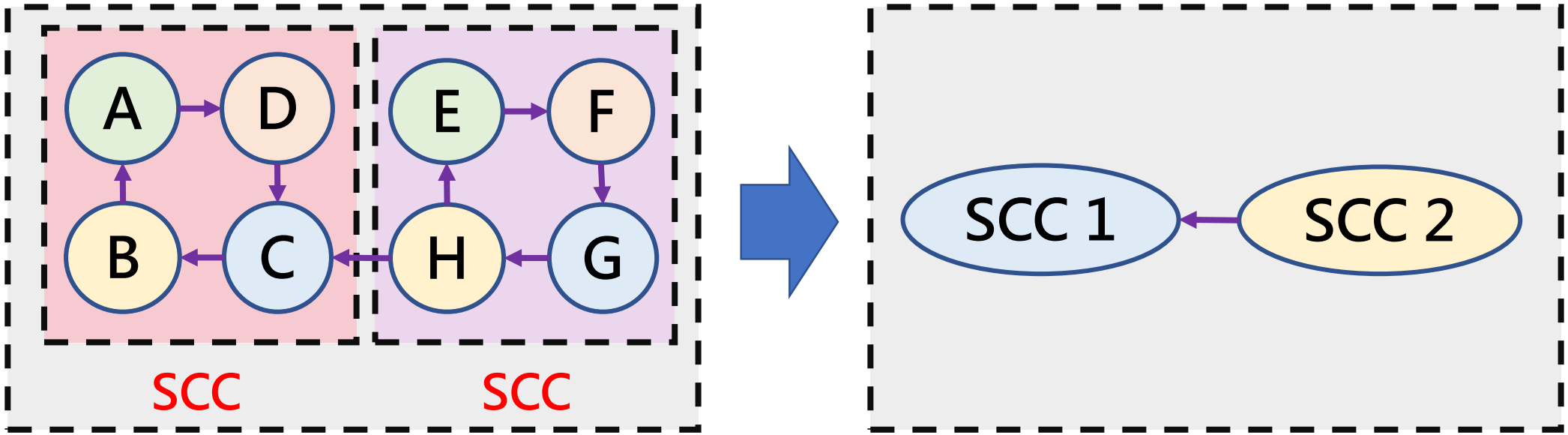
連通圖 (Connected Graph)

- Key Point：不論從 SCC 的哪一點開始 DFS，都可以走遍所有 SCC 的點
  - SCC 定義：任兩點間必有路徑！
  - 適當順序下，該次 DFS 所走過的所有點可以視為同一個 SCC
    - A→D→C→B
    - E→F→G→H



連通圖 (Connected Graph)

- Key Point：把 SCC 看做一個節點，會形成 DAG！
  - 若非 DAG (有環)的話，代表 SCC 1可以到 SCC 2
  - 兩者為同一個 SCC，否證之



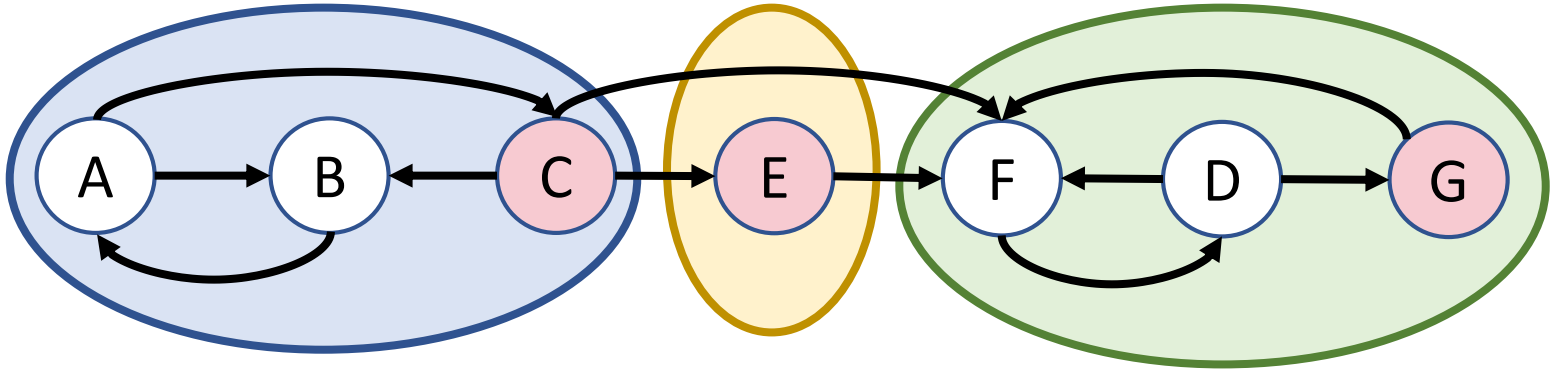
連通圖 (Connected Graph)

- 依照離開順序，由小到大進行DFS
  - 與 DAG 反向，離開順序越小代表越下游 (?)
  - 依照與 DAG 的反向進行 DFS (因 SCC 可連接任意兩點)
    - 第一次 DFS → 找出 SCC 3
    - 第二次 DFS → 找出 SCC 2
    - 第三次 DFS → 找出 SCC 1

  - 這樣可以萬無一失地逐一找出SCC (嗎 ?)

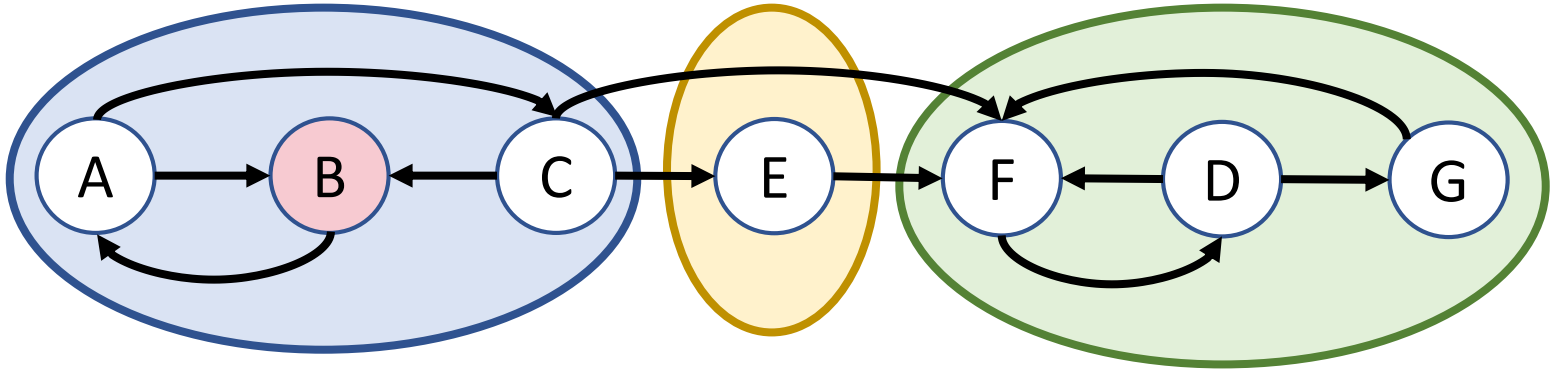
連通圖 (Connected Graph)

- 實際例子：DFS起點 B
  - 進入：BACEFDG
  - 離開：GDFECAB
  - DFS@G → SCC：FDG
  - DFS@E → SCC：E
  - DFS@C → SCC：CBA



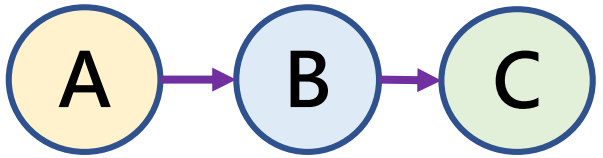
連通圖 (Connected Graph)

- 來看個失敗例子：DFS起點 A
  - 進入：ABCEFDG
  - 離開：BGDFECA
  - DFS@B → SCC：ABCDEFG，錯了！
  - 首先離開的點，有可能在 SCC 構成的 DAG 上游
  - 無法確保離開順序最小的點在 SCC 構成的 DAG 最下游

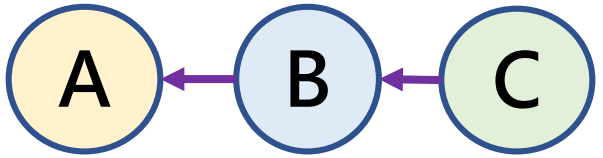


連通圖 (Connected Graph)

- 原始圖 G
  - 進入：BCA
  - 離開：CBA



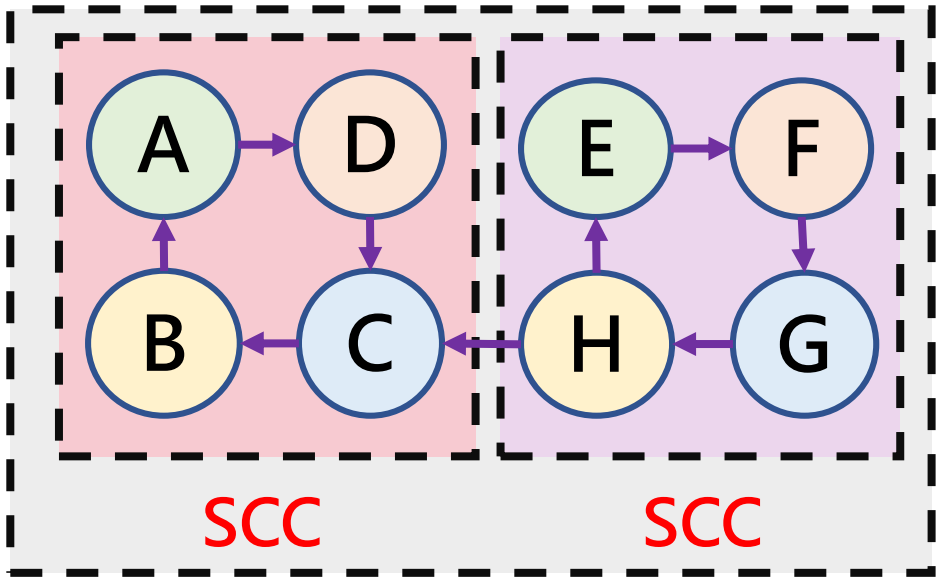
  - 離開順序最小的未必在 DAG 的最下游！
  - 離開順序最大的必在 DAG 的最上游！
  - 但我們想要的從最下游開始呀 QQ
- 顛倒圖 G^T
  - 進入：BAC
  - 離開：ABC

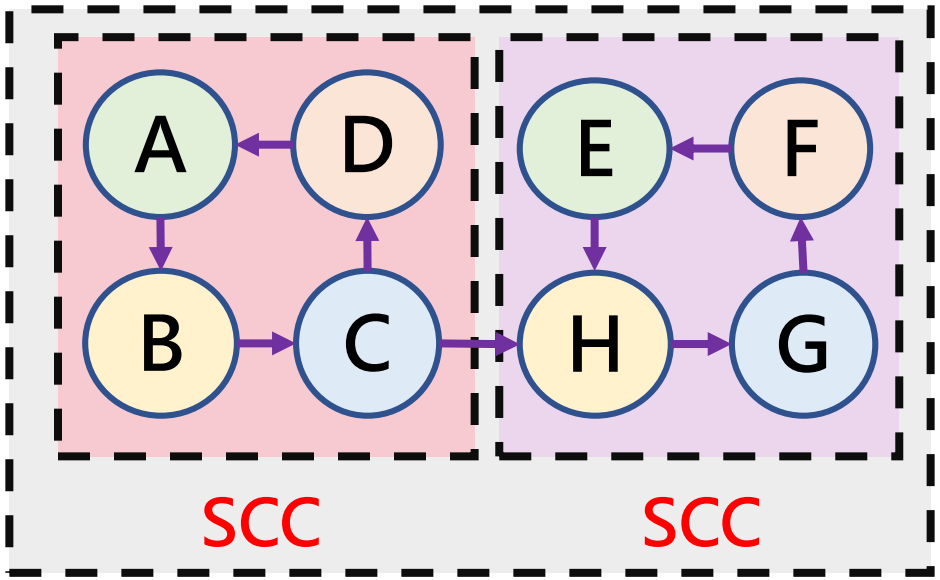


  - 最上游就變最下游了，而且 SCC 不變！
- 以原圖 G 的 DFS 離開大小順序 (ABC) 在 G^T 上找 SCC

連通圖 (Connected Graph)

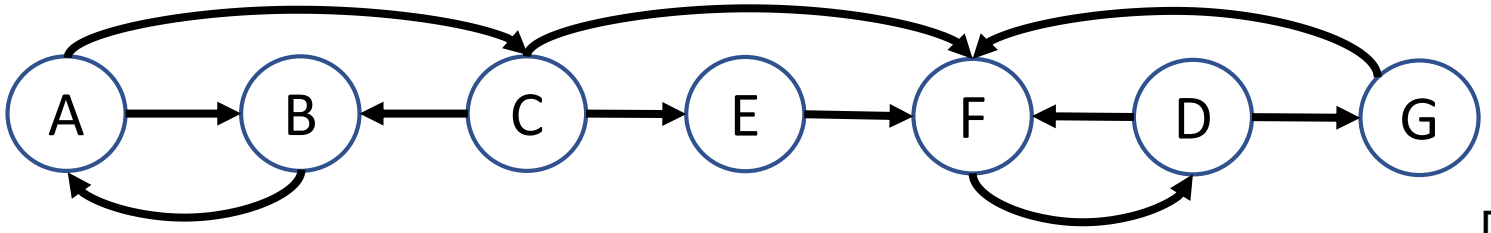
- Key Point：把邊的方向反過來，SCC 還是會一樣！



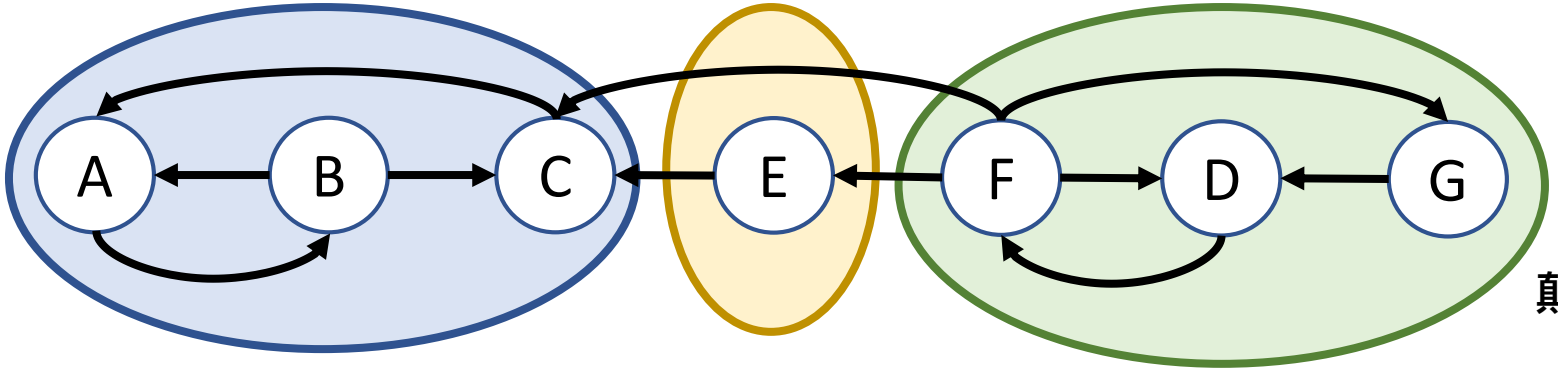


連通圖 (Connected Graph)

- 剛剛的失敗例子：DFS起點 A
  - 進入：ABCEFDG
  - 離開：BGDFECA
  - 原圖離開順序最大的 A 成功落在 G^T 最下游！



原始圖 G



顛倒圖 G^T

連通圖 (Connected Graph)

- Question：
  - 這樣還是只能確保最大的在 G^T 最下游啊？
  - 其他離開順序比較小呢不一定會照順序啊？
    - 因為每輪 DFS 完都會刪去已尋訪過的
    - 剩下離開順序的最大值必在 G^T 剩下圖形的最下游
  - Example：
    1. G 離開順序最大的點在 G^T 進行 DFS 可找出 SCC 3

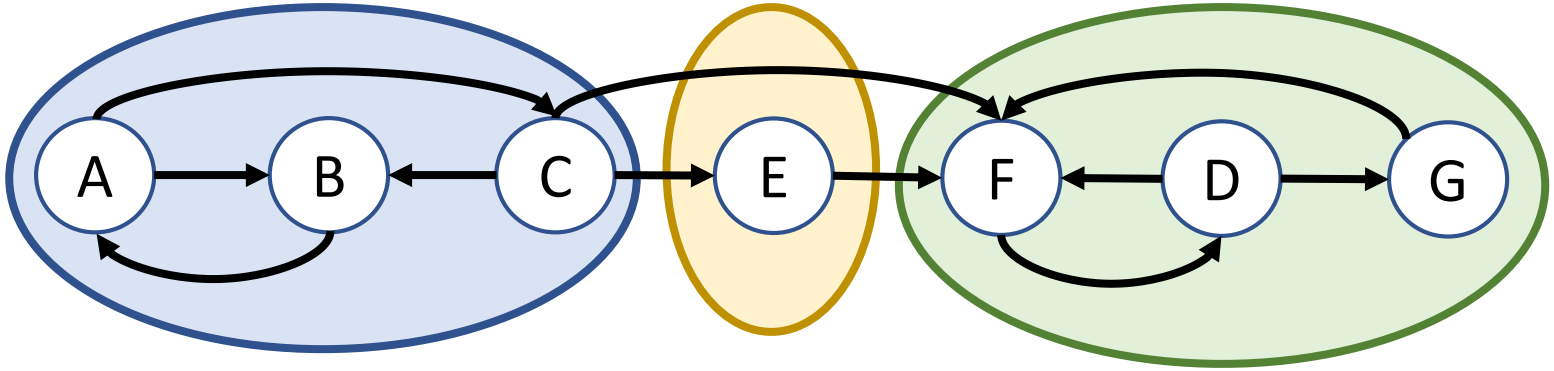
連通圖 (Connected Graph)

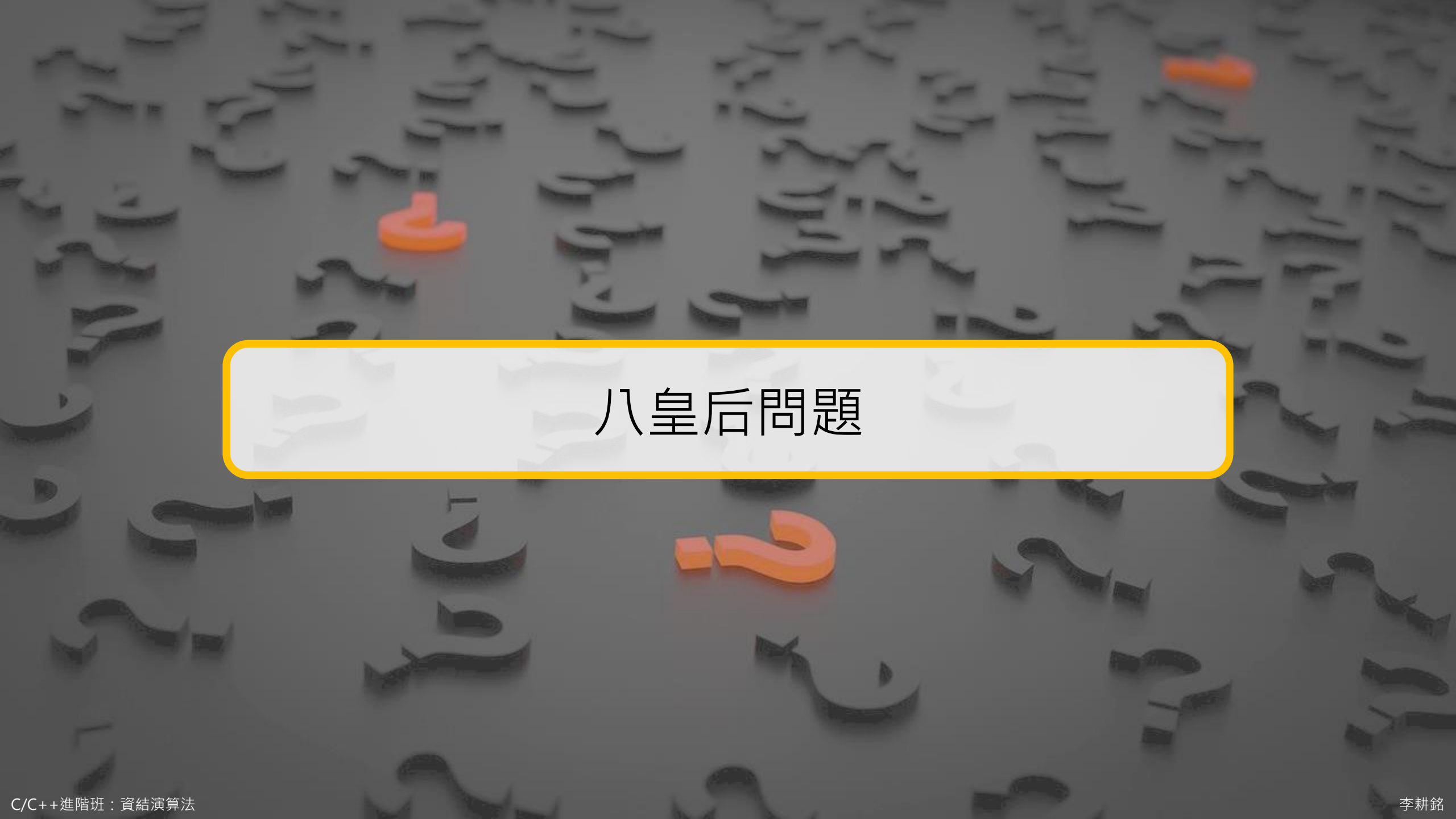
- Question：
  - 這樣還是只能確保最大的在 G^T 最下游啊？
  - 其他離開順序比較小呢不一定會照順序啊？
    - 因為每輪 DFS 完都會刪去已尋訪過的
    - 剩下離開順序的最大值必在 G^T 剩下圖形的最下游
  - Example：
    1. G 離開順序最大的點在 G^T 進行 DFS 可找出 SCC 3
    2. 剩下的點之中，離開順序最大的必在 G^T 的最下游

Example Code

Mission

列出圖裡的所有 SCC 的組合



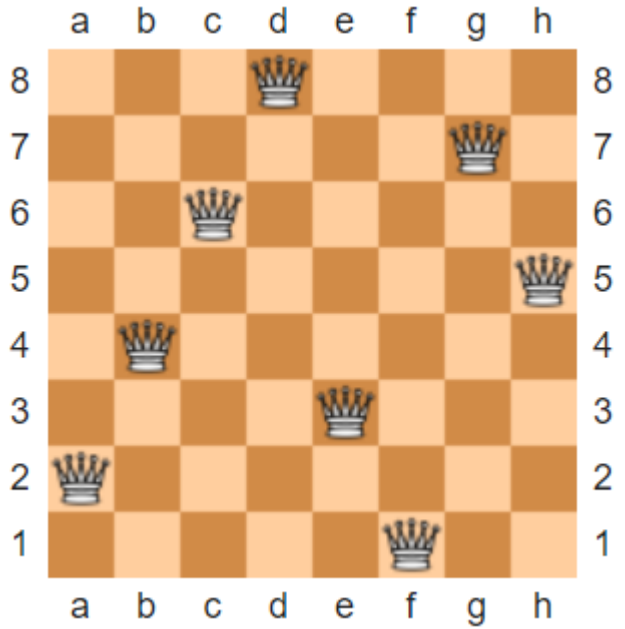


八皇后問題

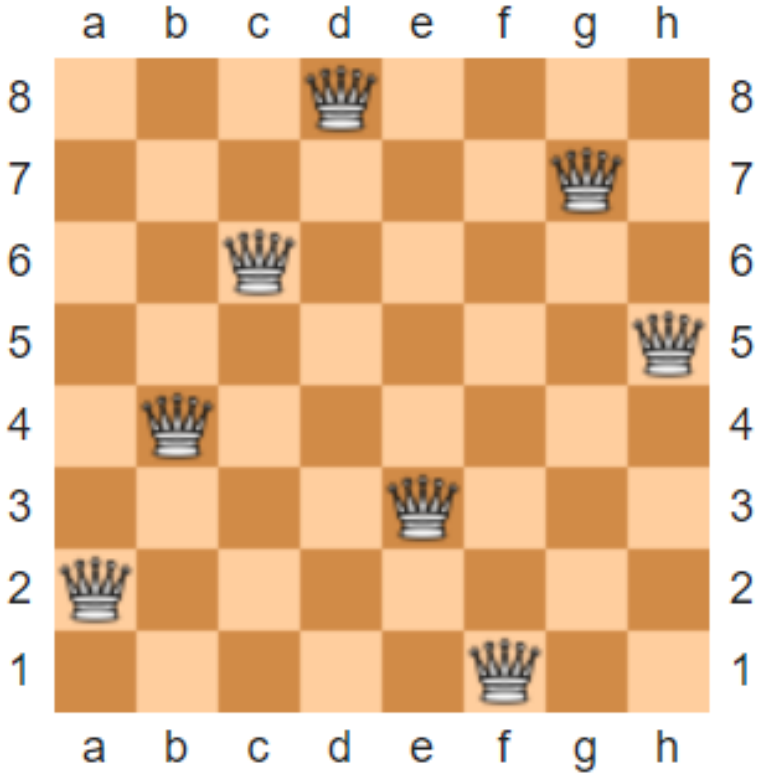
Eight Queens Puzzle

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n queens problem of placing n non-attacking queens on an n×n chessboard, for which solutions exist for all natural numbers n with the exception of n = 2 and n = 3.

Wikipedia

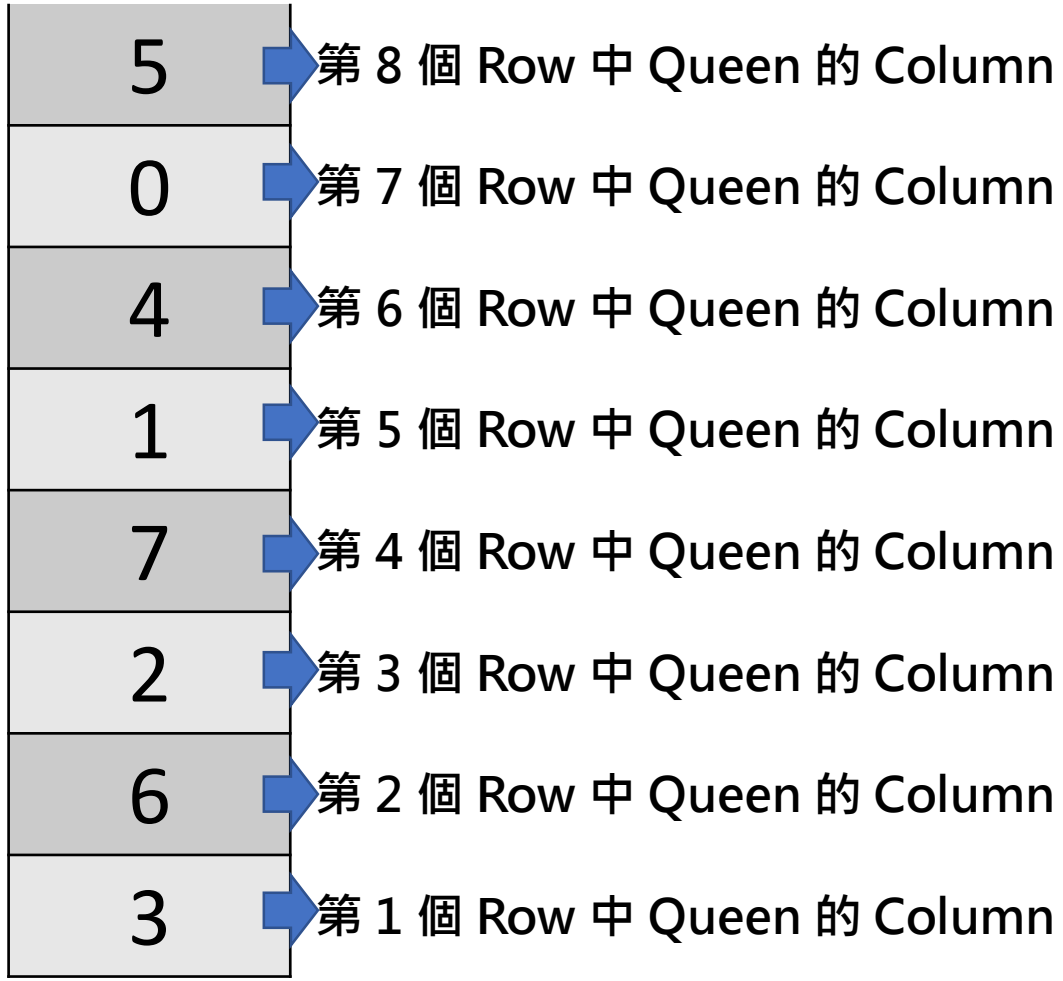


Practice 5

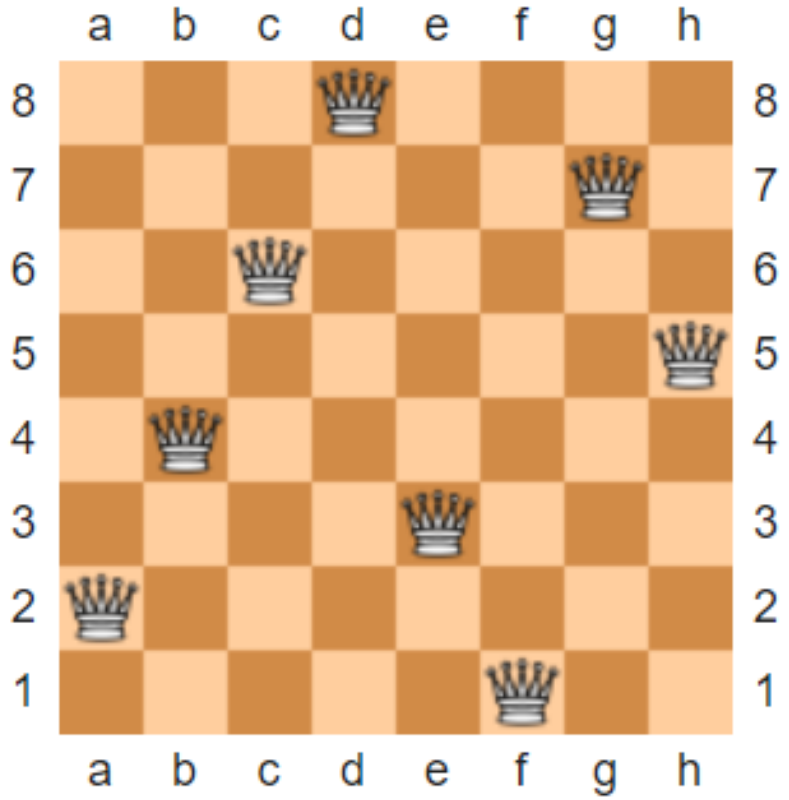



```
void print_stack(stack<int> s){
    if(s.empty())
        return ;
    int col = s.top();
    s.pop();
    print_stack(s);
    cout << col+1 << " ";
    s.push(col);
}
```

Eight Queens Puzzle



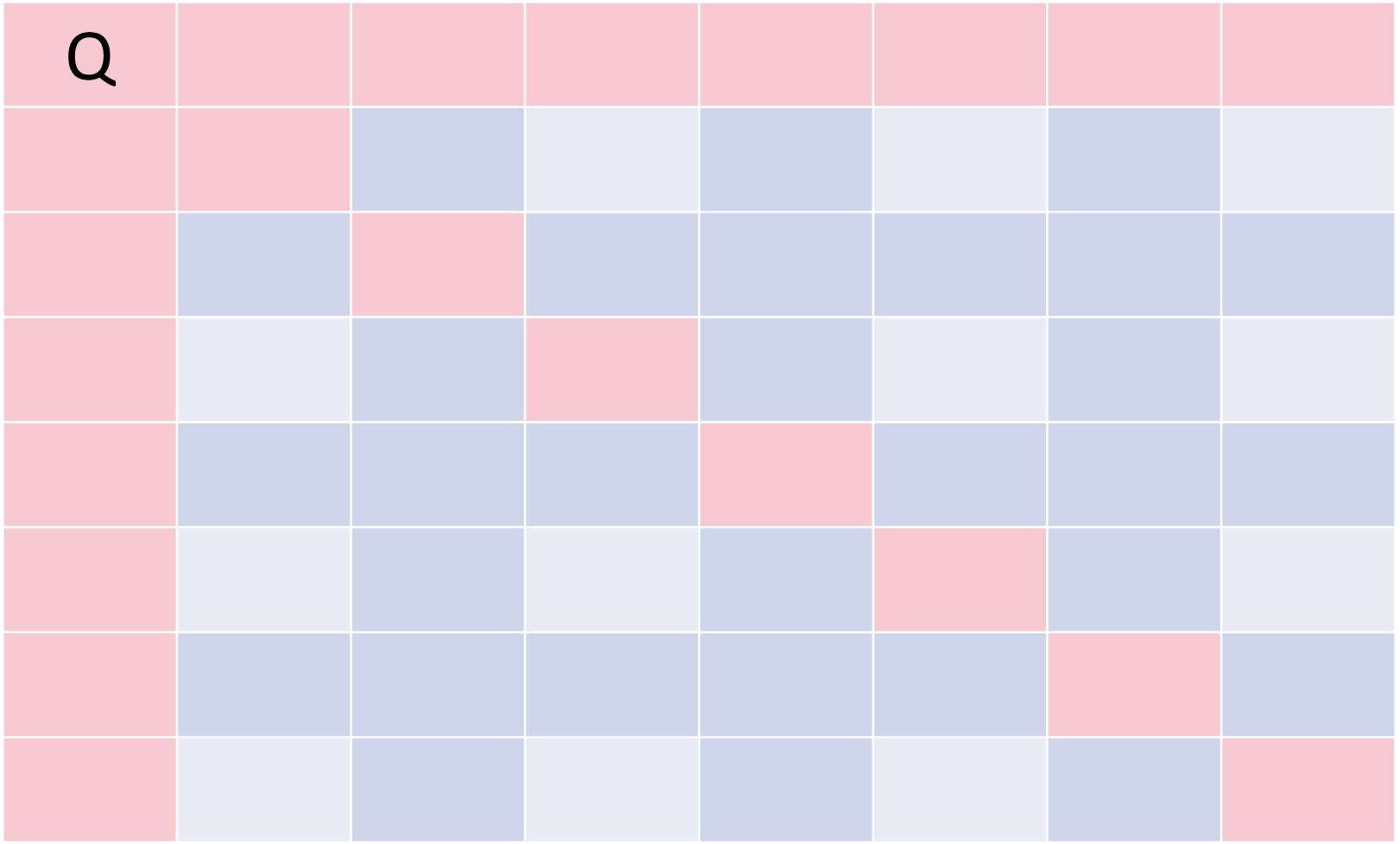
Stack

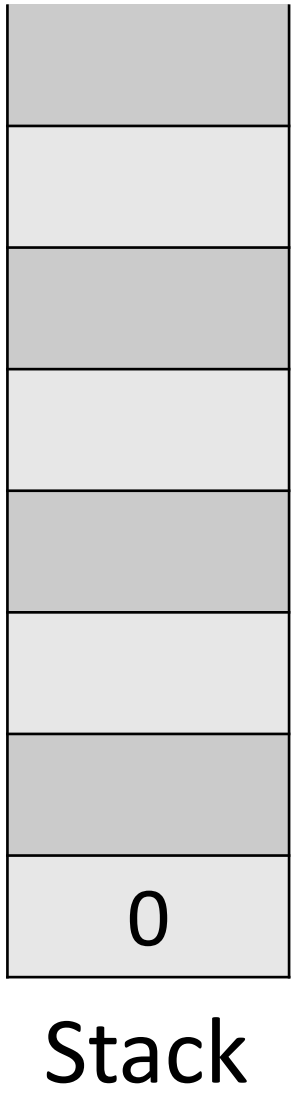


Example Code

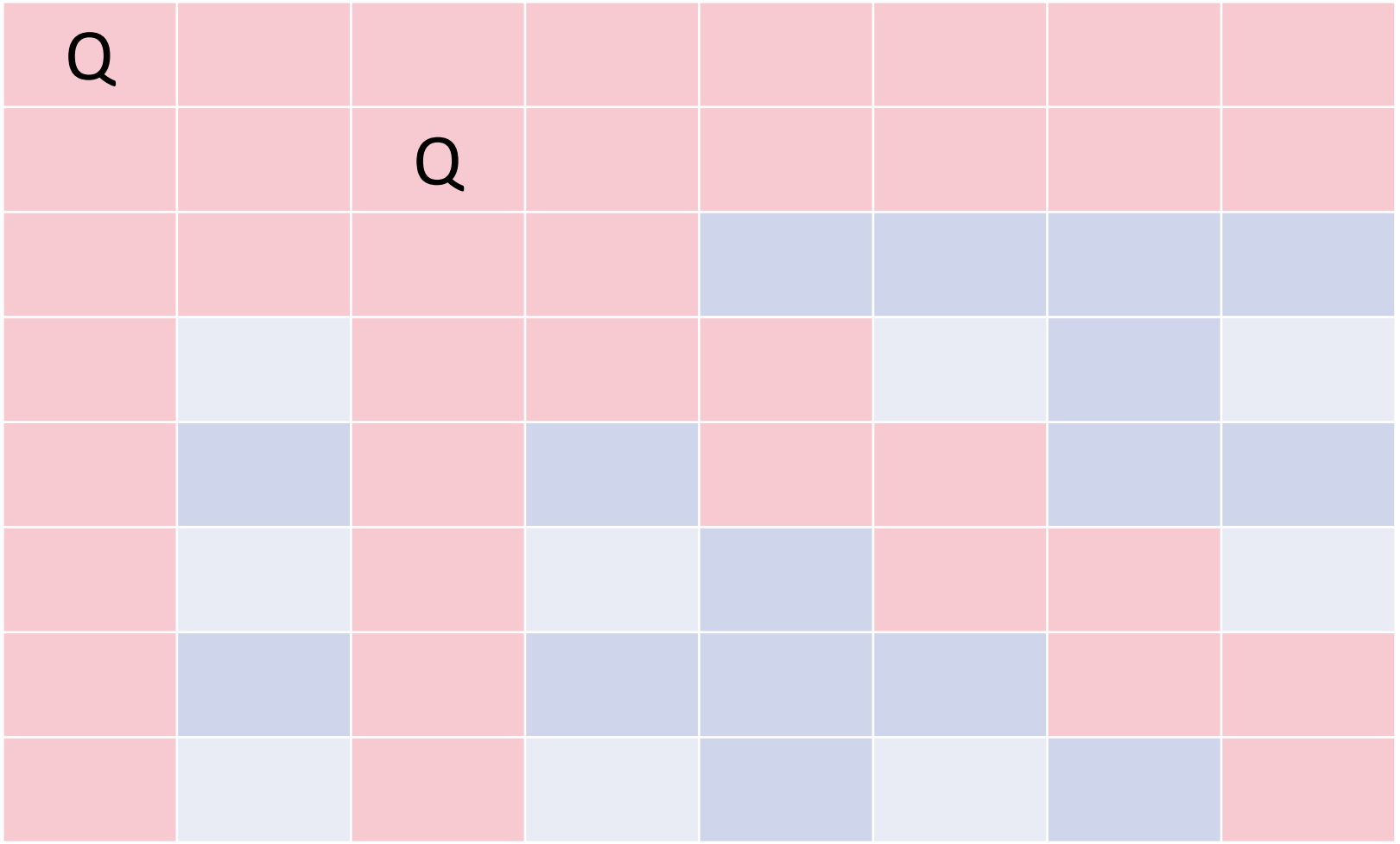
Stack

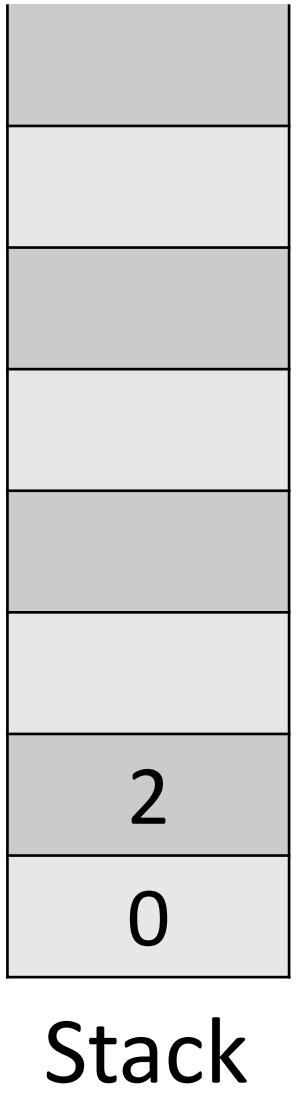
Example Code



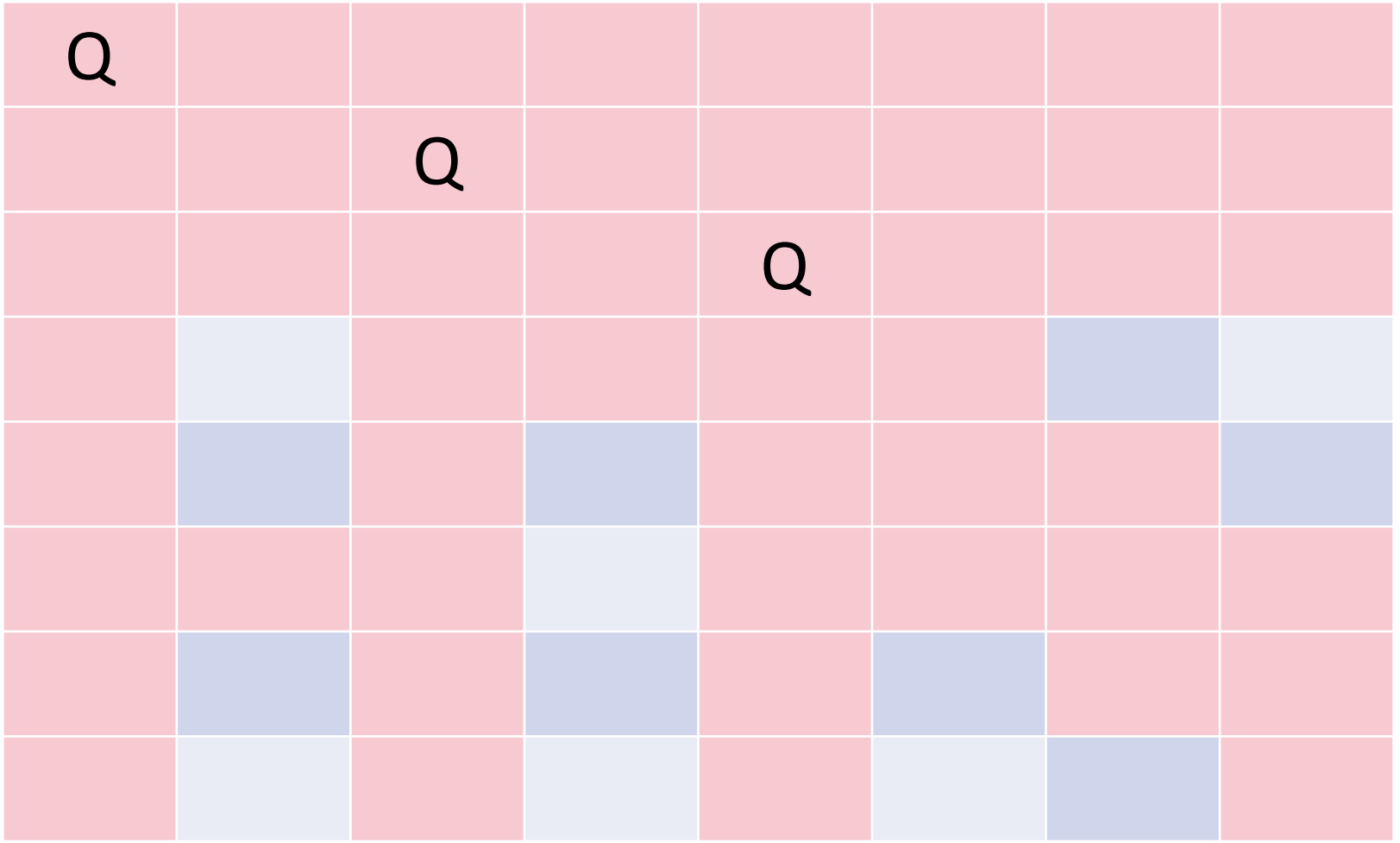


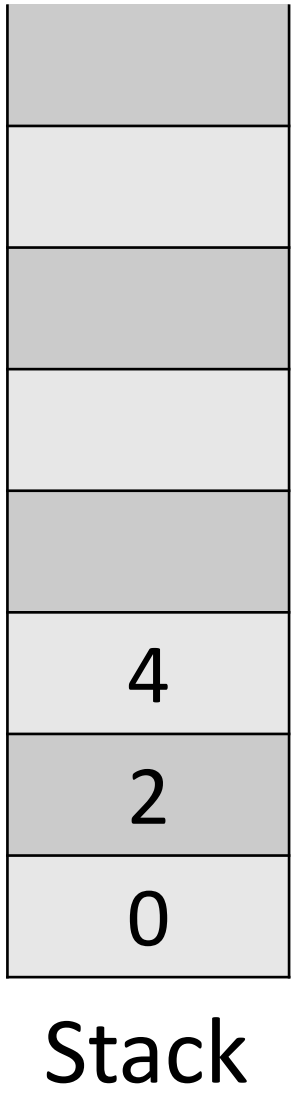
Example Code



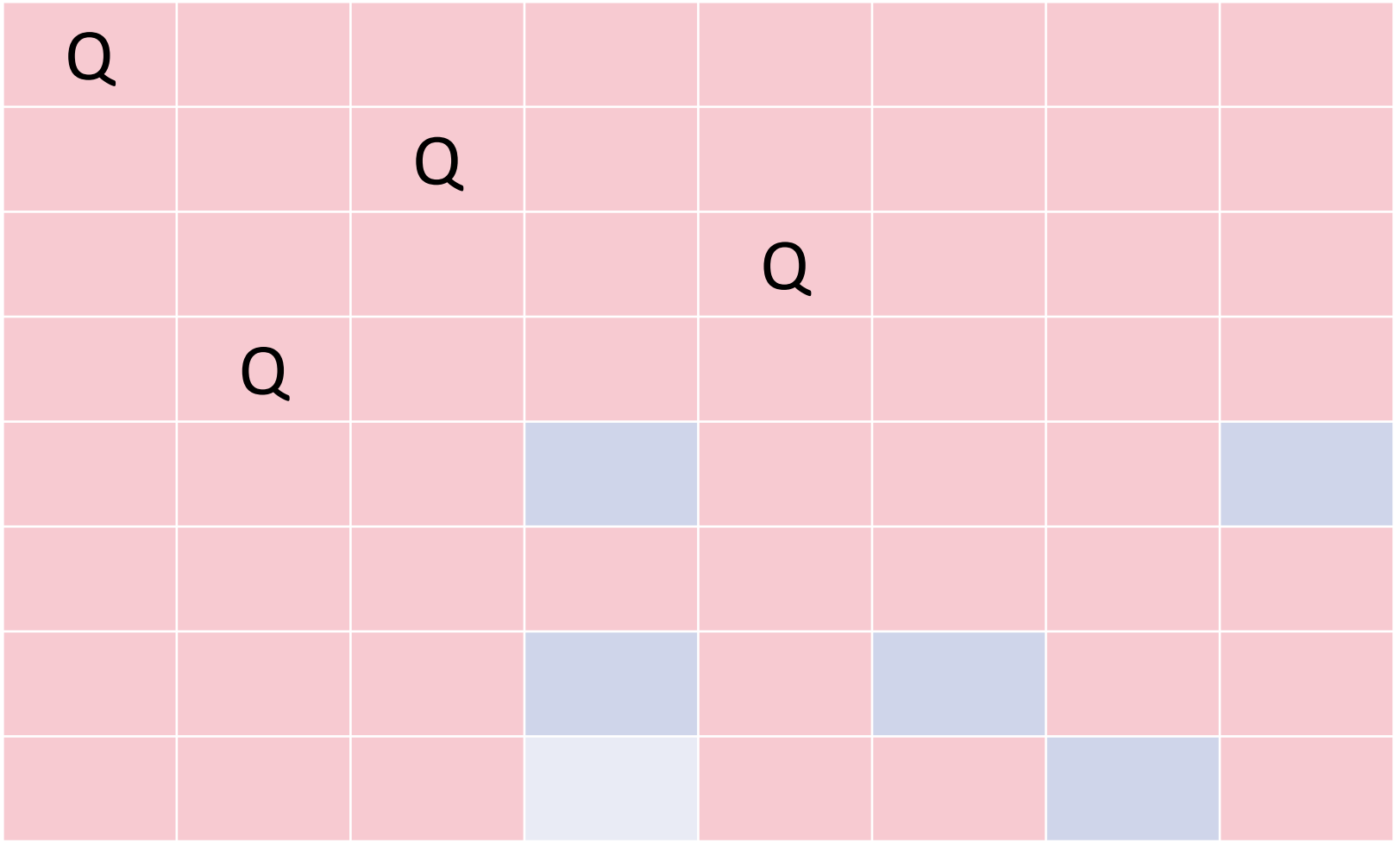


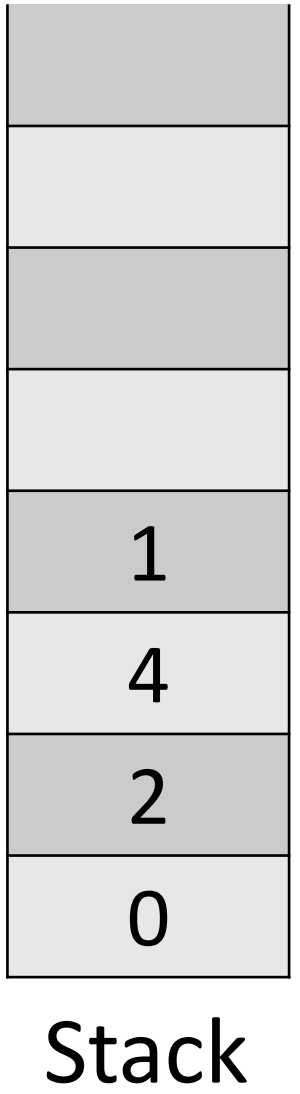
Example Code



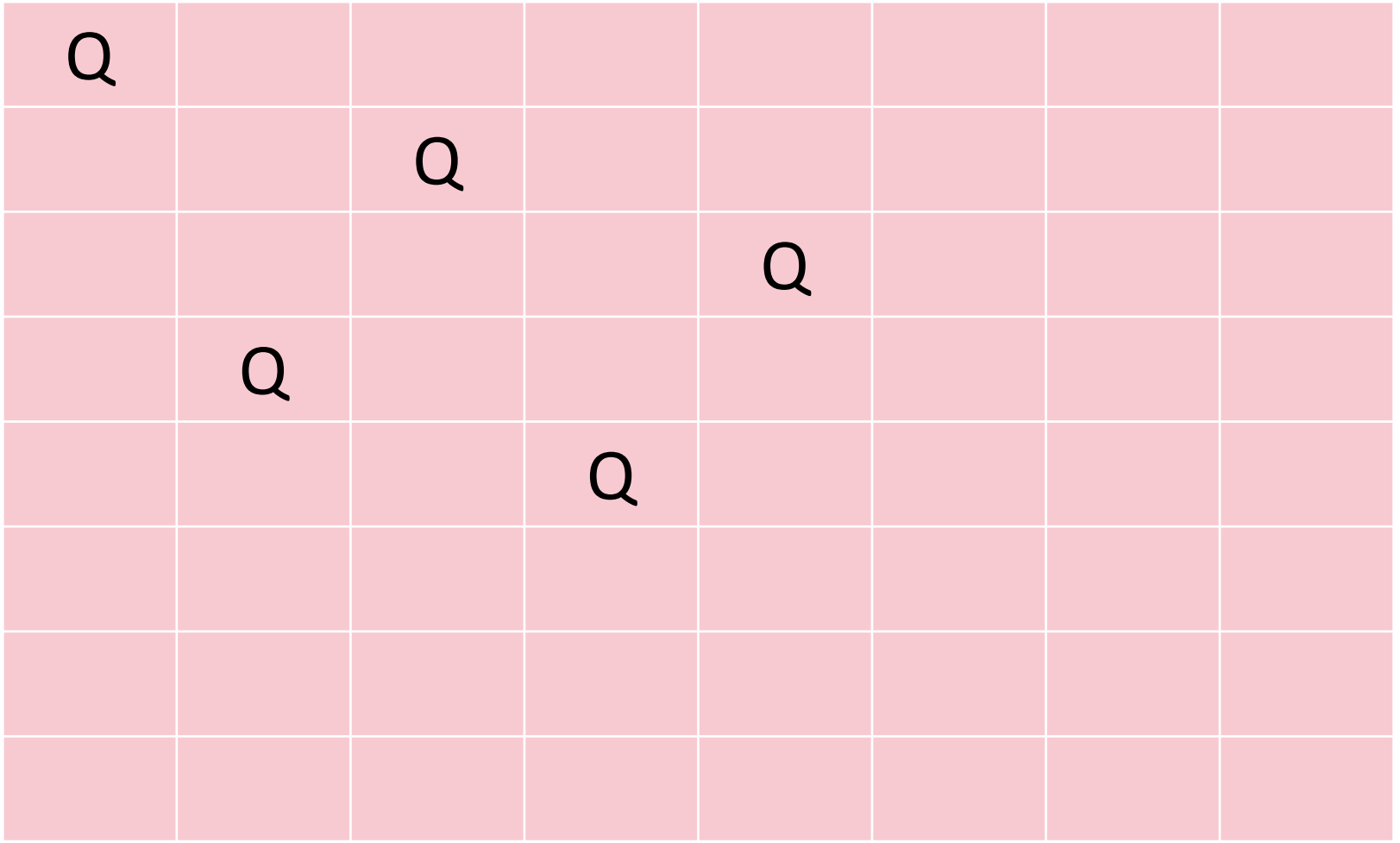


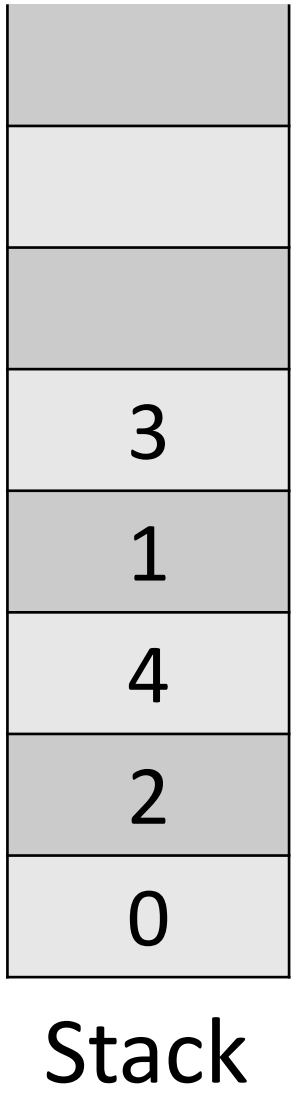
Example Code



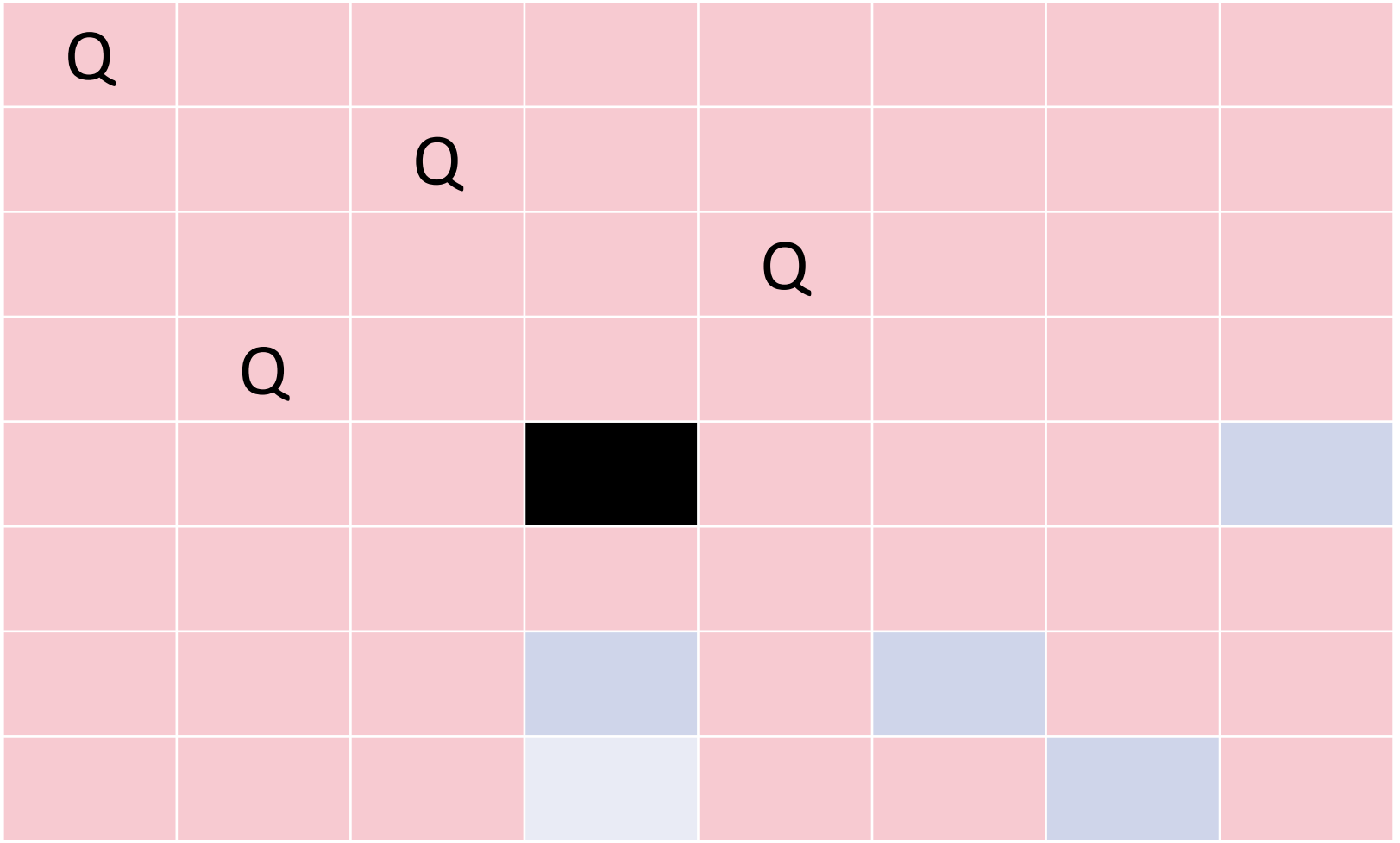


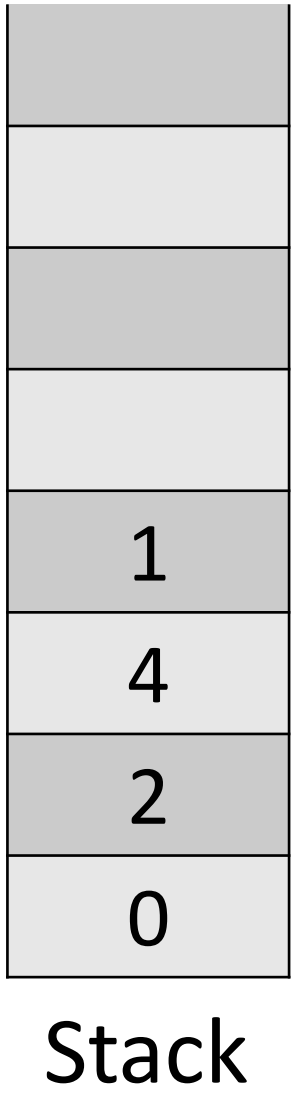
Example Code



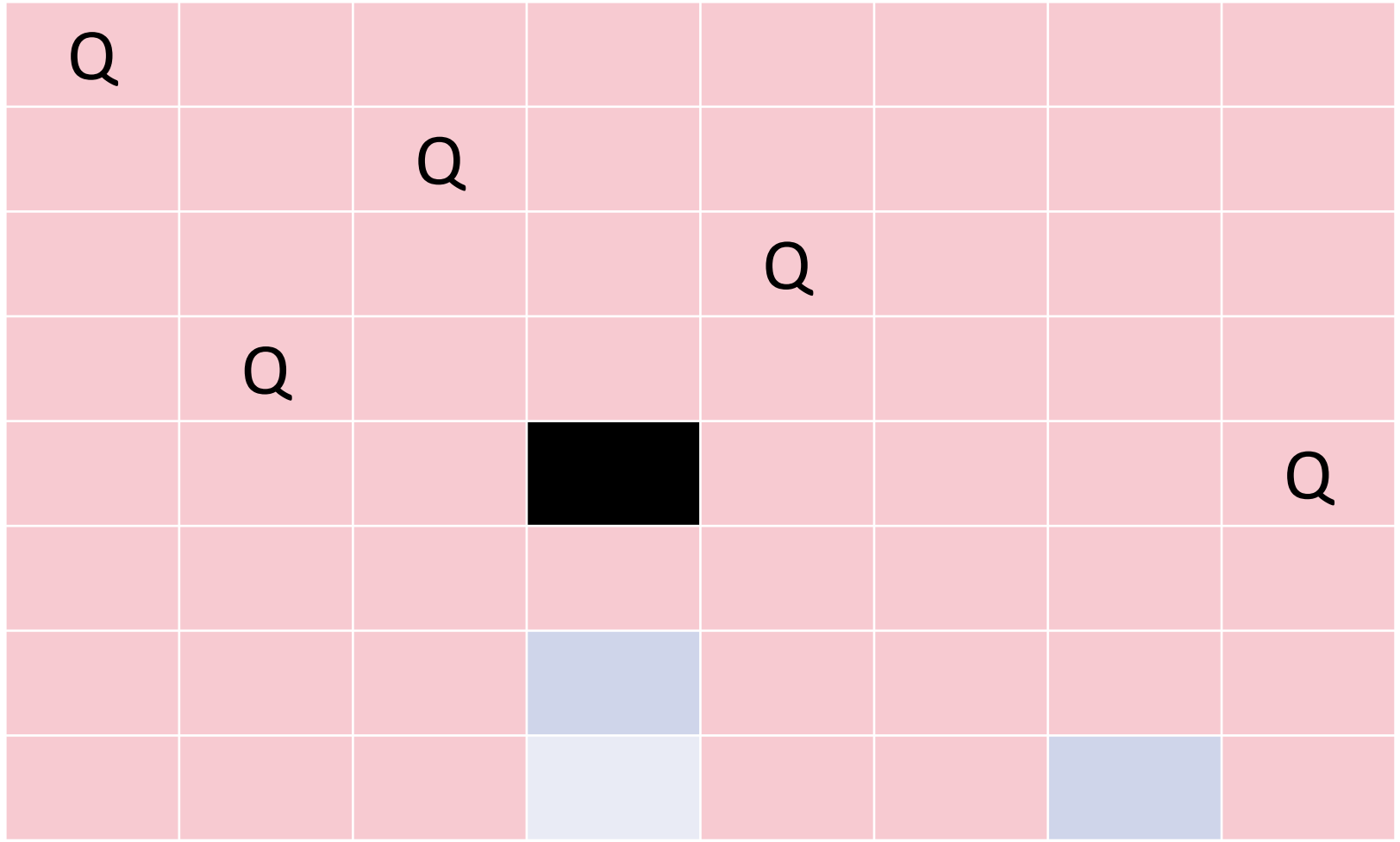


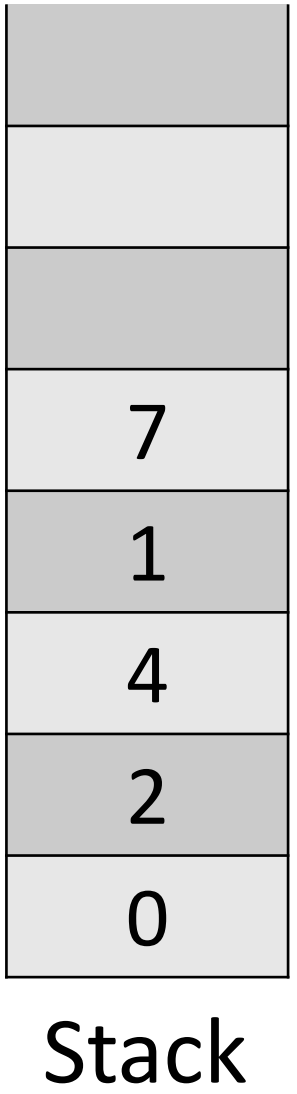
Example Code



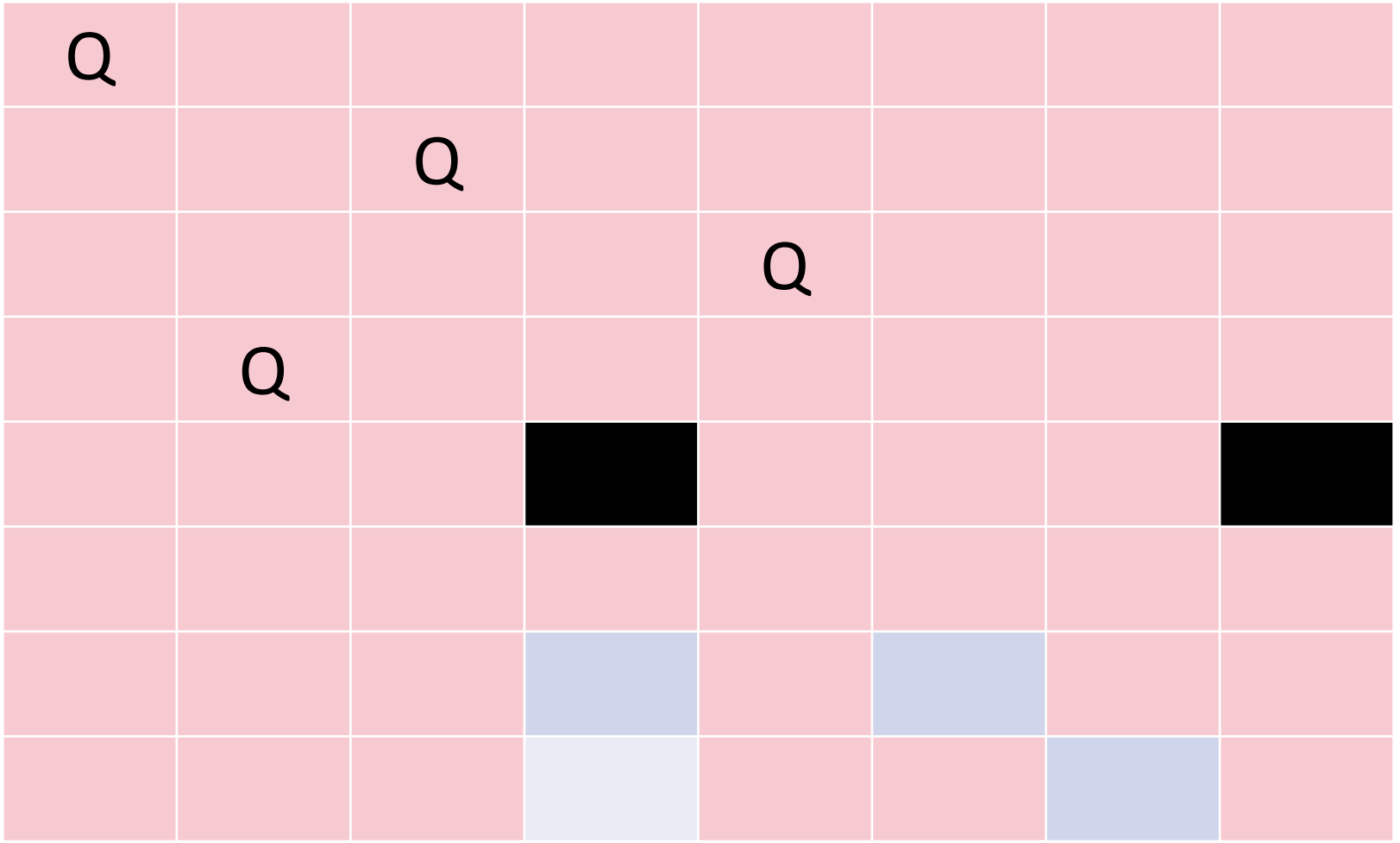


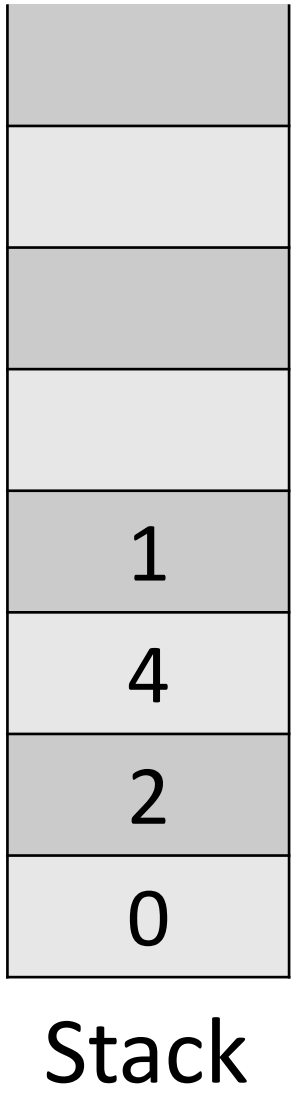
Example Code



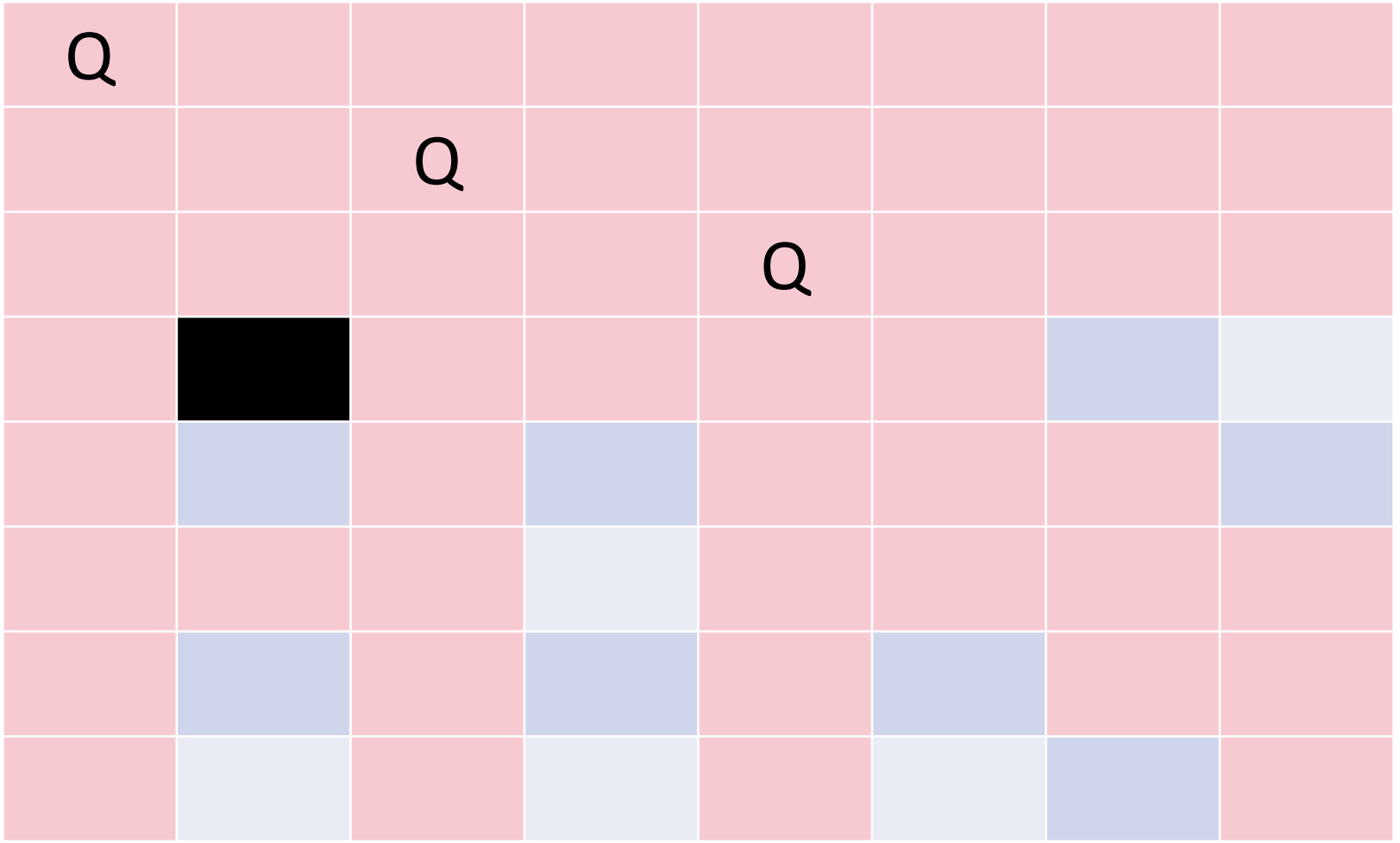


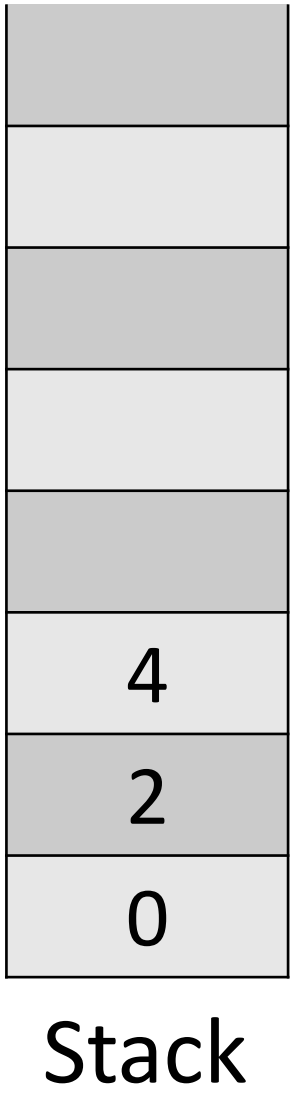
Example Code



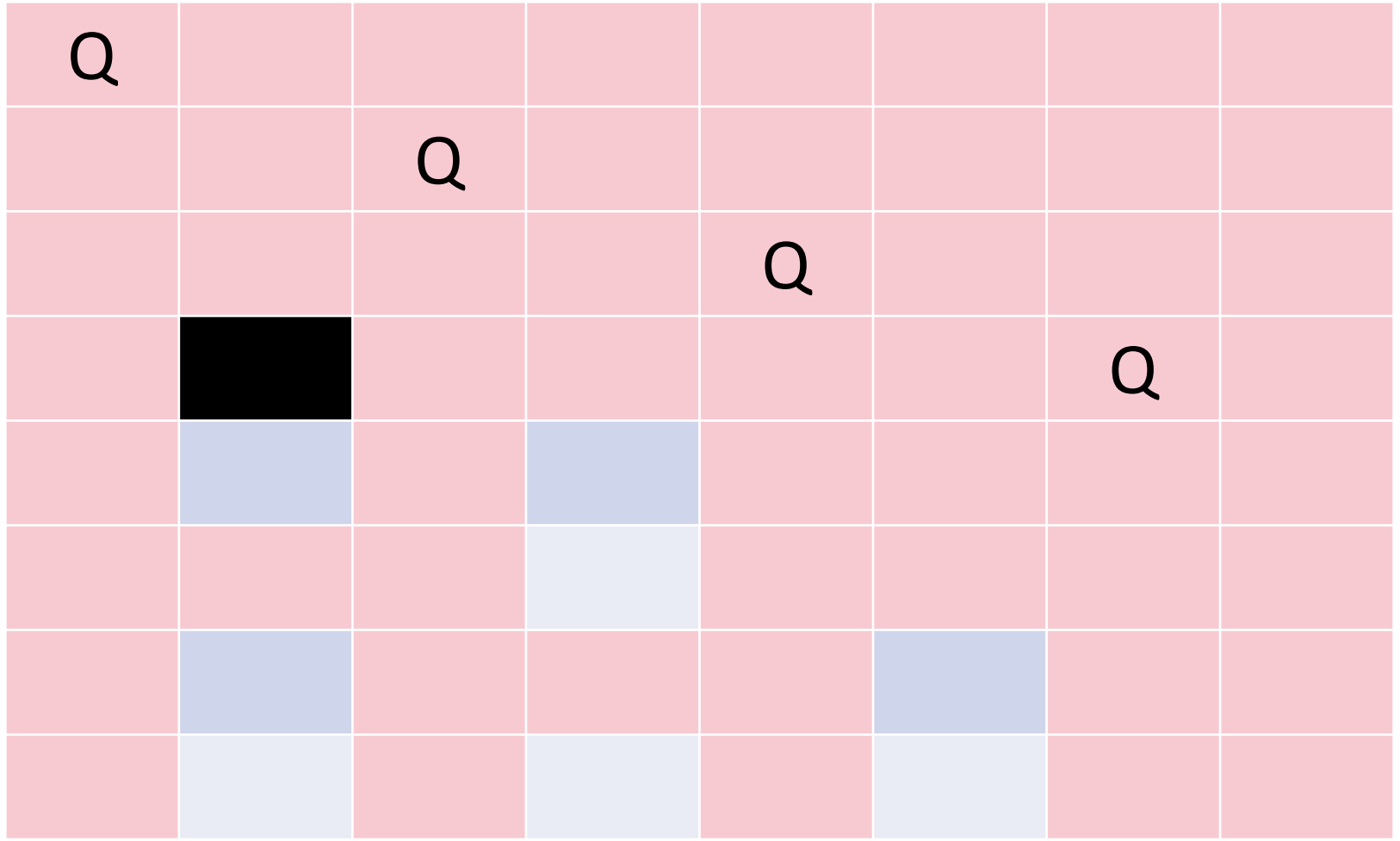


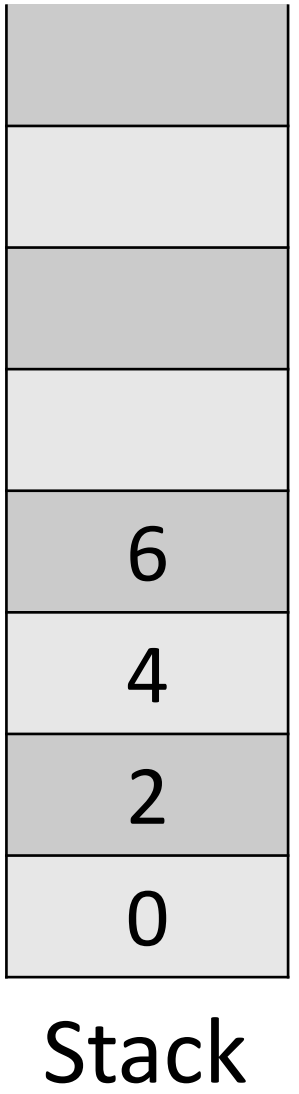
Example Code



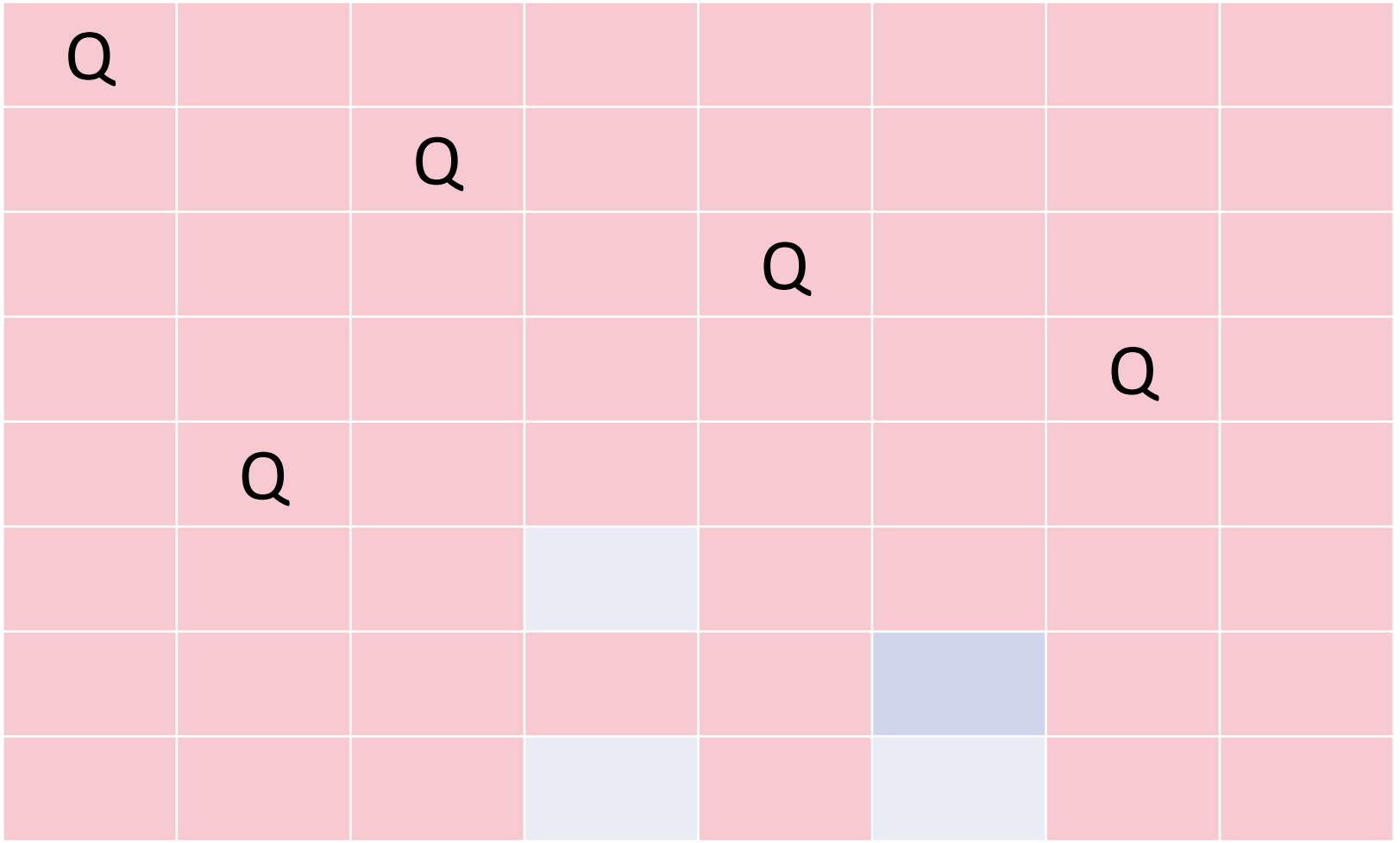


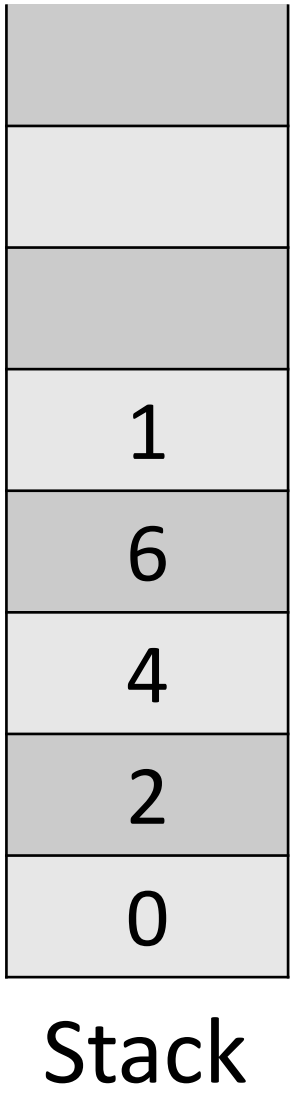
Example Code



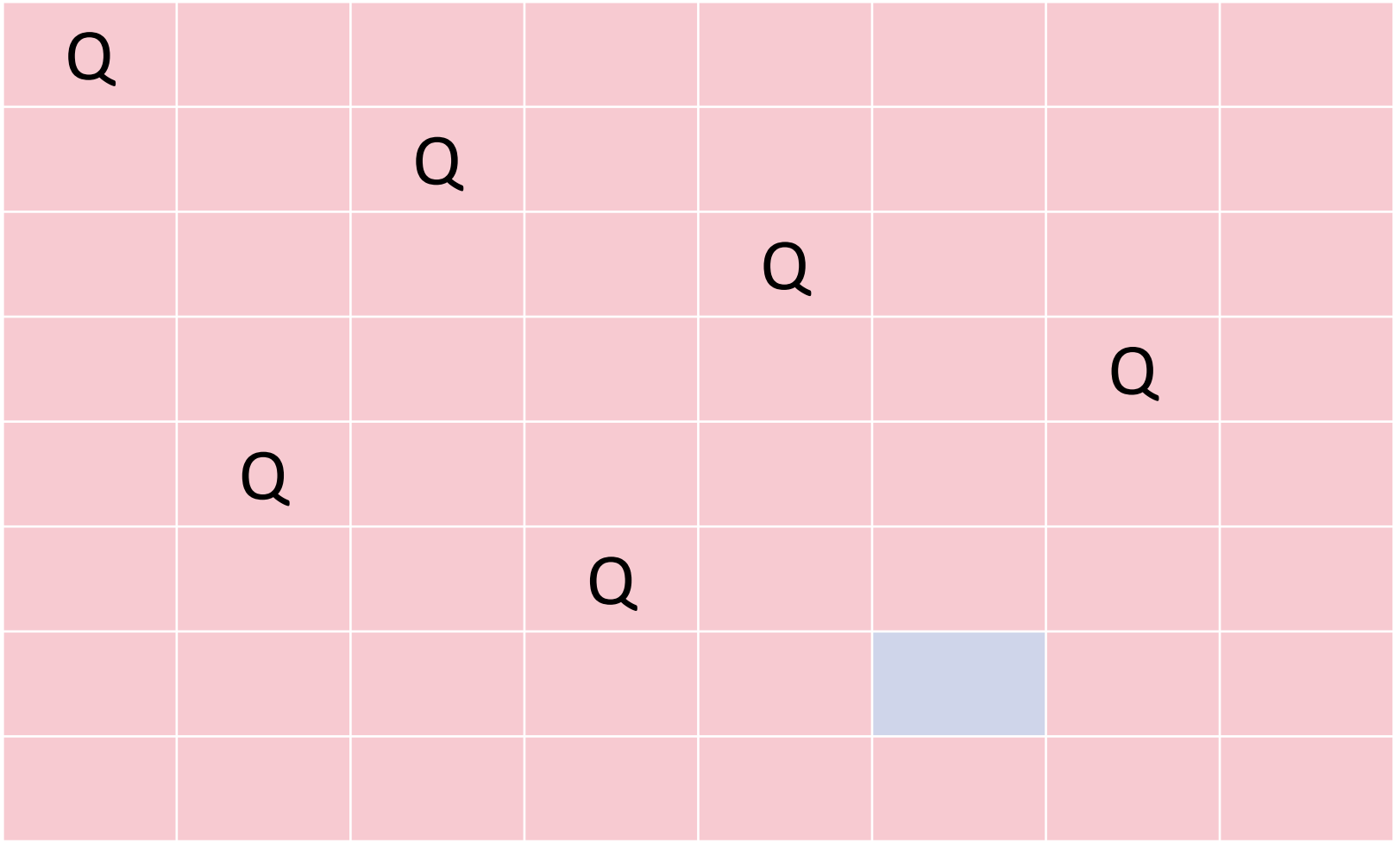


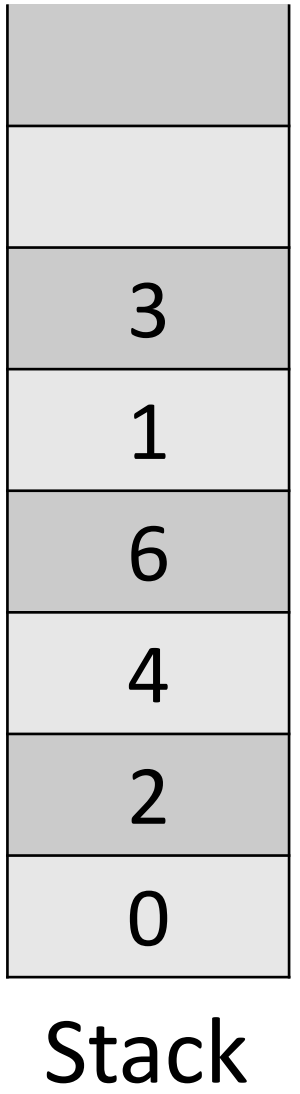
Example Code



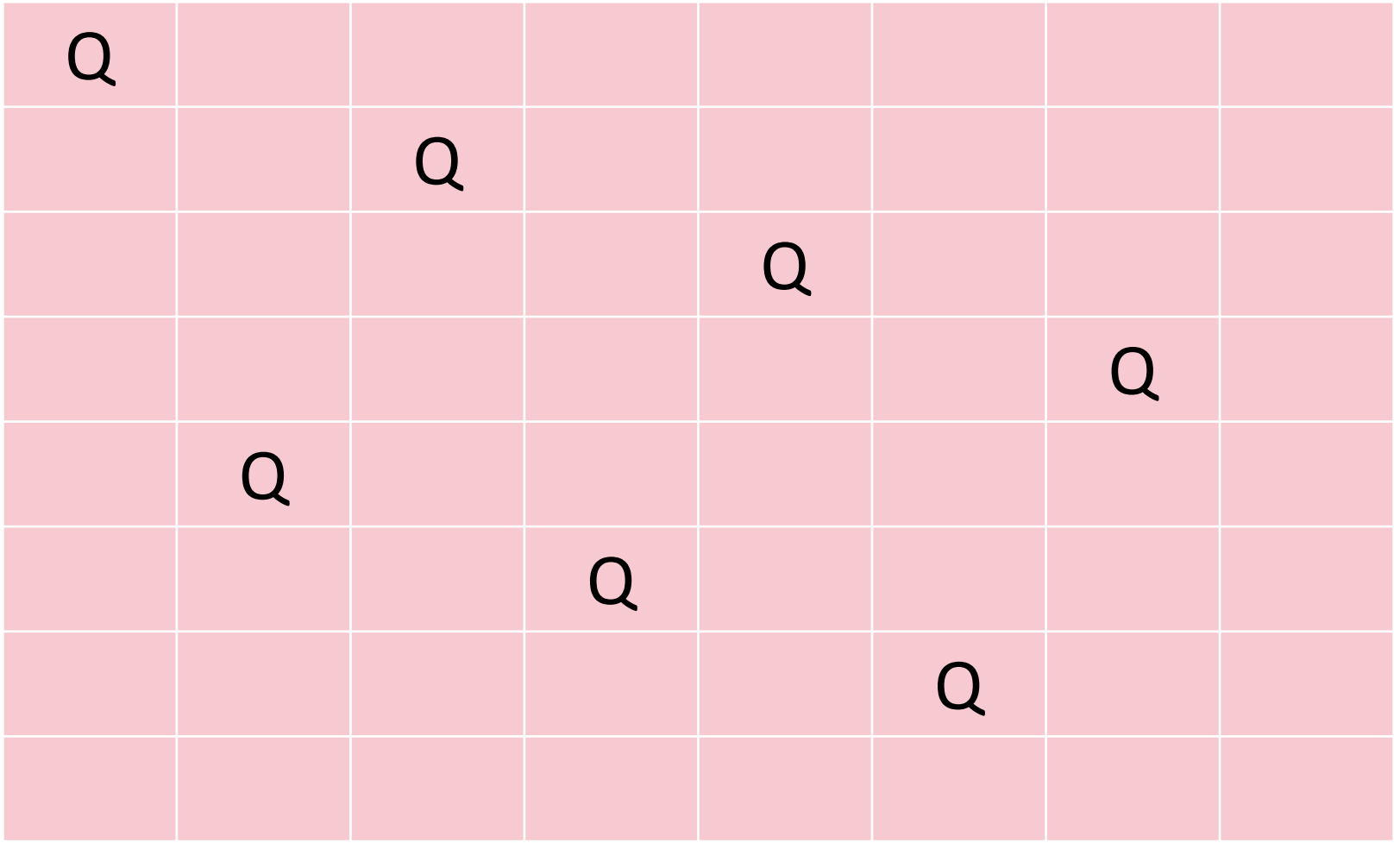


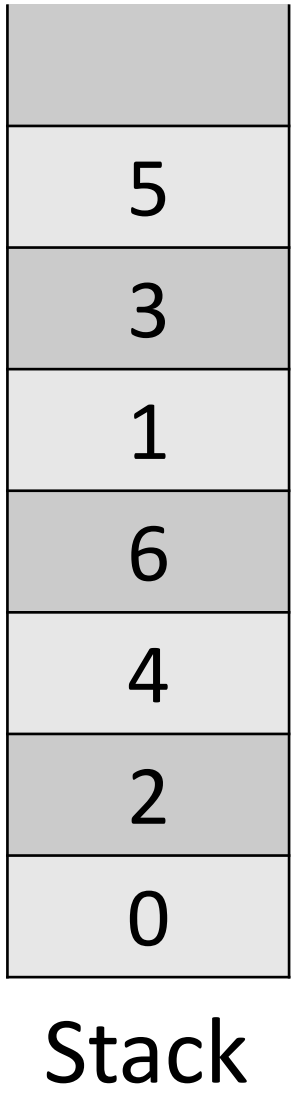
Example Code





Example Code

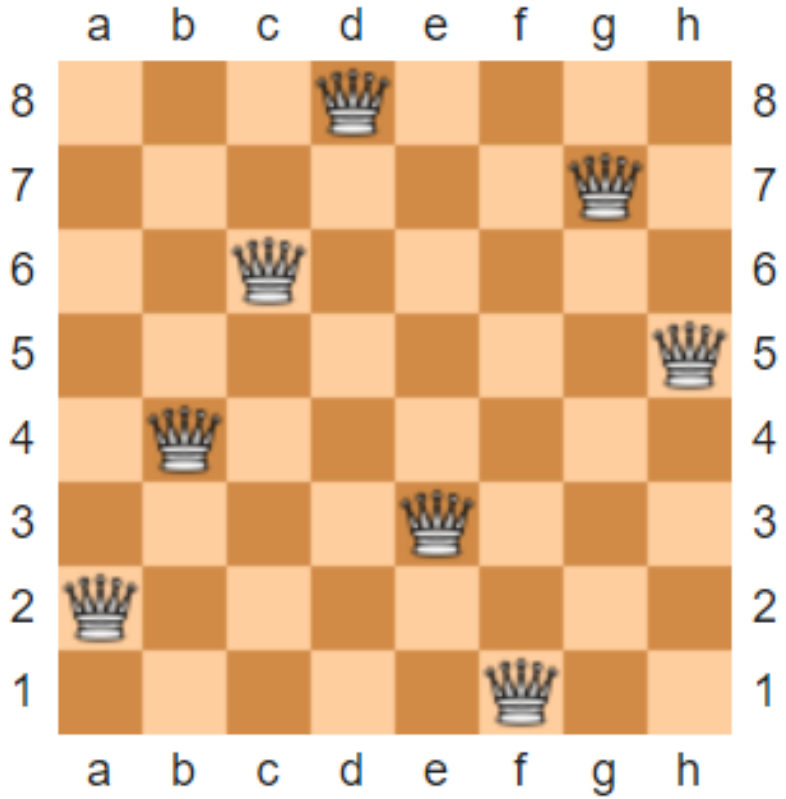




Example Code

Mission

讓使用者輸入 N，輸出 N 皇后問題的所有解答



Practice

Mission

Try LeetCode #51. N-Queens

The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

Ref : https://leetcode.com/problems/n-queens/

BFS 與 DFS 比較

BFS 與 DFS 比較

- 廣度優先搜尋 (BFS)
  1. 通常使用 Queue
  2. 適合最短路徑問題、列舉所有情形
  3. 較占用記憶體空間
- 深度優先搜尋 (DFS)
  1. 通常使用 Stack
  2. 適合搜尋環、分類問題

實戰練習

Practice

Mission

Try LeetCode #695. Max Area of Island

You are given an m x n binary matrix grid. An island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The area of an island is the number of cells with a value 1 in the island. Return the maximum area of an island in grid. If there is no island, return 0.

Ref : https://leetcode.com/problems/max-area-of-island/

Practice

Mission

Try LeetCode #1267. Count Servers that Communicate

You are given a map of a server center, represented as a m * n integer matrix grid, where 1 means that on that cell there is a server and 0 means that it is no server. Two servers are said to communicate if they are on the same row or on the same column.

Return the number of servers that communicate with any other server.

Ref : https://leetcode.com/problems/count-servers-that-communicate/