|  |  |
| --- | --- |
| **Cycle detection in undirected graph using Breadth First Search in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {  public:      // Function to detect cycle in a directed graph.      bool isCyclic(int V, vector<int> adj[]) {          int indegree[V] = {0};          for (int i = 0; i < V; i++) {              for (auto it : adj[i]) {                  indegree[it]++;              }          }          queue<int> q;          for (int i = 0; i < V; i++) {              if (indegree[i] == 0) {                  q.push(i);              }          }          int cnt = 0;          // o(v + e)          while (!q.empty()) {              int node = q.front();              q.pop();              cnt++;              // node is in your topo sort              // so please remove it from the indegree              for (auto it : adj[node]) {                  indegree[it]--;                  if (indegree[it] == 0) q.push(it);              }          }          if (cnt == V) return false;          return true;      }  };  int main() {      //V = 6;      vector<int> adj[6] = {{}, {2}, {3}, {4, 5}, {2}, {}};      int V = 6;      Solution obj;      bool ans = obj.isCyclic(V, adj);      if (ans) cout << "True";      else cout << "Flase";      cout << endl;      return 0;  } | Graph looks like:-  1 → 2 → 3 → 4  ↑ ↓  └→ 5  Adjacency list looks like:- adj[0] = {}  adj[1] = {2}  adj[2] = {3}  adj[3] = {4, 5}  adj[4] = {2}  adj[5] = {}  **Step 1: Calculate Indegree**   * Initialize indegree[] = {0, 0, 0, 0, 0, 0}. * Traverse adjacency list to calculate indegree:   + 1 → 2: indegree[2]++ → indegree[] = {0, 0, 1, 0, 0, 0}   + 2 → 3: indegree[3]++ → indegree[] = {0, 0, 1, 1, 0, 0}   + 3 → 4: indegree[4]++ → indegree[] = {0, 0, 1, 1, 1, 0}   + 3 → 5: indegree[5]++ → indegree[] = {0, 0, 1, 1, 1, 1}   + 4 → 2: indegree[2]++ → indegree[] = {0, 0, 2, 1, 1, 1} * Final indegree[]: {0, 0, 2, 1, 1, 1}.   **Step 2: Add Nodes with indegree == 0 to Queue**   * Nodes with indegree == 0: 0, 1. * Initialize queue = {0, 1}.   **Step 3: Process Queue (Topological Sort)**   1. **Process Node 0**:    * Dequeue 0, cnt++ → cnt = 1.    * Node 0 has no outgoing edges; no changes to indegree[].    * queue = {1}. 2. **Process Node 1**:    * Dequeue 1, cnt++ → cnt = 2.    * Node 1 → Node 2: Decrease indegree[2]-- → indegree[] = {0, 0, 1, 1, 1, 1}.    * Node 2 has indegree != 0, so it is not added to the queue.    * queue = {}.   **Step 4: Check for Remaining Nodes**   * **Cycle Exists**:   + Processed nodes (cnt = 2) < Total nodes (V = 6).   + A cycle exists, as some nodes (like 2, 3, 4, 5) were never processed. |
| **Output:-**  **True** The graph contains a cycle | |