|  |  |
| --- | --- |
| **Check graph is bipartite using Breadth First Search in C++** | |
| #include<bits/stdc++.h>  using namespace std;  class Solution {      // colors a component      private:      bool check(int start, int V, vector<int>adj[], int color[]) {          queue<int> q;          q.push(start);          color[start] = 0;          while(!q.empty()) {              int node = q.front();              q.pop();                for(auto it : adj[node]) {                  // if the adjacent node is yet not colored                  // you will give the opposite color of the node                  if(color[it] == -1) {                        color[it] = !color[node];                      q.push(it);                  }                  // is the adjacent guy having the same color                  // someone did color it on some other path                  else if(color[it] == color[node]) {                      return false;                  }              }          }          return true;      }  public:      bool isBipartite(int V, vector<int>adj[]){          int color[V];          for(int i = 0;i<V;i++) color[i] = -1;            for(int i = 0;i<V;i++) {              // if not coloured              if(color[i] == -1) {                  if(check(i, V, adj, color) == false) {                      return false;                  }              }          }          return true;      }  };  void addEdge(vector <int> adj[], int u, int v) {      adj[u].push\_back(v);      adj[v].push\_back(u);  }  int main(){        // V = 4, E = 4      vector<int>adj[4];        addEdge(adj, 0, 2);      addEdge(adj, 0, 3);          addEdge(adj, 2, 3);          addEdge(adj, 3, 1);      Solution obj;      bool ans = obj.isBipartite(4, adj);      if(ans)cout << "1\n";      else cout << "0\n";        return 0;  } | **Dry Run:**  **Input Graph:**  Edges:   * (0, 2) * (0, 3) * (2, 3) * (3, 1)   **Adjacency List:**  adj[0]: 2, 3  adj[1]: 3  adj[2]: 0, 3  adj[3]: 0, 2, 1  **Execution:**   1. Initialize color array: [-1, -1, -1, -1] 2. Start BFS from node 0:    * Assign color 0 to node 0: color = [0, -1, -1, -1].    * Visit node 2, assign color 1: color = [0, -1, 1, -1].    * Visit node 3, assign color 1: color = [0, -1, 1, 1].    * At this point, node 3 and node 2 (adjacent nodes) have the same color (1). Therefore, the graph is **not bipartite**.   **Output:**   * Since the graph contains an odd-length cycle (e.g., 0 → 2 → 3 → 0), it is not bipartite. * The function isBipartite() returns false, and the output is:   0 |
| **Output:-**  **0** | |