1. **在課程投影片 page 119 (標題為 Testing Strong Connectivity) 提到了一個演算法與 pseudocode 可以檢測 graph 的 strong connectivity。請證明其正確性。**

A: 設 graph G 為 Strong Connected，即假設任意點 s 可以與另一點 v 相通，而 v 有兩種情況：

1.可直接與 s 互通

2.可透過其他 nodes 形成的路徑 p 與 s 相通，

若將 G 的方向性倒轉，情況 1 不受影響仍可互通，情況 2 之下 s 可以從路徑 p 抵達 v 而 v 可直接通到 s，故在 Strong Connected 的情況下，只要測試任意一點 s 在 G 以及方向性倒轉的 G’ BFS 是否都能到達所有點即可驗證。

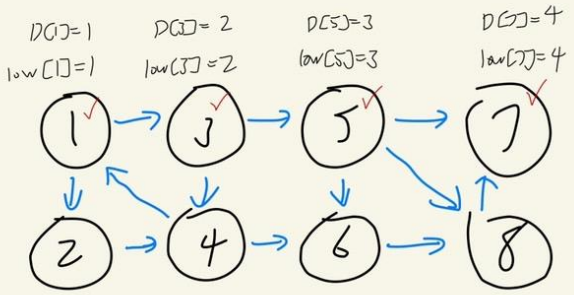
2. **Tarjan's algorithm (https://tinyurl.com/4me6jk6w) 與 Kosaraju's algorithm (https://tinyurl.com/yc3rppn6) 是可以測試 strongly connected components 的兩個演算法。請分別敘述兩個演算法怎麼運作，並且連環圖的方式 (如課程投影片 page 98 那樣的分解動作) 在一個你選定的 graph 上 (|V|≥8、|E|≥12) 示範怎麼進行這兩個演算法。**

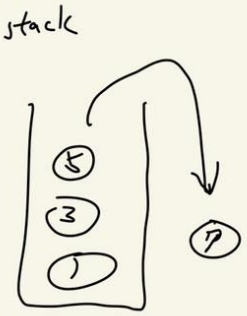
A:

Tarjan:

1. 為當前 node u 設定次序編號與 low 初值，透過一個漸增的 index 變數。
2. 將 u 丟入 stack。
3. 列舉 u 的後續 node，如果另一端的 node v 未被訪問過則遞迴 Tarjan(v)，並將 low[u]調成 min(low[u],low[v])，如果 v 已在 stack 裡，則一樣把 low[u]調成 min(low[u],low[v])。
4. 如果 node 的次序編號與 low 值相同，則從 stack 裡 pop 出 node 直到把 u 自己給 pop 出，這一過程出 pop 出的 node(s)組成一個 strongly connected components。

1、



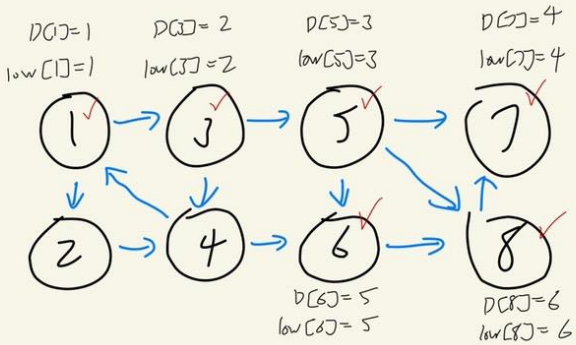


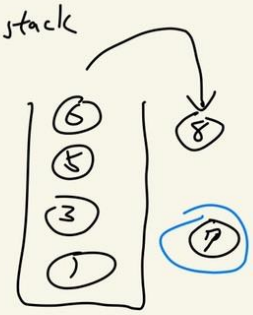
走訪了 1→3→5→7

發現7無後續 node

pop 出自己

2、



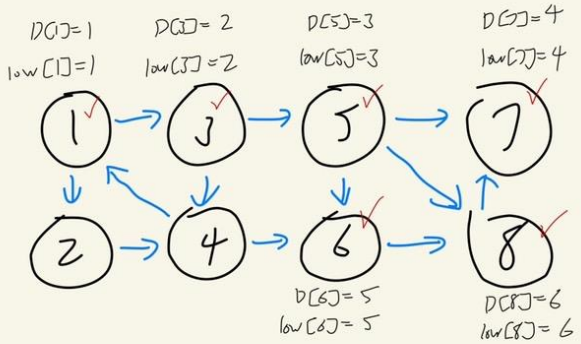


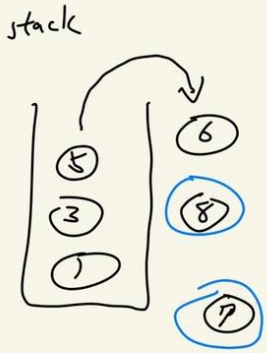
退回5 走訪5→6→8

發現8無後續未走訪 node

pop 出自己

3、



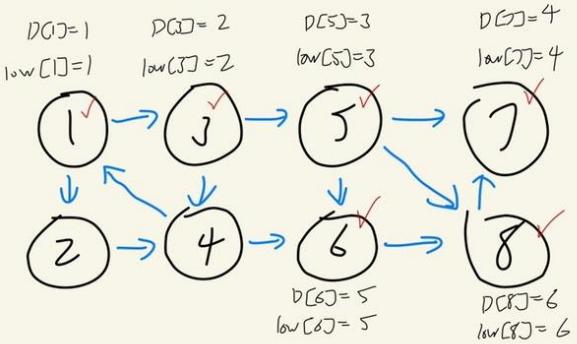


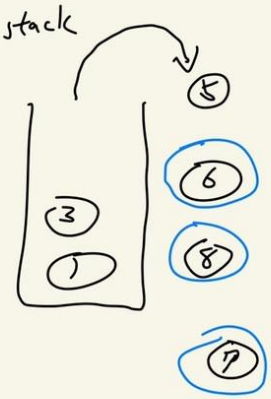
退回6

發現6無後續未走訪 node

pop 出自己

4、



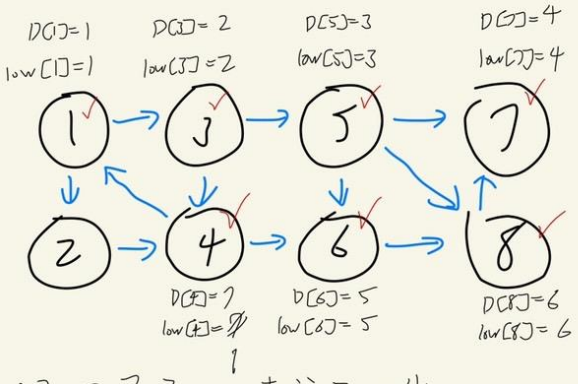


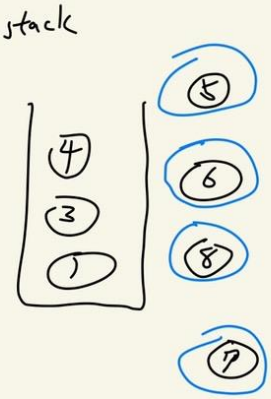
退回5

發現5無後續未走訪 node

pop 出自己

5、





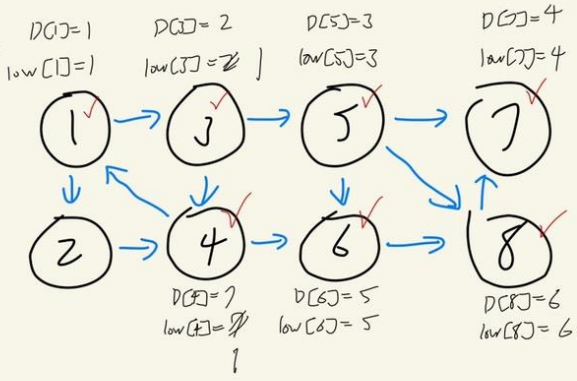
退回了3　走訪了→4

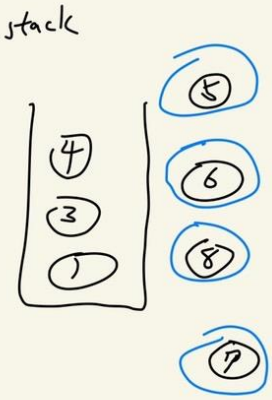
因為4可到達仍在 stack 內的1

low[4] = min[low[1]=1, low[4]=7] = 1

D[4] ≠ low[4]　不需 pop

6、



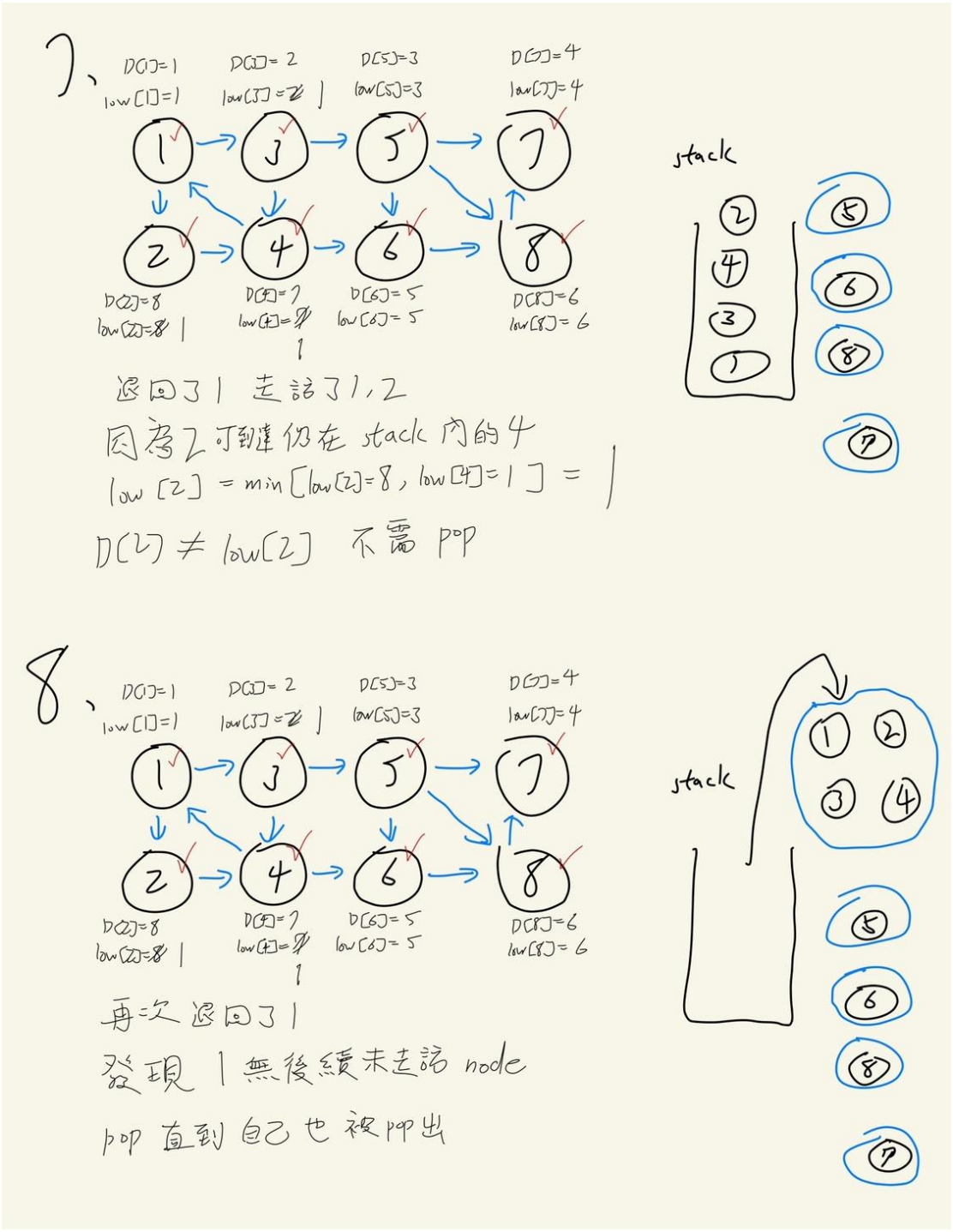


再次退回了3

因為3可到達仍在 stack 內的4

low[3] = min[low[3]=2, low[4]=1] = 1

D[3] ≠ low[3]　不需 pop

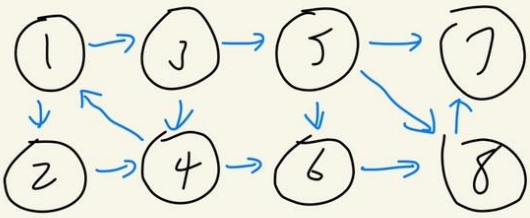


得到 strongly connected components 為{1,2,3,4},{5},{6},{7},{8}。

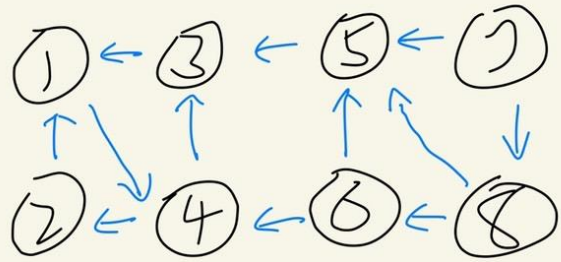
Kosaraju:

1. 對 graph G 的方向性逆轉，得到 G’ 。
2. 對 G 用 DFS，給每個 node 各自的離開時間。
3. 對 G’ 按照最晚離開的排序開始對未走訪的 node 進行 DFS，每一次 DFS 存取的所有 node(s)組成一個 strongly connected components。

1、G:

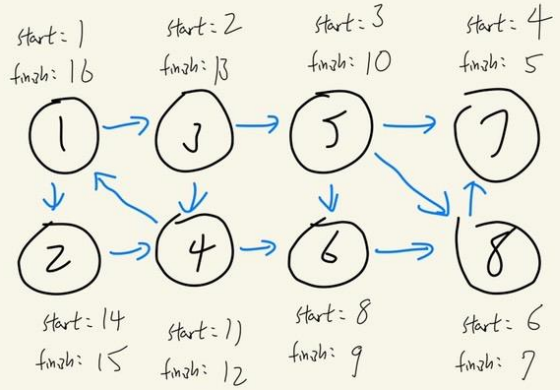


G':



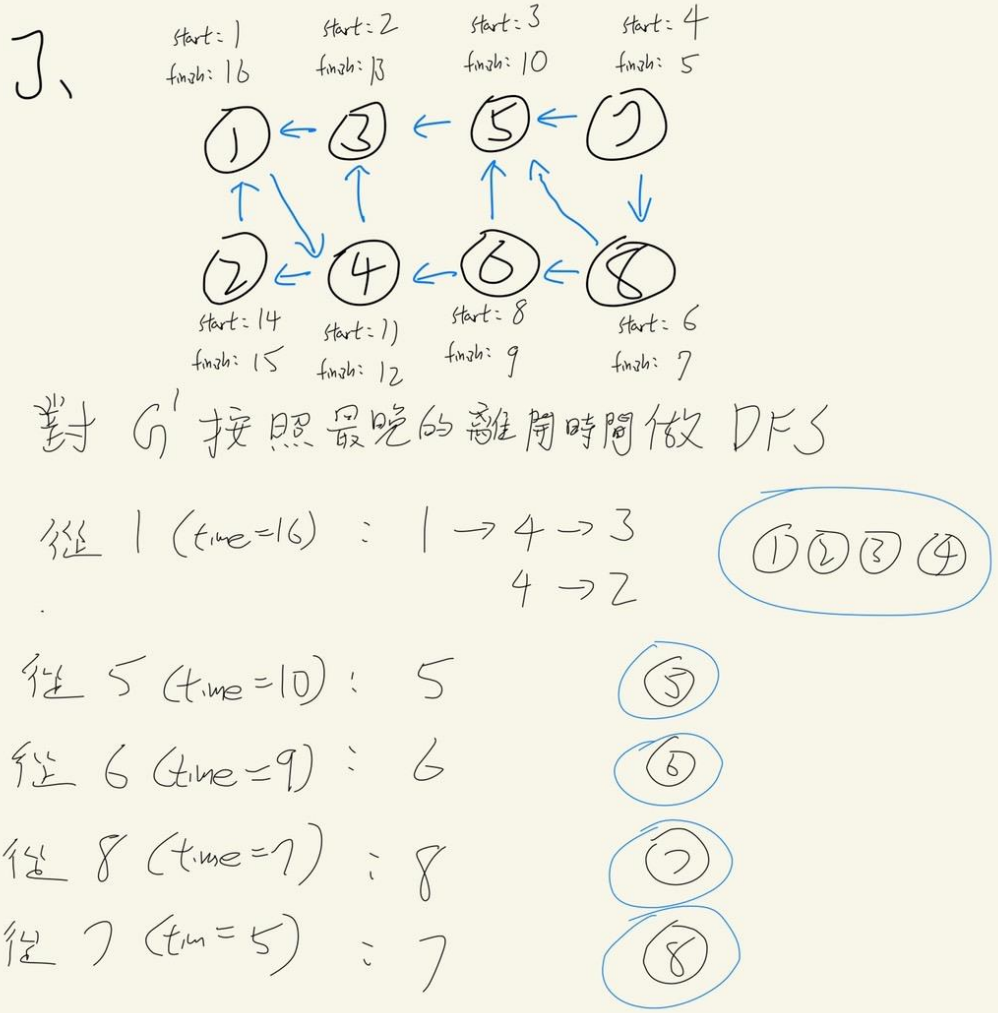
得到逆向圖 G'

2、



對 G 用 DFS，標記進入與離開時間

3、



得到 strongly connected components 為{1,2,3,4},{5},{6},{7},{8}。