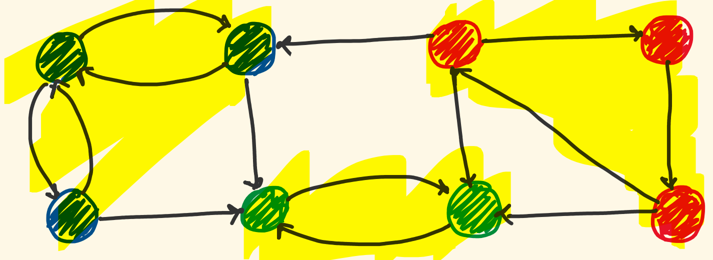
Strongly connected components

Friday, October 27, 2023 5:44 PM

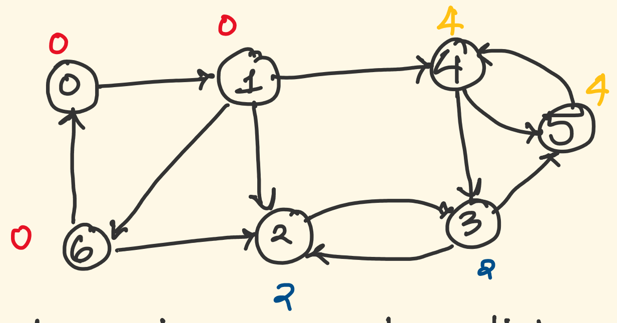
- Self contained cycles within a directed graph.
- Every vertex in the cycle can reach every other vertex in the same cycle.



Once we leave the cycle, there is no coming back

Low link values

- Value of the lowest id reachable from a node.

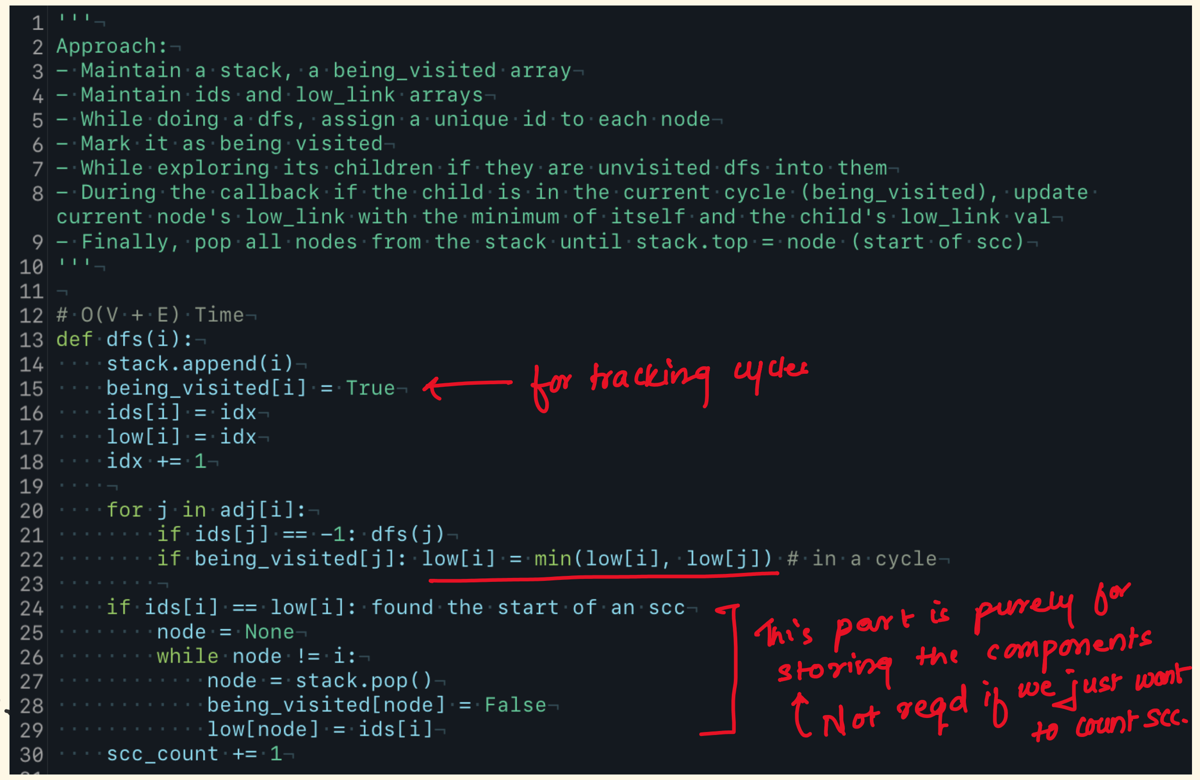


- All nodes with same low link val belong to the same scc.
- DFS cannot be used to determine Low link vals.
  ↳ Due to randomness of traversal.

Tarjan's Strongly connected components.

- To cope with the randomness of DFS, we use a stack to track connected components.
- Start DFS from any unvisited node.
- Give it a uniq ID & push it to stack.
- Also assign itself as the low link.
- Visit all its unvisited neighbors.
- During the callback get the min low link & update.
  low[node] = min (low[node], low[child])
- After exploring all child nodes, pop the component from the stack.

This can be only done if the child is being visited
↔ This denotes that it is in the cycle.



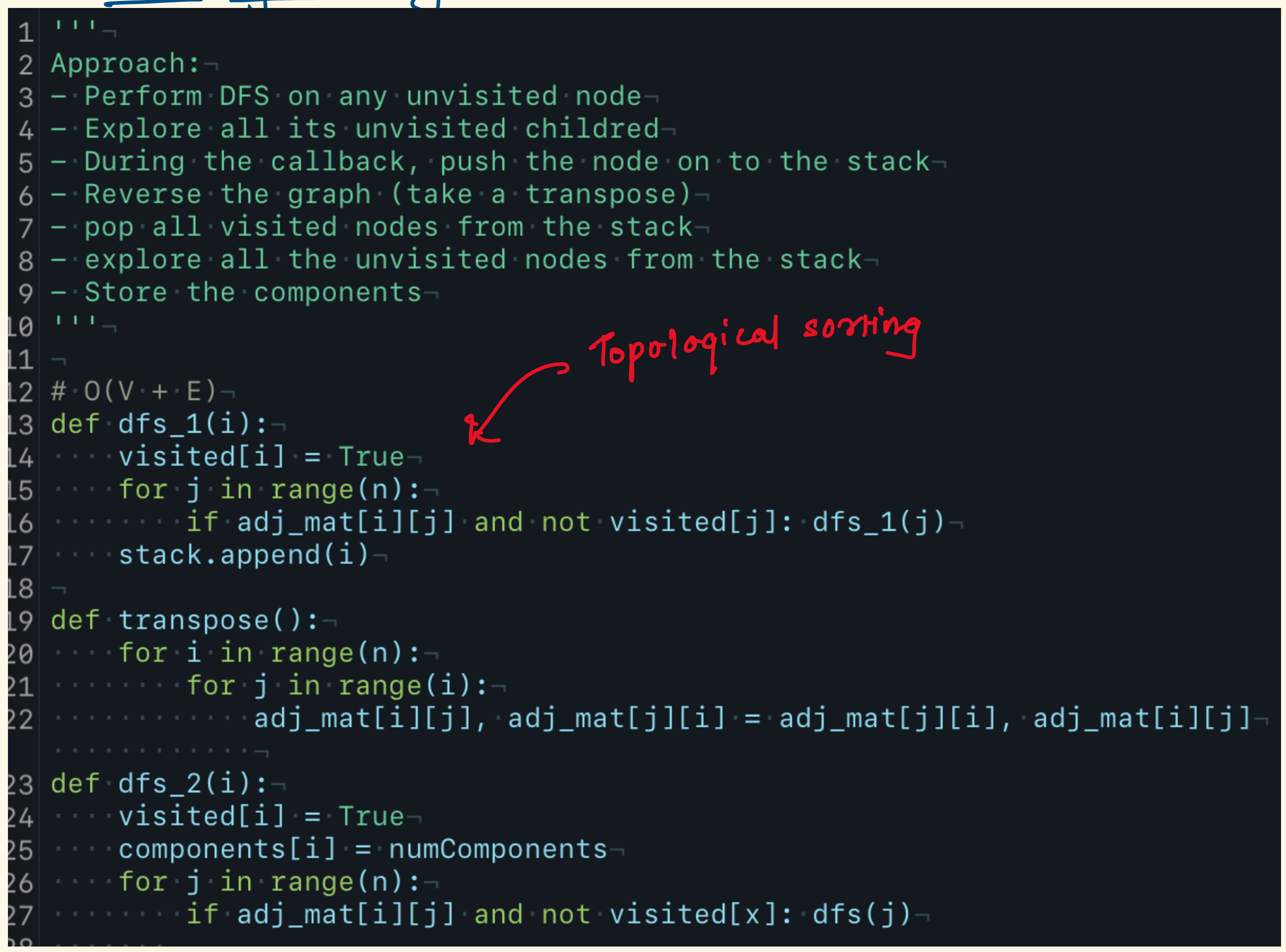
```
'''
Approach:
- Maintain a stack, a being_visited array
- Maintain ids and low_link arrays
- While doing a dfs, assign a unique id to each node
- Mark it as being visited
- While exploring its children if they are unvisited dfs into them
- During the callback if the child is in the current cycle (being_visited), update current node's low_link with the minimum of itself and the child's low_link val
- Finally, pop all nodes from the stack until stack.top = node (start of scc)
'''

# O(V + E) Time
def dfs(i):
    stack.append(i)
    being_visited[i] = True
    ids[i] = idx
    low[i] = idx
    idx += 1
    
    for j in adj[i]:
        if ids[j] == -1: dfs(j)
        if being_visited[j]: low[i] = min(low[i], low[j]) # in a cycle
        
    if ids[i] == low[i]: found the start of an scc
        node = None
        while node != i:
            node = stack.pop()
            being_visited[node] = False
            low[node] = ids[i]
    scc_count += 1
```

for tracking cycles

This part is purely for storing the components
↑ Not reqd if we just want to count scc.

Kosaraju's Algorithm



```
'''
Approach:
- Perform DFS on any unvisited node
- Explore all its unvisited childred
- During the callback, push the node on to the stack
- Reverse the graph (take a transpose)
- pop all visited nodes from the stack
- explore all the unvisited nodes from the stack
- Store the components
'''

# O(V + E)
def dfs_1(i):
    visited[i] = True
    for j in range(n):
        if adj_mat[i][j] and not visited[j]: dfs_1(j)
    stack.append(i)

def transpose():
    for i in range(n):
        for j in range(i):
            adj_mat[i][j], adj_mat[j][i] = adj_mat[j][i], adj_mat[i][j]
            
def dfs_2(i):
    visited[i] = True
    components[i] = numComponents
    for j in range(n):
        if adj_mat[i][j] and not visited[x]: dfs(j)
```

Topological sorting