|  |  |
| --- | --- |
| **Bipartite in Depth First Search in C++** | |
| #include<bits/stdc++.h>  using namespace std;  class Solution {  private:      bool dfs(int node, int col, int color[], vector<int> adj[]) {          color[node] = col;            // traverse adjacent nodes          for(auto it : adj[node]) {              // if uncoloured              if(color[it] == -1) {                  if(dfs(it, !col, color, adj) == false) return false;              }              // if previously coloured and have the same colour              else if(color[it] == col) {                  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 connected components          for(int i = 0;i<V;i++) {              if(color[i] == -1) {                  if(dfs(i, 0, color, adj) == 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;  } | Graph:  0  / \  2 3  \  1  Adj list:  adj[0] = {2, 3}  adj[1] = {3}  adj[2] = {0, 3} adj[3] = {0, 2, 1}   Step-by-Step DFS Traversal:  1. **Node 0:** Start DFS at node 0 and color it 0:   color = [0, -1, -1, -1]  Adjacent nodes: {2, 3}.   1. **Node 2:** Visit node 2 from node 0, and color it 1 (opposite of 0):   color = [0, -1, 1, -1]  Adjacent nodes: {0, 3}.   * + **Node 0** is already colored 0, which does not conflict.   + Move to node 3.  1. **Node 3:** Visit node 3 from node 2, and color it 0 (opposite of 1):   color = [0, -1, 1, 0]  Adjacent nodes: {0, 2, 1}.   * + **Node 0** is already colored 0, which does not conflict.   + **Node 2** is already colored 1, which does not conflict.   + Move to node 1.  1. **Node 1:** Visit node 1 from node 3, and color it 1 (opposite of 0):   color = [0, 1, 1, 0]  Adjacent nodes: {3}.   * + **Node 3** is already colored 0, which does not conflict.   **Conflict in the Graph:**  Now, backtrack to **Node 3**:   * Adjacent nodes: {0, 2, 1}. * Both **Node 2** and **Node 1** are adjacent to **Node 3**, but they are colored the **same** (1).   This is a violation of the bipartite condition because two nodes (1 and 2) that are both connected to 3 have the same color.  **Conclusion:**  The graph is **not bipartite**, and the output is correctly:  0 |
| **Output:-**  0 | |