You have an undirected graph with n nodes, each represented by a number between 0 and n - 1. The graph is described by a 2D array graph, where graph[u] is an array containing nodes adjacent to node u. In other words, there is an undirected edge between node u and each node v present in graph[u]. The graph satisfies the following conditions:

There are no self-edges, meaning that node u is not present in graph[u].

There are no parallel edges, implying that each node appears only once in the adjacency list.

The graph is undirected, so if node v is adjacent to node u, then node u is also adjacent to node v (i.e., an edge between u and v implies an edge between v and u).

The graph may not be fully connected, so there can be two nodes u and v such that there is no direct path between them.

The task is to determine whether the given graph is bipartite or not. A graph is considered bipartite if its nodes can be partitioned into two independent sets, denoted as set A and set B, such that each edge connects a node from set A to a node from set B.

Example:

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

Output : false

0 -- 1

| |

3 -- 2

Important Note: Ensure that you save your solution before progressing to the next question and before submitting your answer.

Exercise-1

Input :

4

1 2 3

0 2

0 1 3

0 2

Output : false

Exercise-2

Input:

4

1 2 3

0 2

0 1

0

Output: false

Exercise-3

Input:

4

1 3

0 2

1 3

0 2

Output:true

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input number of nodes

int n = sc.nextInt();

// Initialize the graph (adjacency list)

List<List<Integer>> graph = new ArrayList<>();

// Input the adjacency list

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

List<Integer> adjList = new ArrayList<>();

String line = sc.nextLine();

String[] parts = line.split(" ");

for (String p : parts) {

if (!p.isEmpty()) {

adjList.add(Integer.parseInt(p));

}

}

graph.add(adjList);

}

// Check if the graph is bipartite

boolean result = isBipartite(graph, n);

System.out.println(result);

}

public static boolean isBipartite(List<List<Integer>> graph, int n) {

int[] colors = new int[n];

Arrays.fill(colors, -1); // -1 means uncolored

// Try to color the graph

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

if (colors[i] == -1) { // Not colored yet

if (!bfsCheck(graph, colors, i)) {

return false;

}

}

}

return true;

}

public static boolean bfsCheck(List<List<Integer>> graph, int[] colors, int startNode) {

Queue<Integer> queue = new LinkedList<>();

queue.offer(startNode);

colors[startNode] = 0; // Start with color 0

while (!queue.isEmpty()) {

int node = queue.poll();

int currentColor = colors[node];

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

if (colors[neighbor] == -1) { // If not colored, color it with opposite color

colors[neighbor] = 1 - currentColor;

queue.offer(neighbor);

} else if (colors[neighbor] == currentColor) {

// If the neighbor has the same color, it's not bipartite

return false;

}

}

}

return true;

}

}