```
class Solution {
    public boolean isBipartite(int[][] graph) {
        int[] colors = new int[graph.length];
        for (int i = 0; i < graph.length; i++) {
            if (colors[i] == 0 && !isValidColor(graph, colors, 1, i)) {
                return false;
            }
        }
        return true;
    }

    private boolean isValidColor(int[][] graph, int[] colors, int color, int node) {
        if (colors[node] != 0) {
            return colors[node] == color;
        }

        colors[node] = color;
        for (int g : graph[node]) {
            if (!isValidColor(graph, colors, -color, g))
                return false;
        }
        return true;
    }
}
```

Given an undirected `graph`, return `true` if and only if it is bipartite.

Recall that a graph is *bipartite* if we can split it's set of nodes into two independent subsets A and B such that every edge in the graph has one node in A and another node in B.

The graph is given in the following form: `graph[i]` is a list of indexes `j` for which the edge between nodes `i` and `j` exists. Each node is an integer between `0` and `graph.length - 1`. There are no self edges or parallel edges: `graph[i]` does not contain `i`, and it doesn't contain any element twice.

Example 1:
Input: `[[1,3], [0,2], [1,3], [0,2]]`
Output: `true`
Explanation:
`The graph looks like this:`
`0----1`

```
|    |
|    |
3----2
```
We can divide the vertices into two groups: {0, 2} and {1, 3}.

Example 2:

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

Output: false

Explanation:

The graph looks like this:

```
0----1
| \  |
|  \ |
3----2
```
We cannot find a way to divide the set of nodes into two independent subsets.

Note:

- `graph` will have length in range `[1, 100]`.
- `graph[i]` will contain integers in range `[0, graph.length - 1]`.
- `graph[i]` will not contain `i` or duplicate values.
- The graph is undirected: if any element `j` is in `graph[i]`, then `i`will be in `graph[j]`.