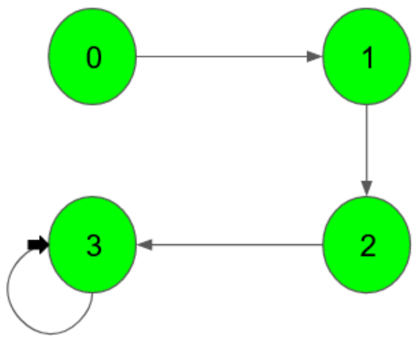
# Directed Graph Cycle

Given a Directed Graph with **V** vertices (Numbered from **0** to **V-1**) and **E** edges, check whether it contains any cycle or not.

**Example 1:**

**Input:**

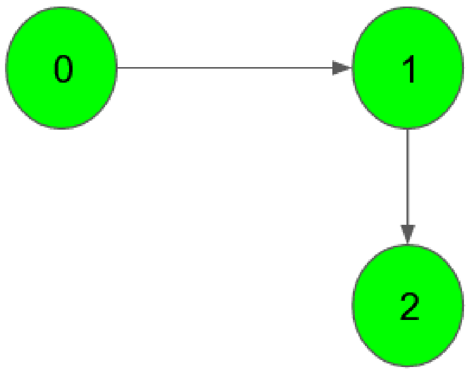


**Output:** 1

**Explanation**: 3 -> 3 is a cycle

**Example 2:**

**Input:**



**Output:** 0

**Explanation**: no cycle in the graph

**Your task:**
You dont need to read input or print anything. Your task is to complete the function **isCyclic()** which takes the integer V denoting the number of vertices and adjacency list **adj** as input parameters and returns a boolean value denoting if the given directed graph contains a cycle or not.
In the adjacency list **adj,** element **adj[i][j]** represents an edge from **i** to **j.**

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

**Constraints:**
$1 \leq V, E \leq 10^5$

```
class Solution {
  public:
    bool dfs(int node, vector<int> adj[], vector<int>& state)
{
        // Mark the node as Visiting (1)
        state[node] = 1;

        // Traverse through all adjacent vertices
        for (int neighbor : adj[node]) {
            // If neighbor is not visited, recursively visit
it
            if (state[neighbor] == 0) {
                if (dfs(neighbor, adj, state)) {
                    return true; // Cycle found
                }
            }
            // If neighbor is in Visiting state, a cycle
exists
            else if (state[neighbor] == 1) {
                return true; // Cycle found
            }
        }

        // Mark the node as Visited (2) after all its
neighbors are explored
```

```
        state[node] = 2;
        return false;
    }

    // Function to detect cycle in a directed graph.
    bool isCyclic(int V, vector<int> adj[]) {
        // Create a state array to track the visitation state
of each node
        vector<int> state(V, 0); // 0: Not visited, 1:
Visiting, 2: Visited

        // Check for cycles in each component of the graph
        for (int i = 0; i < V; i++) {
            if (state[i] == 0) { // Start DFS only for
unvisited nodes
                if (dfs(i, adj, state)) {
                    return true; // Cycle found
                }
            }
        }

        return false; // No cycle found
    }
};
```