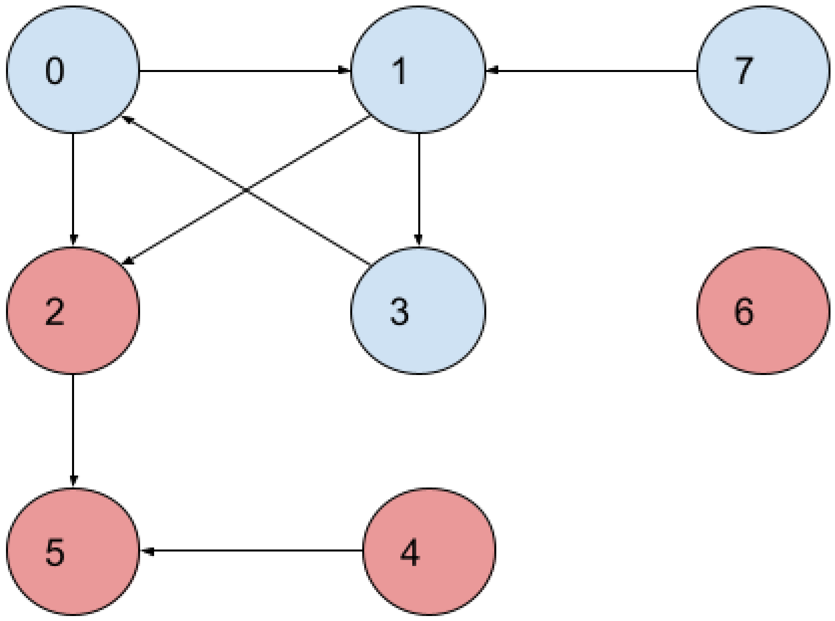
MEDIUM

Eventual Safe States using Topological Sort

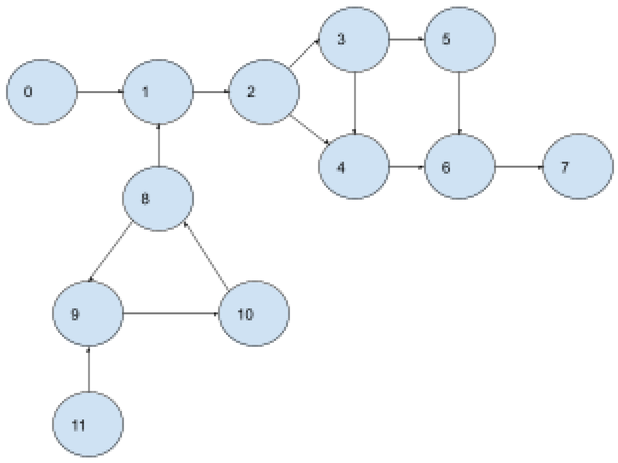
Intuition

Safe Node : Those nodes that lead to the terminal nodes are called as safe nodes
Terminal Nodes : Those nodes whose outdegree is 0

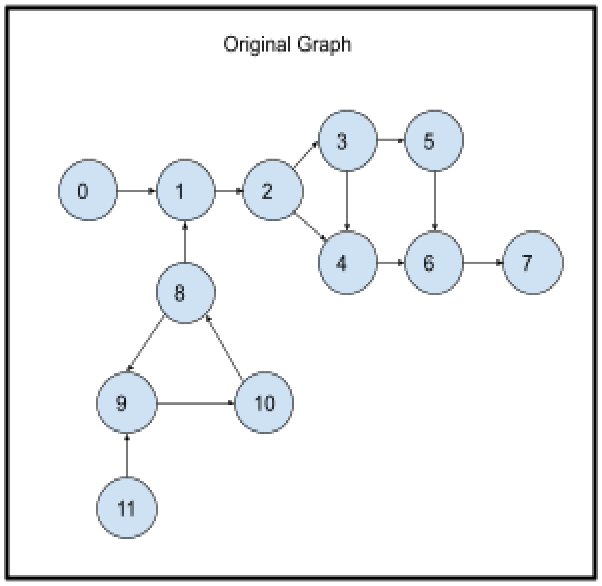


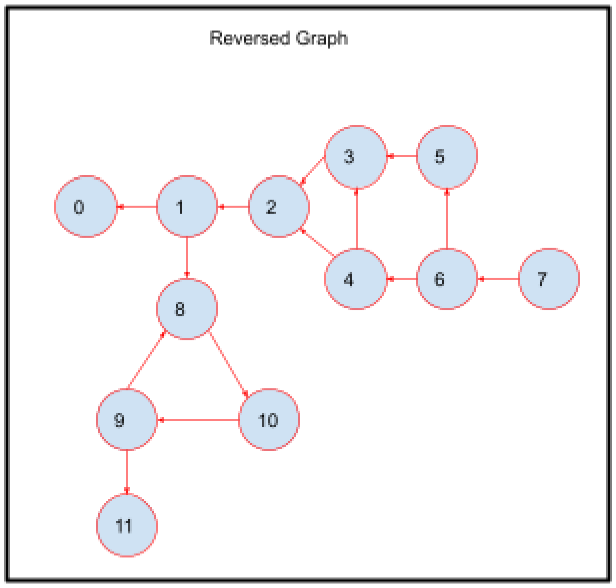
We can say that if a node leads to a cycle than it can not be a terminal node, that is where the topological sort algorithm can help us as it is only applicable to the Directed Acyclic Graphs

Eg.



Reverse the edges





Get all nodes with indegree 0 and then :

Removal of neighbors on adjacent nodes

Eg. 7 - 0, 6 - 1, ….. [actually these are out degrees but now to apply topological sort reversed all edges]

Now terminal nodes are those with indegree 0

Perform topological sort as : indegree 0s to the queue | reduce indegrees |

Eg.
Queue = [7]
result vector = [ ]

Queue = [6]
Result = [7]
……

Queue = [3]
Result = [7, 6, 5, 4, 3 ……]
…….

Queue = [ ]
Result = [7 6 5 4 3 2 1 0]

Approach

- Reverse the graph
- Create an indegree vector that contains the indegree of all elements
- Create an empty queue
- Insert elements with 0 indegree into the queue
- Declare empty result / safe vector
- Traverse until queue becomes empty :
    - Extract front of queue
    - Pop front of queue
    - Add front to safe vector
    - Traverse for adjacent elements of front :
        - Reduce indegree of adjacent nodes
        - If indegree becomes 0 add them to queue
- Sort the safe vector
- Return safe vector

Function Code

```
vector<int> eventualSafeNodes(int V, vector<int> graph[]) {
        // getting dimensions
        int n = V;
        // reversing directions of edges
        vector<vector<int>> adj(n);
        // creating indegree vector and initializing it
        vector<int> indegree(n,0);
        for(int i=0;i<n;i++)
        {
            for(auto it: graph[i])
            {
                adj[it].push_back(i);
                indegree[i]++;
            }
        }

        // declaring queue
        queue<int> q;
        for(int i=0;i<n;i++)
        {
        // pushing elements with 0 indegree into queue
            if(indegree[i]==0)
            {
                q.push(i);
            }
        }
        // declaring vector to store safe nodes
        vector<int> safe;
        // until q becomes empty
        while(!q.empty())
        {
        // extracting and popping first element from queue
        // adding first element to safe nodes
            int node = q.front();
            q.pop();
            safe.push_back(node);
            // traversing through the adjacent nodes
            for(auto it: adj[node])
            {
        // reducing indegree and checking if it becomes 0 then adding
element to queue
                indegree[it]--;
                if(indegree[it]==0)
                {
                    q.push(it);
                }
            }
        }
        // sorting the safe nodes
        sort(safe.begin(),safe.end());
        // returning safe vector
        return safe;
    }
```

Time Complexity

O(V+E)