<https://leetcode.com/problems/possible-bipartition/>

We want to split a group of n people (labeled from 1 to n) into two groups of **any size**. Each person may dislike some other people, and they should not go into the same group.

Given the integer n and the array dislikes where dislikes[i] = [ai, bi] indicates that the person labeled ai does not like the person labeled bi, return true *if it is possible to split everyone into two groups in this way*.

**Example 1:**

Input: n = 4, dislikes = [[1,2],[1,3],[2,4]]

Output: true

Explanation: The first group has [1,4], and the second group has [2,3].

**Example 2:**

Input: n = 3, dislikes = [[1,2],[1,3],[2,3]]

Output: false

Explanation: We need at least 3 groups to divide them. We cannot put them in two groups.

**Constraints:**

* 1 <= n <= 2000
* 0 <= dislikes.length <= 104
* dislikes[i].length == 2
* 1 <= ai < bi <= n
* All the pairs of dislikes are **unique**.

**Attempt 1: 2023-01-03**

**Solution 1:  Recursive traversal (30 min)**

class Solution {

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

// Build graph

Map<Integer, Set<Integer>> adj = new HashMap<Integer, Set<Integer>>();

for(int i = 1; i <= n; i++) {

adj.put(i, new HashSet<Integer>());

}

// dislikes[i] = [ai, bi] indicates that the person labeled ai does

// not like the person labeled bi, if we treat dislikes[i] as an edge,

// the two nodes at both ends on this edge cannot assign into same group

for(int[] dislike : dislikes) {

adj.get(dislike[0]).add(dislike[1]);

adj.get(dislike[1]).add(dislike[0]);

}

int[] colors = new int[n + 1];

// Graph traversal start on each node

for(int i = 1; i <= n; i++) {

// Initially set non-colored node as 1, if not able to paint then not applicable

if(colors[i] == 0 && !helper(i, adj, colors, 1)) {

return false;

}

}

return true;

}

private boolean helper(int i, Map<Integer, Set<Integer>> adj, int[] colors, int color) {

// If node already painted, check if its current color match the color try

// to paint on it, if not match means not bipartite graph

// e.g we try to paint this node as color = 1, but if this node already

// painted as -1, not as expected, then not bipartite graph

if(colors[i] != 0) {

return colors[i] == color;

}

colors[i] = color;

// Additional check than 785. Is Graph Bipartite? since the graph in this

// problem is a raw graph, contains remained nodes rather than 785 which

// not contains remained nodes, and only when node value == 1 means it

// is the vertex of an edge given in dislikes and present in graph, if

// graph.get(next) == null means its not the vertex of an edge which setup = 1

// when build the graph, not belong to the edge, then not able to apply

// dfs on the node

if(adj.get(i) != null) {

// Flip the color and attempt for all neighbour nodes, if any attempt

// failure then not bipartite graph

for(int j : adj.get(i)) {

if(!helper(j, adj, colors, -color)) {

return false;

}

}

}

return true;

}

}

Time Complexity : O(V + E)

Space Complexity: O(V + E)

**Refer to**

<https://www.cnblogs.com/grandyang/p/10317141.html>

这道题又是关于二分图的题，第一次接触的时候是 Is Graph Bipartite?，那道题给的是建好的邻接链表（虽然是用数组实现的，但是本质上和这道题是一样的，同一条边上的两点是不能在同一个集合中的，那么这就相当于本题中的 dislike 的关系，也可以把每个 dislike 看作是一条边，那么两端的两个人不能在同一个集合中。看透了题目的本质后，就不难做了，跟之前的题相比，这里唯一不同的就是邻接链表没有给我们建好，需要自己去建。不管是建邻接链表，还是邻接矩阵都行，反正是要先把图建起来才能遍历。那么这里我们先建立一个邻接矩阵好了，建一个大小为 (N+1) x (N+1) 的二维数组g，其中若 g[i][j] 为1，说明i和j互相不鸟。那么先根据 dislikes 的情况，把二维数组先赋上值，注意这里 g[i][j] 和 g[j][i] 都要更新，因为是互相不鸟，而并不是某一方热脸贴冷屁股。下面就要开始遍历了，还是使用染色法，使用一个一维的 colors 数组，大小为 N+1，初始化是0，由于只有两组，可以用1和 -1 来区分。那么开始遍历图中的结点，对于每个遍历到的结点，如果其还未被染色，还是一张白纸的时候，调用递归函数对其用颜色1进行尝试染色。在递归函数中，现将该结点染色，然后就要遍历所有跟其合不来的人，这里就发现邻接矩阵的好处了吧，不然每次还得遍历 dislikes 数组。由于这里是邻接矩阵，所以只有在其值为1的时候才处理，当找到一个跟其合不来的人，首先检测其染色情况，如果此时两个人颜色相同了，说明已经在一个组里了，这就矛盾了，直接返回 false。如果那个人还是白纸一张，我们尝试用相反的颜色去染他，如果无法成功染色，则返回 false。循环顺序退出后，返回 true

**Classic adjacency list**

class Solution {

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

// Initial a N \* N 2D array prepared for build graph

int[][] graph = new int[N][N];

// Build graph with given 2D array dislikes, recognize all

// presented relations in dislikes as an edge as mutual way

// (graph[i][j] and graph[j][i]) and set as both nodes to 1

// to distinguish them from remained nodes (Note: this is

// the major difference than 785. Is Graph Bipartite?, Since

// in 785 the problem give graph directly with all edges

// presented in graph already and no remained nodes)

for(int[] d : dislikes) {

graph[d[0] - 1][d[1] - 1] = 1;

graph[d[1] - 1][d[0] - 1] = 1;

}

int[] colors = new int[N];

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

// Initially set non-colored node as 1, if not able to paint then not applicable

if(colors[i] == 0 && !helper(i, graph, colors, 1)) {

return false;

}

}

return true;

}

private boolean helper(int node, int[][] graph, int[] colors, int color) {

// If node painted, check if it match the color try to paint on it

// if not match means not bipartite graph

if(colors[node] != 0) {

return colors[node] == color;

}

colors[node] = color;

for(int i = 0; i < graph[node].length; i++) {

// Additional check than 785. Is Graph Bipartite? since the graph in this

// problem is a raw graph, contains remained nodes rather than 785 which

// not contains remained nodes, and only when node value == 1 means it

// is the vertex of an edge given in dislikes and present in graph, if

// graph[node][i] != 1 means its not the vertex of an edge which setup = 1

// when build the graph, not belong to the edge, then not able to apply

// dfs on the node

if(graph[node][i] == 1) {

// The adjacent nodes paint with different color as -1

if(!helper(i, graph, colors, -color)) {

return false;

}

}

}

return true;

}

}

**DFS + Build graph with map**

class Solution {

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

// Initial graph as map

Map<Integer, Set<Integer>> graph = new HashMap<Integer, Set<Integer>>();

for(int[] d : dislikes) {

int a = d[0];

int b = d[1];

graph.putIfAbsent(a, new HashSet<Integer>());

graph.putIfAbsent(b, new HashSet<Integer>());

graph.get(a).add(b);

graph.get(b).add(a);

}

int[] colors = new int[N + 1];

for(int i = 1; i <= N; i++) {

// Initially set non-colored node as 1, if not able to paint then not applicable

if(colors[i] == 0 && !helper(i, graph, colors, 1)) {

return false;

}

}

return true;

}

private boolean helper(int node, Map<Integer, Set<Integer>> graph, int[] colors, int color) {

// If node painted, check if it match the color try to paint on it

// if not match means not bipartite graph

if(colors[node] != 0) {

return colors[node] == color;

}

colors[node] = color;

// Additional check than 785. Is Graph Bipartite? since the graph in this

// problem is a raw graph, contains remained nodes rather than 785 which

// not contains remained nodes, and only when node value == 1 means it

// is the vertex of an edge given in dislikes and present in graph, if

// graph.get(next) == null means its not the vertex of an edge which setup = 1

// when build the graph, not belong to the edge, then not able to apply

// dfs on the node

if(graph.get(node) == null) {

return true;

}

for(int next : graph.get(node)) {

// The adjacent nodes paint with different color as -1

if(!helper(next, graph, colors, -color)) {

return false;

}

}

return true;

}

}