|  |  |
| --- | --- |
| **Cycle detection in undirected graph using Depth First Search in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {    private:      bool dfs(int node, int parent, int vis[], vector<int> adj[]) {          vis[node] = 1;          // visit adjacent nodes          for(auto adjacentNode: adj[node]) {              // unvisited adjacent node              if(!vis[adjacentNode]) {                  if(dfs(adjacentNode, node, vis, adj) == true)                      return true;              }              // visited node but not a parent node              else if(adjacentNode != parent) return true;          }          return false;      }    public:      // Function to detect cycle in an undirected graph.      bool isCycle(int V, vector<int> adj[]) {         int vis[V] = {0};         // for graph with connected components         for(int i = 0;i<V;i++) {             if(!vis[i]) {                 if(dfs(i, -1, vis, adj) == true) return true;             }         }         return false;      }  };  int main() {        // V = 4, E = 2      vector<int> adj[4] = {{}, {2}, {1, 3}, {2}};      Solution obj;      bool ans = obj.isCycle(4, adj);      if (ans)          cout << "1\n";      else          cout << "0\n";      return 0;  } | Graph looks like: -  1 -- 2 – 3  Adjacency list looks like:- adj[0] = {}  adj[1] = {2}  adj[2] = {1, 3}  adj[3] = {2}  Step-by-Step Execution:   1. **Initialization**:    * vis = {0, 0, 0, 0} (all nodes unvisited). 2. **Node 0**:    * vis[0] = 0 (no edges from node 0, skip). 3. **Node 1**:    * vis[1] = 0, start DFS from node 1. 4. **DFS from Node 1**:    * node = 1, parent = -1.    * Mark 1 as visited: vis = {0, 1, 0, 0}.    * Visit adjacent node 2 (unvisited):      + Call dfs(2, 1). 5. **DFS from Node 2**:    * node = 2, parent = 1.    * Mark 2 as visited: vis = {0, 1, 1, 0}.    * Visit adjacent nodes:      + Node 1: Already visited, but it's the parent node (skip).      + Node 3: Unvisited:        - Call dfs(3, 2). 6. **DFS from Node 3**:    * node = 3, parent = 2.    * Mark 3 as visited: vis = {0, 1, 1, 1}.    * Visit adjacent nodes:      + Node 2: Already visited, but it's the parent node (skip). 7. **DFS Ends**:    * Backtrack to node 2, then to node 1. 8. **Node 1 Ends**:    * Continue checking other nodes in isCycle().    * Node 0, 2, and 3 are already visited. 9. **Cycle Check**:    * No cycles found during traversal. |
| **Output:-**  **0**  **No cycle** | |