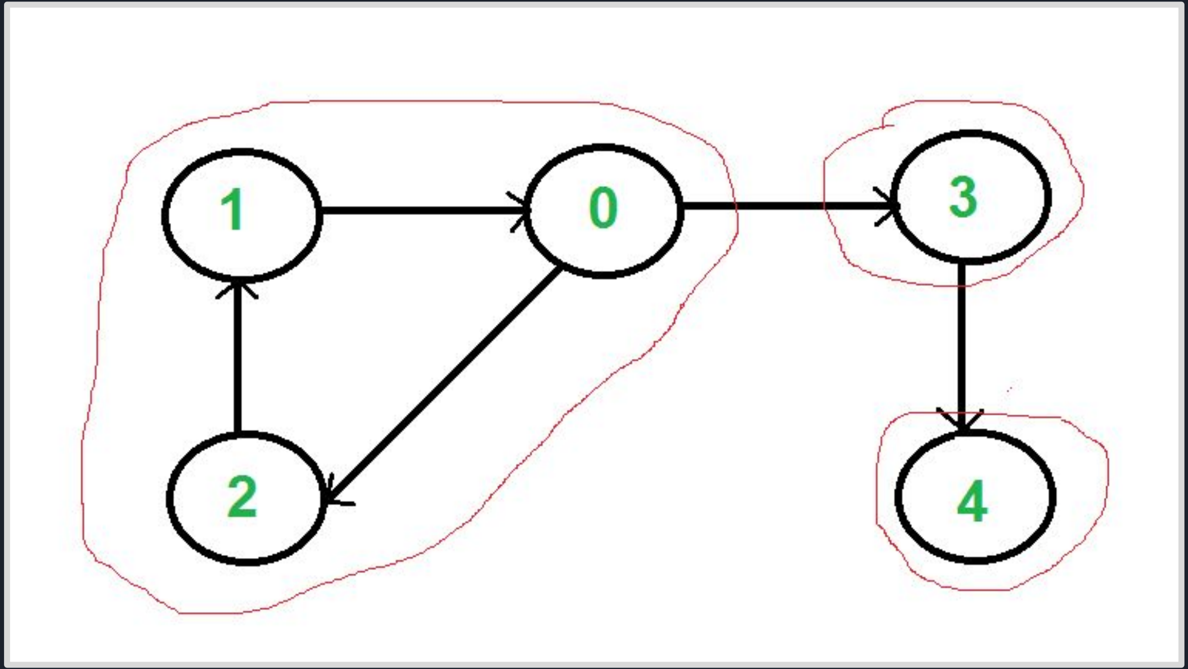
Strongly Connected Components

Darren Peng

What is a strongly connected component?

A directed graph is strongly connected if there is a path between all pairs of vertices.

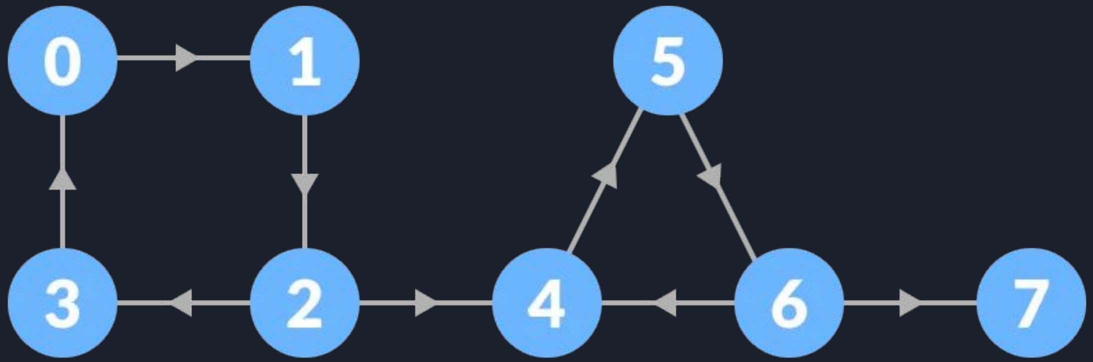
Note that a single node can be a SCC if it doesn't form a bigger SCC with other nodes.



Kosaraju’s Algorithm

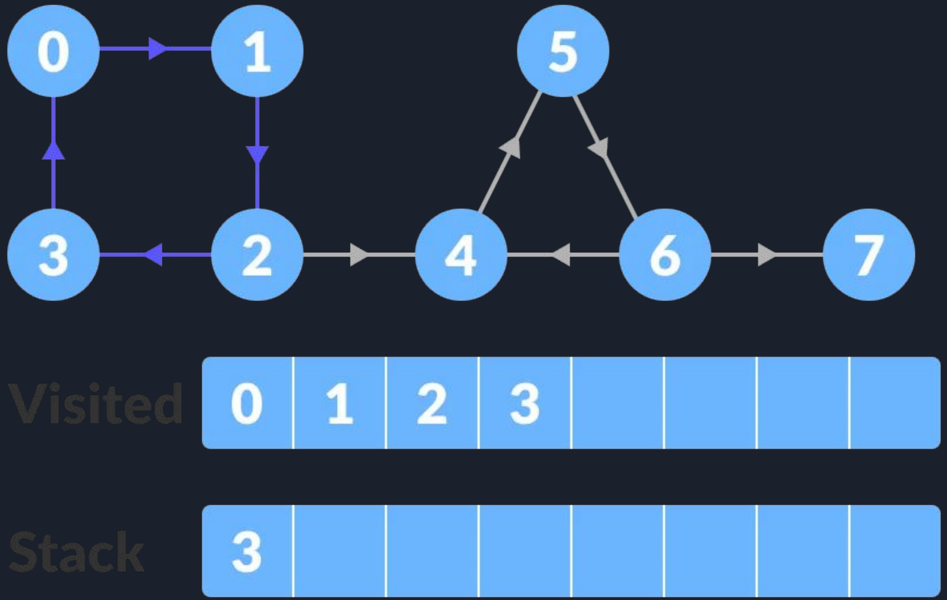
Step 1: Perform DFS traversal of the graph. While traversing, push nodes to stack when exhausted all outgoing edges.
Step 2: Find the transpose graph by reversing the edges.
Step 3: Pop nodes one by one from the stack and again to DFS on the modified graph.

Time complexity: O(V+E)

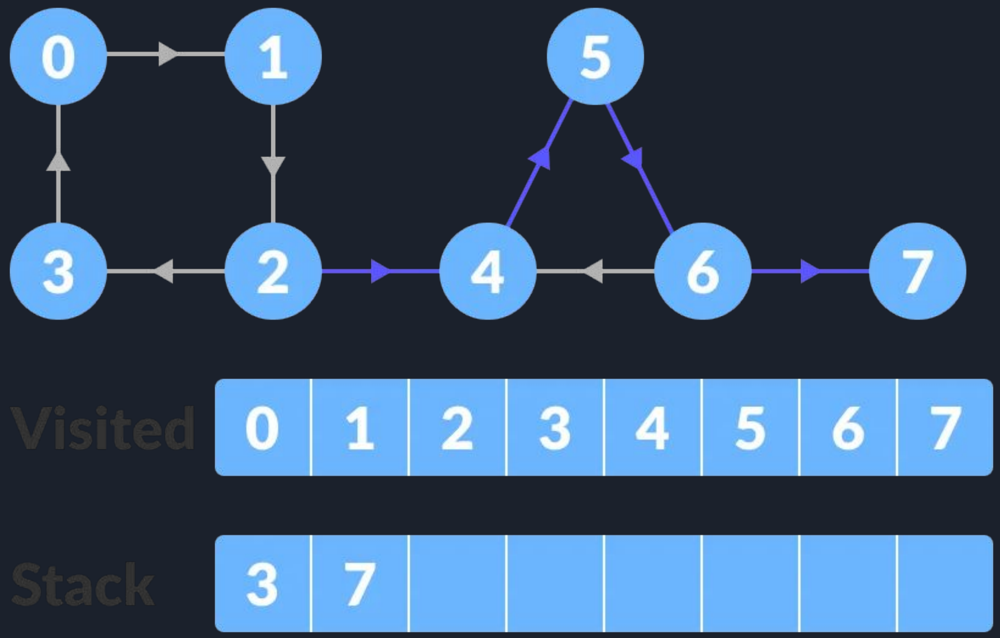


Example step 1

Push node 3 because we've exhausted all its edges. Can't revisit visited nodes



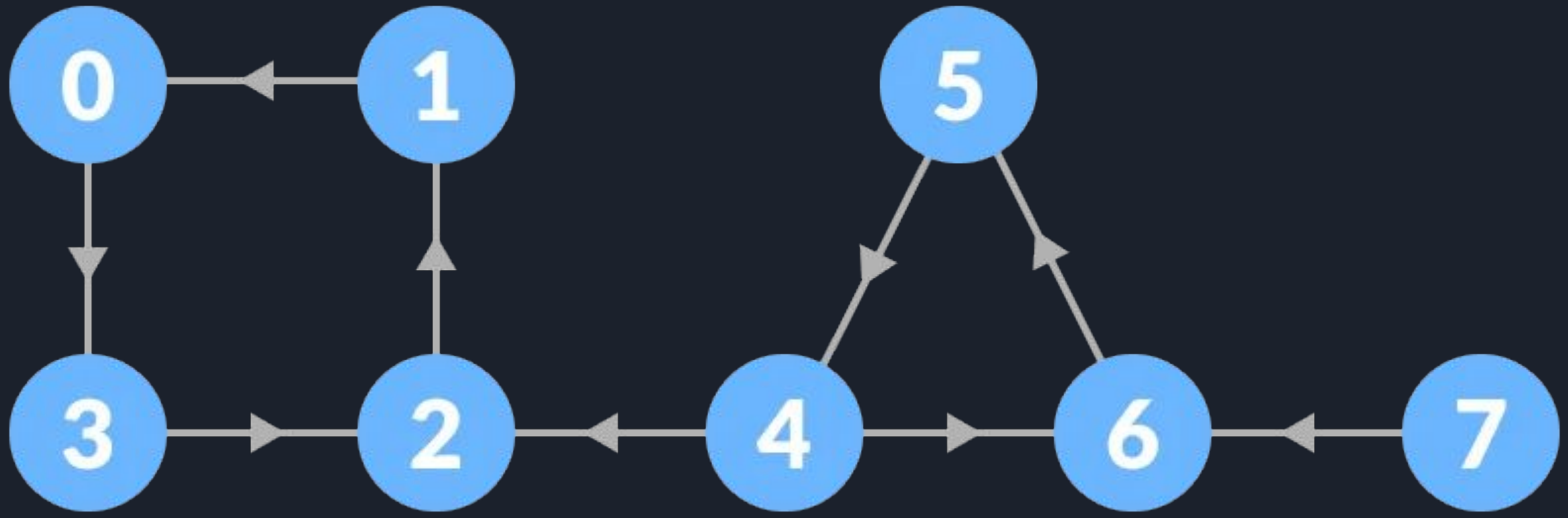
Example step 1



Example Step 1

Visited	0	1	2	3	4	5	6	7
Stack	3	7	6	5	4	2	1	0

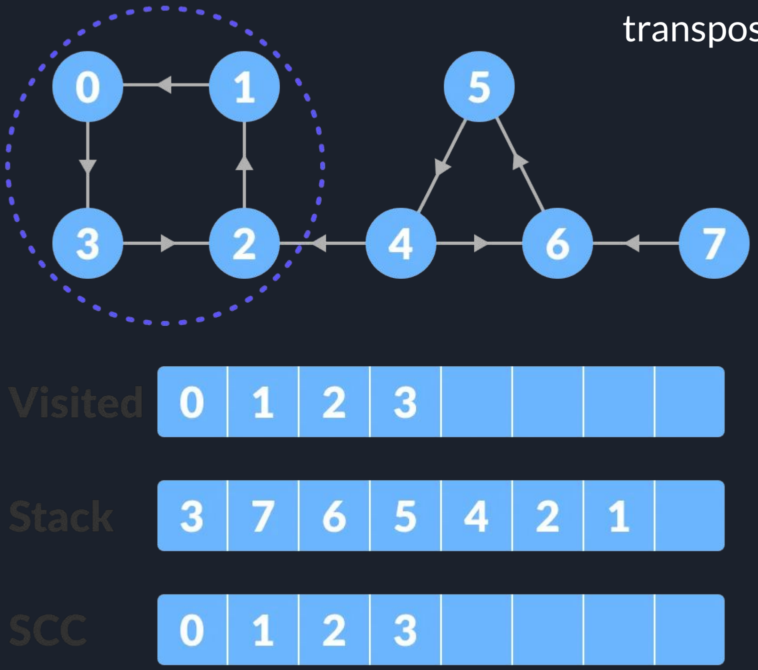
Example step 2



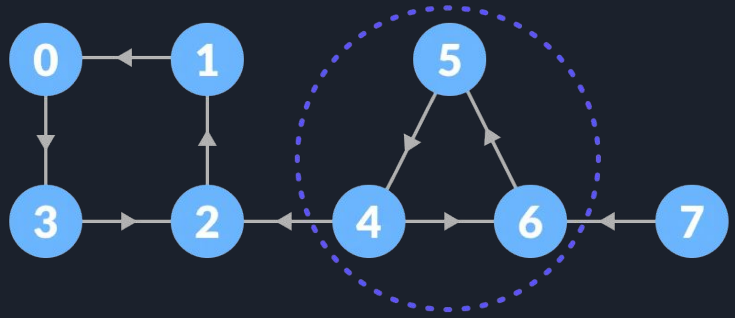
Example step 3

1) Pop 0 out of stack
2) DFS traversal of all nodes we can reach
3) Stop when we reach a visited node
4) Skip all visited nodes when we pop out of the stack

Note that this is the transposed graph.

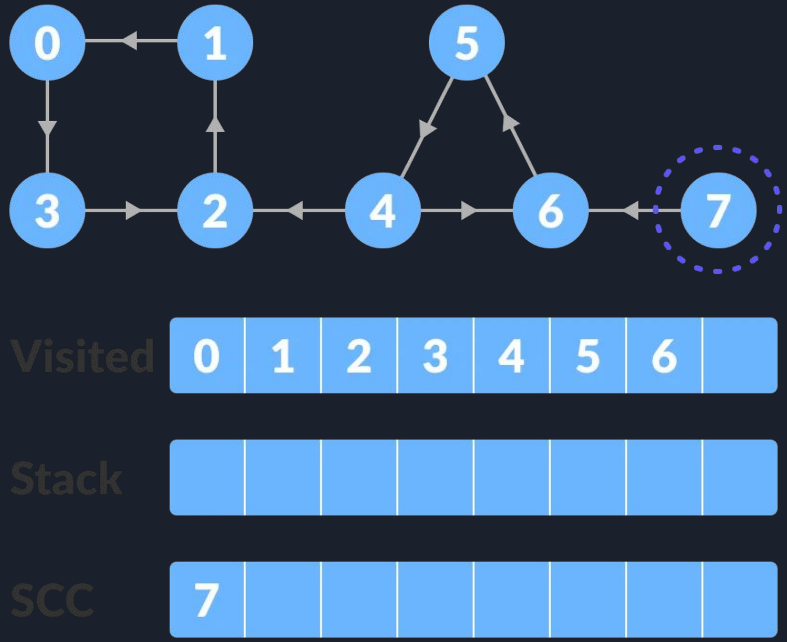


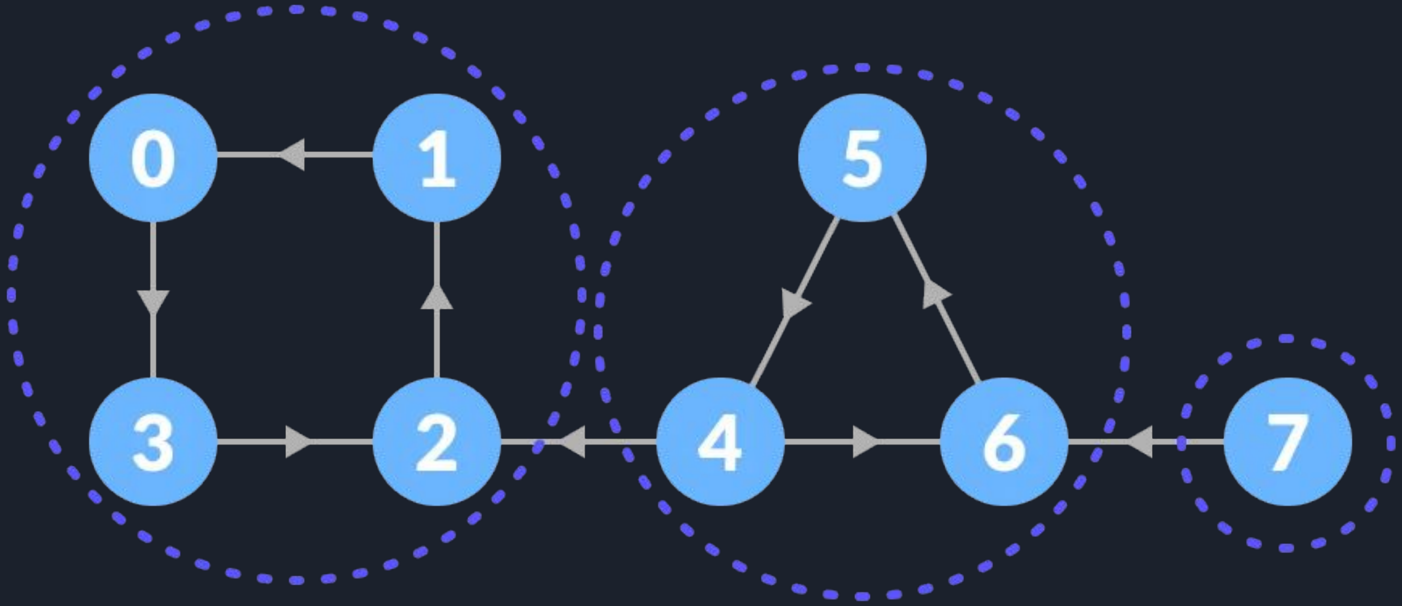
Example step 3



Visited	0	1	2	3	4	5	6	
Stack	3	7	6	5				
SCC	4	5	6					

Example step 3




0
1
5
3
2
4
6
7

Code

Step 1: Perform DFS traversal of the graph. While traversing, push nodes to stack when exhausted all outgoing edges.
Step 2: Find the transpose graph by reversing the edges.
Step 3: Pop nodes one by one from the stack and again to DFS on the modified graph.

```
void Kosaraju() {
  for (int i = 0; i < n; i++)
    if (!visited[i]) dfs_1(i);

  for (int i = 0; i < n; i++)
    visited[i] = false;

  while (!S.empty()) {
    int v = S.top();
    S.pop();
    if (!visited[v]) {
      printf("Component %d: ", numComponents);
      dfs_2(v);
      numComponents++;
      printf("\n");
    }
  }
}
```

```
void dfs_1(int x) {
  visited[x] = true;
  for (int i = 0; i < g[x].adj.size(); i++) {
    if (!visited[g[x].adj[i]]) dfs_1(g[x].adj[i]);
  }
  S.push(x);
}
```

```
void dfs_2(int x) {
  printf("%d ", x);
  component[x] = numComponents;
  components[numComponents].push_back(x);
  visited[x] = true;
  for (int i = 0; i < g[x].rev_adj.size(); i++) {
    if (!visited[g[x].rev_adj[i]]) dfs_2(g[x].rev_adj[i]);
  }
}
```

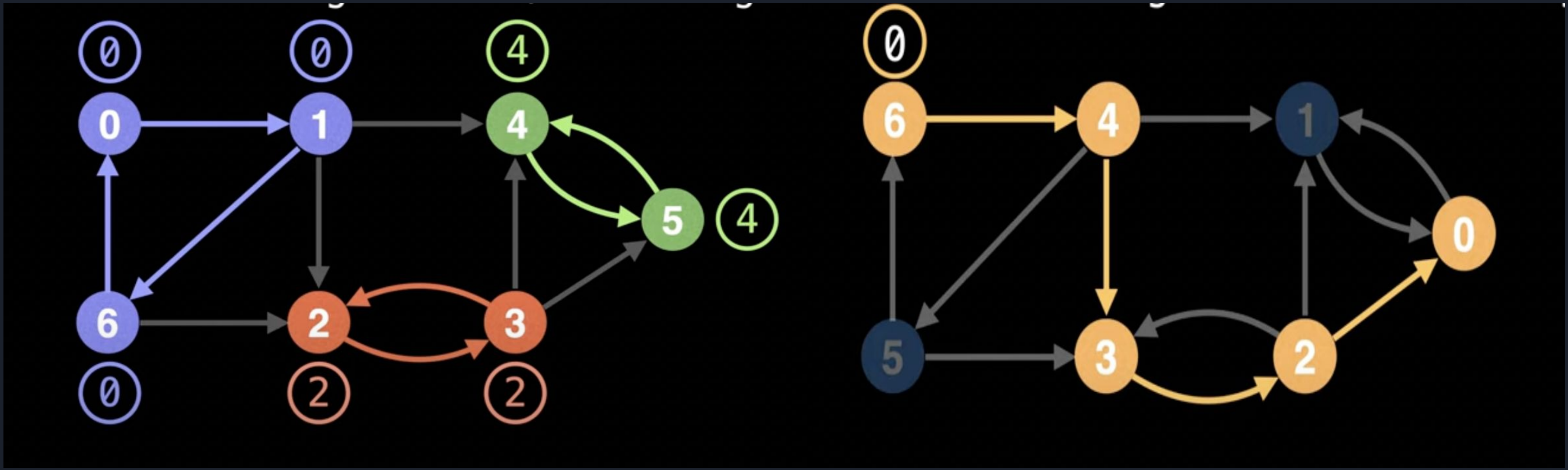
Tarjan’s Algorithm

- As opposed to Kosaraju’s algorithm, Tarjan’s algorithm only needs one DFS traversal.
- Low-link values: The low-link value of a node is the smallest node ID reachable from that node when doing DFS, including itself.
- However, there is a catch with doing a DFS on the graph, as it is highly dependent on the traversal order of the DFS, which is effectively random.

The Catch

Right

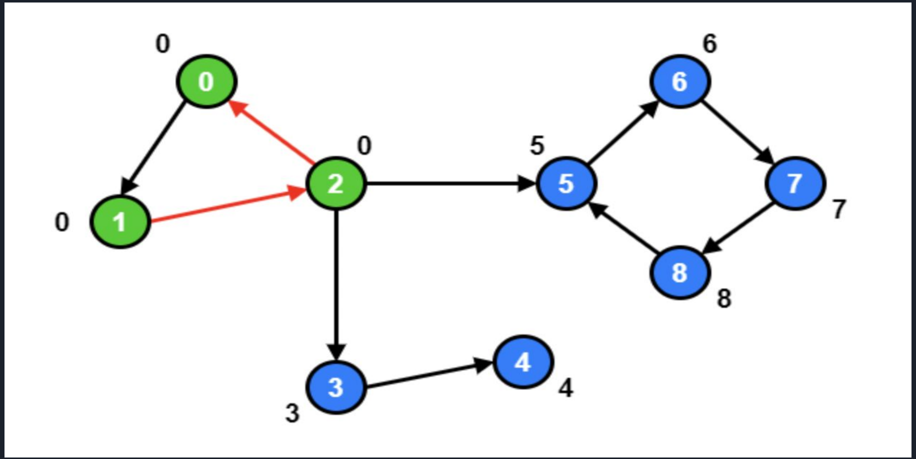
Wrong



Order of assignment differs. Depending on where the DFS starts, and the order in which nodes/edges are visited, the low-link values for identifying SCCs could be wrong.

Tarjan’s Algorithm Notes

- To cope with the random traversal order of the DFS, Tarjan’s algorithm maintains a stack of valid nodes from which to update low-link values. Nodes are added to the stack of valid nodes as they are explored for the first time. Nodes are removed from the stack each time a complete SCC is found.
- Update condition for low-link value: If u and v are nodes in a graph and we were currently exploring u, then our new low-link update condition is, to update node u to node v low-link there has to be a path of edges from u to v and node v must be on the stack.
- O(V+E)

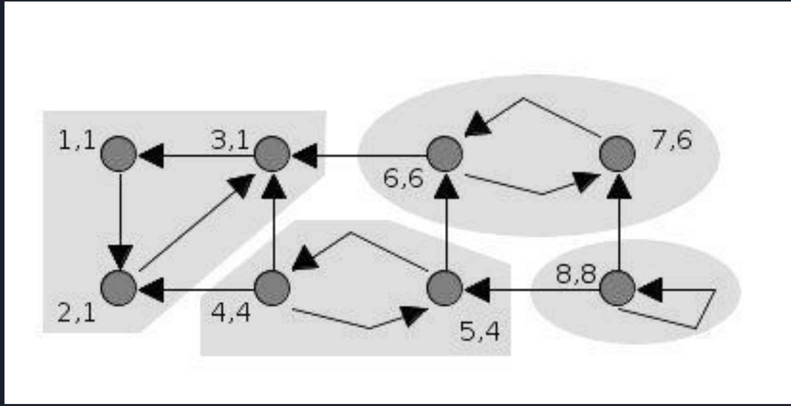


Step by step Tarjan’s

- Mark the id of each node as unvisited.
- Start DFS. Upon visiting a node assign it an id and a low-link value. Also mark the current nodes as visited and add them to a seen stack.
- On DFS callback, if the previous node is on the stack then min the current node’s low-link value with the last node’s low-link value.
- After visiting all neighbours, if the current node started a connected component and it has exhausted all its outgoing edges then pop nodes off stack until current node is reached.

Animation

- First number is the id of the node.
- Second number is the low-link value.
- As seen by node with id 3, node 3's low-link value will be updated to 1 because 1 is on the stack. Then the DFS call back updates the low-link value of node 2 because node 3 is in stack. Node 2 will be updated to node 3's low link value. Note that we don't go to visited nodes.
- The low link value and the id of the final node will be the same.



Pseudocode

```
UNVISITED = -1
n = number of nodes in graph
g = adjacency list with directed edges

id = 0        # Used to give each node an id
sccCount = 0 # Used to count number of SCCs found

# Index i in these arrays represents node i
ids = [0, 0, … 0, 0]                     # Length n
low = [0, 0, … 0, 0]                     # Length n
onStack = [false, false, …, false] # Length n
stack = an empty stack data structure

function findSccs():
  for(i = 0; i < n; i++): ids[i] = UNVISITED
  for(i = 0; i < n; i++):
    if(ids[i] == UNVISITED):
      dfs(i)
  return low
```

Pseudocode

```
function dfs(at):
  stack.push(at)
  onStack[at] = true
  ids[at] = low[at] = id++

  # Visit all neighbours & min low-link on callback
  for(to : g[at]):
    if(ids[to] == UNVISITED): dfs(to)
    if(onStack[to]): low[at] = min(low[at],low[to])

  # After having visited all the neighbours of 'at'
  # if we're at the start of a SCC empty the seen
  # stack until we're back to the start of the SCC.
  if(ids[at] == low[at]):
    for(node = stack.pop();;node = stack.pop()):
      onStack[node] = false
      low[node] = ids[at]
      if(node == at): break
    sccCount++
```

Problems

https://usaco.guide/adv/SCC?lang=cpp

https://www.hackerearth.com/practice/algorithms/graphs/strongly-connected-components/practice-problems/

Questions?