|  |  |
| --- | --- |
| **Cycle detection in undirected graph using Breadth First Search in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {    private:    bool detect(int src, vector<int> adj[], int vis[]) {        vis[src] = 1;        // store <source node, parent node>        queue<pair<int,int>> q;        q.push({src, -1});        // traverse until queue is not empty        while(!q.empty()) {            int node = q.front().first;            int parent = q.front().second;            q.pop();              // go to all adjacent nodes            for(auto adjacentNode: adj[node]) {                // if adjacent node is unvisited                if(!vis[adjacentNode]) {                    vis[adjacentNode] = 1;                    q.push({adjacentNode, node});                }                // if adjacent node is visited and is not it's own parent node                else if(parent != adjacentNode) {                    // yes it is a cycle                    return true;                }            }        }        // there's no cycle        return false;    }    public:      // Function to detect cycle in an undirected graph.      bool isCycle(int V, vector<int> adj[]) {          // initialise them as unvisited          int vis[V] = {0};          for(int i = 0;i<V;i++) {              if(!vis[i]) {                  if(detect(i, adj, vis)) 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 Definition (Adjacency List)**  vector<int> adj[4] = {  {}, // 0 → No neighbors  {2}, // 1 → 2  {1, 3}, // 2 → 1, 3  {2} // 3 → 2  };  Visual graph:  1 -- 2 -- 3   * It's a **linear graph**, no cycle expected.   **🧠 Variables**   * vis[4] = {0, 0, 0, 0} (all unvisited initially) * Queue for BFS: stores pairs {node, parent}   **🔁 Step-by-Step Traversal Table**   | **Iter** | **Queue** | **node** | **parent** | **Neighbours** | **Action** | | --- | --- | --- | --- | --- | --- | | 1 | {1, -1} | 1 | -1 | [2] | 2 is unvisited → mark visited, enqueue {2, 1} | | 2 | {2, 1} | 2 | 1 | [1, 3] | 1 is parent → skip; 3 is unvisited → mark visited, enqueue {3, 2} | | 3 | {3, 2} | 3 | 2 | [2] | 2 is parent → skip | | 4 | empty | — | — | — | Loop ends |   **Visited array after traversal**: [0, 1, 1, 1]  No condition parent != adjacentNode && vis[adjacentNode] == 1 was met.  **✅ Final Output**  0 // No cycle found  **📋 Summary Table**   | **Node** | **Parent** | **Visited** | **Notes** | | --- | --- | --- | --- | | 1 | -1 | ✅ | Starting node | | 2 | 1 | ✅ | Connected from node 1 | | 3 | 2 | ✅ | Connected from node 2 | | 0 | - | ❌ | Isolated node (not connected) |   **🧠 Conclusion**   * **No cycle** detected — the output is 0. |
| **Output:-**  **0** No cycle was found in any component of the graph | |