**Value of friendships**

**JAVA Solution: (**<https://leetcode.com/problems/possible-bipartition/>)

class Solution {

List<Integer>[] graph;

Map<Integer, Integer> color;

private boolean dfs(int node, int nodeColor) {

if(color.containsKey(node)) return color.get(node) == nodeColor;

color.put(node, nodeColor);

for(int newNode : graph[node])

if(!dfs(newNode, nodeColor ^ 1)) return false;

return true;

}

public boolean possibleBipartition(int N, int[][] dislikes) {

graph = new List[N + 1];

for(int i = 1; i <= N; i++) graph[i] = new LinkedList<>();

for(int[] edge : dislikes) {

graph[edge[0]].add(edge[1]);

graph[edge[1]].add(edge[0]);

}

color = new HashMap<>();

for(int node = 1; node <= N; ++node)

if(!color.containsKey(node) && !dfs(node, 0)) return false;

return true;

}

}

**Python Code:**

#!/bin/python3

class Graph:

    def \_\_init\_\_(self, n):  
        self.total\_nodes = n  
        self.color = [-1 for \_ in range(n + 1)]  
        self.nbr = {}  
        for i in range(1, n + 1):  
            self.nbr[i] = []

def is\_group\_possible(n, friends):  
    g = Graph(n)  
    for frnd in friends:  
        p1, p2 = frnd  
        g.nbr[p1].append(p2)  
        g.nbr[p2].append(p1)

    # Do BFS from every node  
    for i in range(1, n + 1):  
        queue = [i]  
        while queue:  
            curr = queue.pop()  
            if g.color[curr] == -1:  
                g.color[curr] = 1

            # check for its neighbour  
            for nb in g.nbr[curr]:  
                if g.color[nb] == -1:  
                    g.color[nb] = 1 - g.color[curr]  
                    queue.append(nb)

                elif g.color[nb] == g.color[curr]:  
                    return False

    return True

**CPP Code:**

class Solution {

public:

bool possibleBipartition(int N, vector<vector<int>>& dislikes) {

// Check for bipartite graph can be done using 2-color graph coloring.

// Idea is to use two colors and assign colors to each node such that

// each adjacent node gets a different color, if we this assignment is not

// possible, then it is not bipartite

// create the graph

vector<vector<int>> g(N);

for(auto &edge: dislikes) {

g[edge[0] - 1].emplace\_back(edge[1] - 1);

g[edge[1] - 1].emplace\_back(edge[0] - 1);

}

// for keeping track of color of each node

// -1: unvisited

// 0: color 1

// 1: color 2

vector<int> color(N, -1);

// in case of disconnected graph

for(int i = 0; i < N; i++) {

// take the current as root node

if(color[i] == -1) {

// color it

color[i] = 1;

queue<int> q;

q.emplace(i);

while(!q.empty()) {

int curr = q.front();

q.pop();

// color the adjacent nodes

for(int adj = 0; adj < g[curr].size(); adj++) {

int adj\_idx = g[curr][adj];

if(color[adj\_idx] == -1) {

// color it to mark visited

color[adj\_idx] = 1 - color[curr];

q.emplace(adj\_idx);

}

else if(color[adj\_idx] == color[curr])

return false;

}

}

}

}

return true;

}

};