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];
```

Graph Structure

Vertices: V = 4
Edges:

- 0 ↔ 2
- 0 ↔ 3
- 2 ↔ 3
- 3 ↔ 1

Adjacency List:

0: [2, 3]
1: [3]
2: [0, 3]
3: [0, 2, 1]

🧪 Dry Run of check() Function (BFS for Coloring)

We want to color the graph with **2 colors (0 and 1)** such that no two adjacent nodes have the same color.

Step	Node	Queue	Color Status	Action
1	0	[0]	[-1, -1, -1, -1]	Start BFS with node 0 → color[0] = 0
2	0	[2, 3]	[0, -1, 1, 1]	2 & 3 uncolored → assign opposite color
3	2	[3]	[0, -1, 1, 1]	0 already colored & valid → continue
4	2	[3]	[0, -1, 1, 1]	3 already colored **with same color** → 🚨
				Conflict found → graph is **not bipartite**

❌ Output:

0

```
    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;
}
```

Output:-
0