**Eventual Safe States:**

**Medium** Accuracy: **55.52%** Submissions: **38K+** Points: **4**

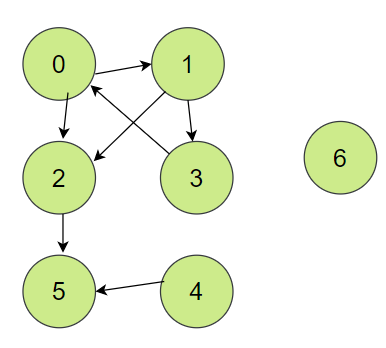
A directed graph of **V** vertices and **E** edges is given in the form of an adjacency list **adj**. Each node of the graph is labelled with a distinct integer in the range **0** to **V - 1**.

A node is a **terminal node** if there are no outgoing edges. A node is a **safe node** if every possible path starting from that node leads to a **terminal node**.

You have to return an array containing all the **safe nodes** of the graph. The answer should be sorted in **ascending** order.

**Example 1:**

**Input:**



**Output:**

2 4 5 6

**Explanation:**

The given graph is shown above.

Nodes 5 and 6 are terminal nodes as there are no

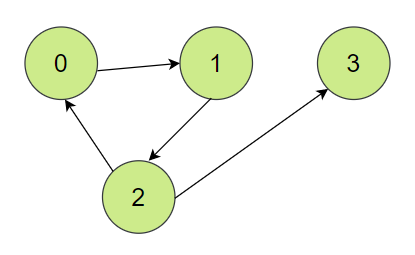
outgoing edges from either of them.

Every path starting at nodes 2, 4, 5, and 6 all

lead to either node 5 or 6.

**Example 2:**

**Input:**

****

**Output:**

3

**Explanation:**

Only node 3 is a terminal node, and every path

starting at node 3 leads to node 3.

**Your Task:**  
You don't need to read or print anything. Your task is to complete the function **eventualSafeNodes()**which takes an integer **V** denoting no. of vertices and **adj** denoting adjacency list of the graph and returns an array of **safe nodes**.

**Expected Time Complexity:** O(V + E)

**Expected Space Complexity:** O(V)

**Constraints:**

* 1 <= V <= 104
* 0 <= E <= 104
* The graph won't contain self loops.
* Each node in the graph has a distinct value in the range 0 to V - 1.

**Code: -**

//{ Driver Code Starts

// Initial Template for C++

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function Template for C++

class Solution {

public:

bool helper(int cur, vector<int> adj[], vector<bool> &vis, vector<int> &dp){

// base case

if(adj[cur].empty())

return dp[cur] = true;

if(dp[cur] != -1)

return dp[cur];

// recursive case

bool possible = true;

// checking every outgoing paths for current vertex

for(auto &vertex : adj[cur]){

// already repeating

if(vis[vertex] == true){

if(dp[vertex] == -1)

possible = possible & false;

else

possible = possible & dp[vertex];

}

else{

vis[vertex] = true;

possible = possible & helper(vertex, adj, vis, dp);

}

}

return dp[cur] = possible;

}

vector<int> eventualSafeNodes(int V, vector<int> adj[]) {

// code here

vector<bool> vis(V, false);

vector<int> dp(V, -1), ans;

for(int i = 0; i < V; ++i){

if(!vis[i]){

bool possible = helper(i, adj, vis, dp);

if(possible) ans.push\_back(i);

}

else{

if(dp[i]) ans.push\_back(i);

}

}

return ans;

}

};

//{ Driver Code Starts.

int main() {

int t;

cin >> t;

while (t--) {

int V, E;

cin >> V >> E;

vector<int> adj[V];

for (int i = 0; i < E; i++) {

int u, v;

cin >> u >> v;

adj[u].push\_back(v);

}

Solution obj;

vector<int> safeNodes = obj.eventualSafeNodes(V, adj);

for (auto i : safeNodes) {

cout << i << " ";

}

cout << endl;

}

}

// } Driver Code Ends

**T.C: - O(V + E)**

**S.C: - O(V)**