

Find eventual safe state in C++

```
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    bool dfsCheck(int node, vector<int> adj[], int vis[],
int pathVis[],
        int check[]) {
        vis[node] = 1;
        pathVis[node] = 1;
        check[node] = 0;
        // traverse for adjacent nodes
        for (auto it : adj[node]) {
            // when the node is not visited
            if (!vis[it]) {
                if (dfsCheck(it, adj, vis, pathVis, check) == true) {
                    check[node] = 0;
                    return true;
                }
            }
            // if the node has been previously visited
            // but it has to be visited on the same path
            else if (pathVis[it]) {
                check[node] = 0;
                return true;
            }
        }
        check[node] = 1;
        pathVis[node] = 0;
        return false;
    }
public:
    vector<int> eventualSafeNodes(int V, vector<int>
adj[]) {
        int vis[V] = {0};
        int pathVis[V] = {0};
        int check[V] = {0};
        vector<int> safeNodes;
        for (int i = 0; i < V; i++) {
            if (!vis[i]) {
                dfsCheck(i, adj, vis, pathVis, check);
            }
        }
        for (int i = 0; i < V; i++) {
            if (check[i] == 1) safeNodes.push_back(i);
        }
        return safeNodes;
    }
};

int main() {
    //V = 12;
    vector<int> adj[12] = {{1}, {2}, {3}, {4, 5}, {6}, {6}, {7}, {},
{1, 9}, {10},
    {8}, {9}};
    int V = 12;
    Solution obj;
    vector<int> safeNodes = obj.eventualSafeNodes(V,
adj);
    for (auto node : safeNodes) {
```

Goal

We want to find all the **eventual safe nodes** in a **directed graph**, i.e., nodes from which **every path eventually ends in a terminal node** (a node with no outgoing edges). This is solved using **DFS cycle detection**.

Q Key Concepts

- vis[] → marks if a node has been visited.
- pathVis[] → tracks the current recursion path.
- check[] → 1 if node is *safe*, 0 if not.

A node is **not safe** if:

- A cycle is detected starting from it (or reachable from it).

Input Graph (Adjacency List)

```
0 → 1
1 → 2
2 → 3
3 → 4,5
4 → 6
5 → 6
6 → 7
7 → {} ← terminal node
8 → 1,9
9 → 10
10 → 8
11 → 9
```

DFS Cycle Detection

Let's go through the DFS starting from each unvisited node:

Node	Path	Cycle Detected	Safe?
0	0→1→2→3→4→6→7	No	✓ Yes
1	Already visited from 0	-	✓ Yes
2	Already visited from 0	-	✓ Yes
3	Already visited from 0	-	✓ Yes
4	Already visited from 0	-	✓ Yes
5	5→6→7	No	✓ Yes
6	Already visited	-	✓ Yes
7	Terminal	No	✓ Yes
8	8→1→... (already	✓ Yes	✗ No

<pre> cout << node << " "; } cout << endl return 0; }</pre>		visited) AND 8→9→10→8 (cycle)		
	9	9→10→8→9	✔ Yes	✘ No
	10	10→8→9→10	✔ Yes	✘ No
	11	11→9→cycle	✔ Yes	✘ No
<p>✔ Safe Nodes</p> <p>From the table above, the safe nodes are:</p> <p>0 1 2 3 4 5 6 7</p>				
<p>Output:- 0 1 2 3 4 5 6 7</p>				