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";
```

Graph Construction (4 vertices, 4 edges):

```
addEdge(adj, 0, 2);  // 0 - 2
addEdge(adj, 0, 3);  // 0 - 3
addEdge(adj, 2, 3);  // 2 - 3
addEdge(adj, 3, 1);  // 3 - 1
```

🔁 Adjacency List:

Vertex	Neighbors
0	2, 3
1	3
2	0, 3
3	0, 2, 1

🎯 DFS Coloring Attempt:

- Initialize all colors as -1.
- Try to color graph with **two colors**: 0 and 1.

🧠 Dry Run Table

Node Visited	Action	Color Assigned	Stack/Call Stack	Conflict?
0	Start DFS	0	dfs(0, 0)	No
2	Visit from 0	1	dfs(2, 1)	No
3	Visit from 2	0	dfs(3, 0)	No
0	Already colored	0	Check if conflict with 0	✅ Match
1	Visit from 3	1	dfs(1, 1)	No
3	Already colored	0	Check if conflict with 1	✅ Match
2	Already colored	1	Check if conflict with 3 (expect 1, found 0)	❌ **Conflict!**

At this point, DFS at node 3 sees that its neighbor 2 is also colored 1, and this **violates the bipartite condition**, because both are expected to have **opposite** colors.

❌ Final Result:

0

return 0; }	
Output:- 0	