MEDIUM

Find Eventual safe states

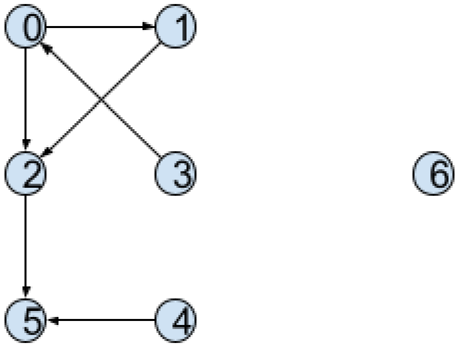
Intuition

Given : A directed graph having V vertices and E edges in form of adjacency list

Definitions :

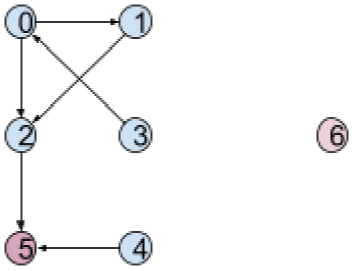
- Terminal Node : A node is called a terminal node if there are no outgoing edges.
- Safe Node : A node is called a safe node if every possible path starting from that node leads to a terminal node

We have to return an array containing the safe nodes of a graph in ascending order



Safe Nodes = []

Ie. Every path ends up at a terminal node
We will be solving this using the cycle detection technique.(If out-degree == 0)
0 - not a terminal node
1 - not a terminal node
2 - not a terminal node
3 - not a terminal node
4 - not a terminal node
5 - terminal node
6 - terminal node



0 -> 1 -> 3 -> 0 hence 0 is not a safe node
1 -> 3 -> 0 -> 1 hence 1 is not a safe node
2 -> 5 ends up at 5 hence 2 is a safe node
3 -> 0 -> 1 -> 3 hence 3 is not a safe node [one path of 3 ends up at a terminal node, but the other path ends up at a non-terminal node]
4 -> 5 ends up at 4 hence 4 is a safe node
5 -> itself a terminal node hence is a safe node
6 -> itself a terminal node hence is a safe node

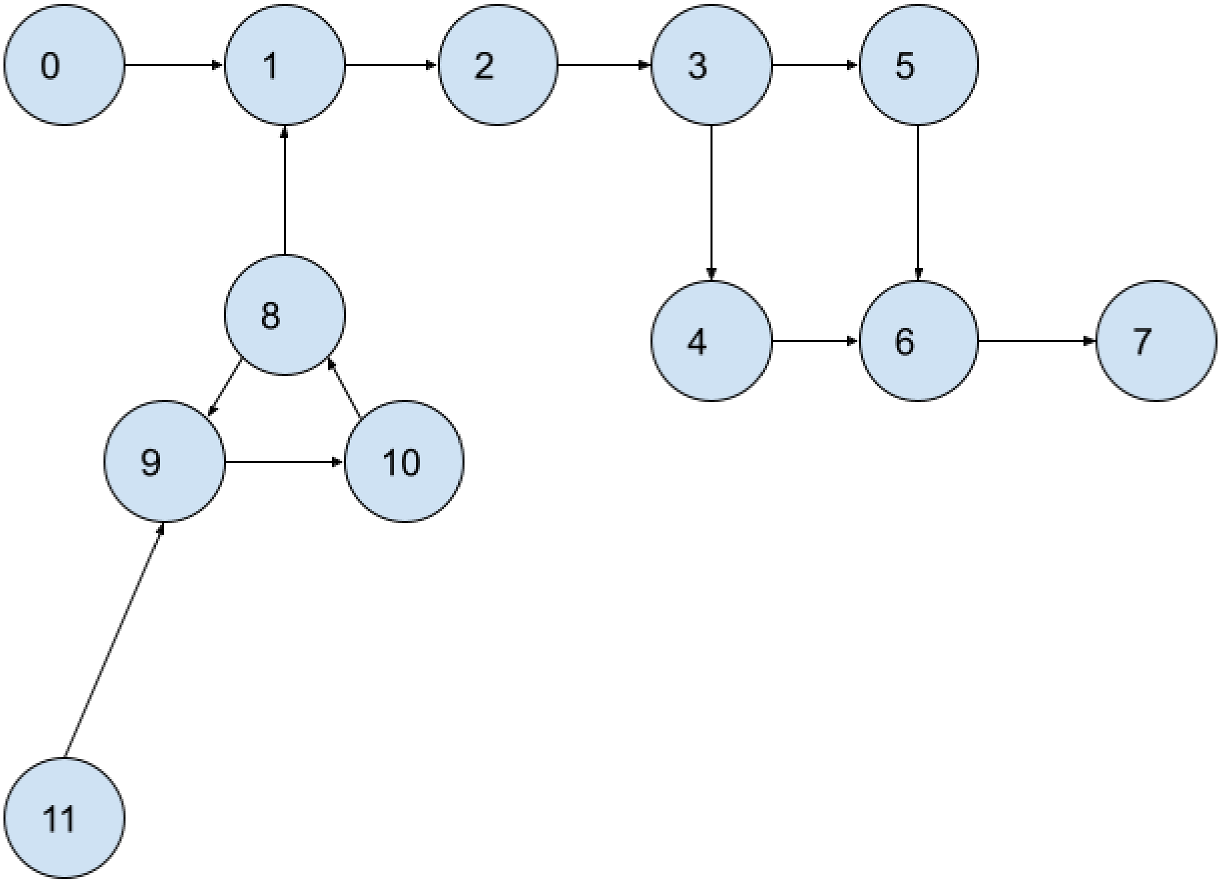
Returns [2, 4, 5, 6]

Inference :

- If a node leads to a path having a cycle or is a part of a cycle then they can't be a safe node
    - PART OF A CYCLE
    - CONNECTED TO A CYCLE

We will be able to solve the problem using DFS technique

Eg.



Adjacency list :

0 - [1]
1 - [2]
2 - [3]
3 - [4,5]
4 - [6]
5 - [6]
6 - [7]
7 - [ ]
8 - [1,9]
9 - [10]
10 - [8]
11 - [9]

Visited

0	1	2	3	4	5	6	7	8	**9**	10	11
1	1	1	1	1	1	1	1	1	**1**	1	

Path visited

0	1	2	3	4	5	6	7	8	**9**	10	11
0	0	0	0	0	0	0	0	1	**1**	1	

Different catch, Do not reset if the path visited if you get a cycle

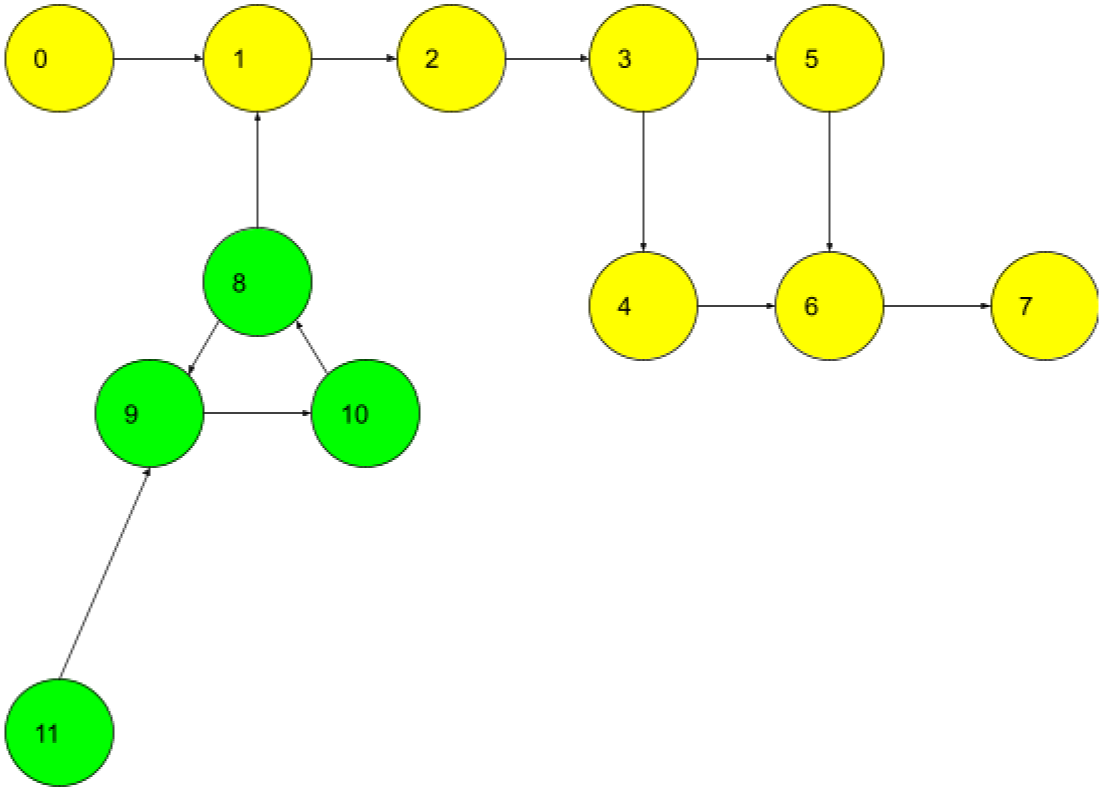
After 7 we do not have any further node hence, it's a safe node

Safe_nodes = [7, 6, 4, 5, 3, 2, 1]
At 10 -> 9 we occurred a cycle

We will return sorted (Safe Node vector)

Ie. [1, 2, 3, 4, 5, 6, 7]

Result :



Approach

eventualSafeNodes()

- Declare :
    - Visited array having size n initially assigned 0s
    - Path visited array having size n initially assigned 0s
    - Check array to mark safe nodes of n size initially assigned 0s
- Traverse through all the component nodes :
    - Make a call for dfs to mark the safe nodes as :
        - dfs(node,adj,visited,pathvisited,check)
- Traverse through all component nodes :
    - Check if the nodes are safe :
        - Append the safe nodes to the result / safe nodes vector
- Return the safe nodes

DFS()

- Mark the node as visited
- Mark the node as pathvisiited
- Mark the node as unsafe
- Traverse through all adjacent components of the node :
    - Check if the adjacent node is unvisited :

        - If unvisited then make a dfs call to check if this runs to a cycle as dfs(node, adj, visited, pathvisited, check) :
            - If runs into cycle mark unsafe
            - Return true
        - Else if its visited and path visited then :
            - Mark as unsafe
            - Return true
- If this all runs then outside the loop mark the node as safe
- Unvisit the visited path
- Return false

Function Code

```
bool dfs(int node,vector<int> adj[],vector<int> &visited,vector<int> &pathvisited,vector<int> &check)
    {
        // marking the node as visited and path visited and initially considering it is not a safe node
        visited[node] = 1;
        pathvisited[node] = 1;
        check[node] = 0;
        // traversing through the adjacent elements of the passed source
        for(int i:adj[node])
        {
            if(!visited[i])
            {
                // check if it forms a cycle
                if(dfs(i,adj,visited,pathvisited,check)==true)
                {
                    // if its a cycle it cannot be a safe node
                    check[node] = 0;
                    return true;
                }
            }
            // checking if the node is visited and path visited already and forms a cycle
            else if(pathvisited[i]==1)
            {
                // if its a cycle it cannot be a safe node
                check[node] = 0;
                return true;
            }
        }
        // if the above fails then its a safe node and mark it as safe and
also unvisit the path
        check[node] = 1;
        pathvisited[node] = 0;
        return false;
    }

    vector<int> eventualSafeNodes(int n, vector<int> adj[]) {
        // declaring
        // visited vector having n elements and all assigned 0 initially
        vector<int> visited(n,0);
        vector<int> pathvisited(n,0);
        // marking the safe nodes
        vector<int> check(n,0);
        // vector containing the safe nodes
        vector<int> safe;
        // traversing through all component nodes
        for(int i=0;i<n;i++)
        {
            // checking if the node is unvisited
            if(!visited[i])
            {
                // call for the dfs traversal as follows
               dfs(i,adj,visited,pathvisited,check);
            }
        }
        // traverse through all nodes
        for(int i=0;i<n;i++)
        {
            // check if its a safe node then append it to the safe node
vector
            if(check[i]==1)safe.push_back(i);
        }

        // return the vector containing the safe nodes
        return safe;
    }
```

Time Complexity

O (V + E)