Undirected Graph Cycle

Given an undirected graph with V vertices labelled from 0 to V-1 and E edges, check whether it contains any cycle or not. Graph is in the form of adjacency list where adj[i] contains all the nodes ith node is having edge with.

NOTE: The adjacency list denotes the edges of the graph where edges[i] stores all other vertices to which ith vertex is connected.

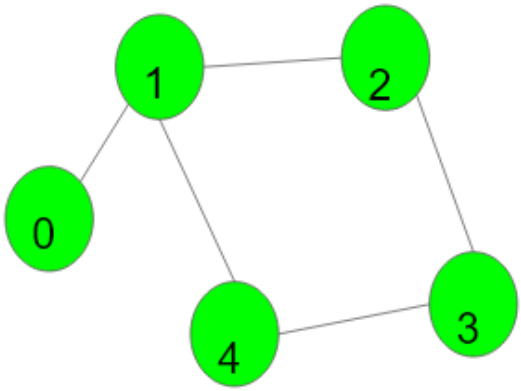
Examples:

Input: V = 5, E = 5

adj = [[1], [0, 2, 4], [1, 3], [2, 4], [1, 3]]

Output: 1

Explanation:



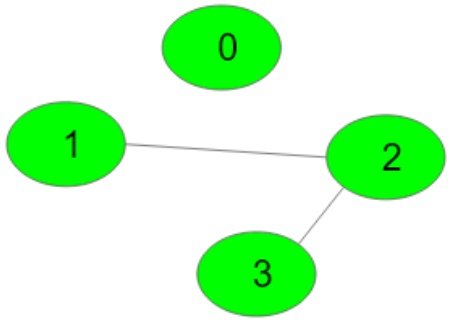
1->2->3->4->1 is a cycle.

Input: V = 4, E = 2

adj = [[], [2], [1, 3], [2]]

Output: 0

Explanation:



No cycle in the graph.

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

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

```
class Solution {
  public:
    bool dfs(int node, int parent, vector<int> adj[],
vector<bool>& visited) {
        // Mark the current node as visited
        visited[node] = true;

        // Traverse through all the adjacent nodes
        for (int neighbor : adj[node]) {
            // If neighbor is not visited, recursively visit
it
            if (!visited[neighbor]) {
                if (dfs(neighbor, node, adj, visited)) {
                    return true;
                }
            }
```

```
            // If neighbor is visited and it's not the parent
of the current node, there's a cycle
            else if (neighbor != parent) {
                return true;
            }
        }

        return false;
    }


// Function to detect cycle in an undirected graph.
    bool isCycle(int V, vector<int> adj[]) {
        vector<bool> visited(V, false);

        // Traverse all vertices
        for (int i = 0; i < V; i++) {
            // If a vertex is not visited, apply DFS from that
vertex
            if (!visited[i]) {
                if (dfs(i, -1, adj, visited)) {
                    return true;  // Cycle found
                }
            }
        }

        return false;
    }
};
```