785. Is Graph Bipartite?

Given a `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 ubsets.
```

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.

Seen this question in a real interview before?

- Difficulty:Medium
- Total Accepted:1.5K
- Total Submissions:4.1K
- Contributor:awice
-
- Subscribe to see which companies asked this question.

`Our goal` is trying to use two colors to color the graph and see if there are any adjacent nodes having the same color.
Initialize a color[] array for each node. Here are three states for colors[] array:
`-1: Haven't been colored.`
`0: Blue.`
`1: Red.`
For each node,

1. If it hasn't been colored, use a color to color it. Then use another color to color all its adjacent nodes (DFS).
2. If it has been colored, check if the current color is the same as the color that is going to be used to color it. (Please forgive my english… Hope you can understand it.)

```
class Solution {
    public boolean isBipartite(int[][] graph) {
        int n = graph.length;
        int[] colors = new int[n];
        Arrays.fill(colors, -1);

        for (int i = 0; i < n; i++) {              //This graph might
be a disconnected graph. So check each unvisited node.
            if (colors[i] == -1 && !validColor(graph, colors, 0, i))
{
                return false;
            }
        }
        return true;
    }

    public boolean validColor(int[][] graph, int[] colors, int color,
int node) {
        if (colors[node] != -1) {
            return colors[node] == color;
        }
        colors[node] = color;
        for (int next : graph[node]) {
            if (!validColor(graph, colors, 1 - color, next)) {
                return false;
            }
        }
        return true;
    }
}

//C++
bool isBipartite(vector<vector<int>> &graph)
{
     int n = graph.size();
     vector<int> color(n,-1);
     for(int i=0;i<n;++i)
     {
          if(color[i]==-1 && !isValidGraph(graph,color,0,i)) return
false;
     }
     return true;
}

bool isValidGraph(vector<vector<int>> &graph, vector<int> &colors,
int color, int node)
{
     if(colors[node]!=-1)
     {
          return colors[node]==color;
     }
     colors[node]=color;
     vector<int> neighbours = graph[node];
     for(auto nei:neighbours)
     {
          if(!isValidGraph(graph,colors,1-color,nei))
               return false;
     }
     return true;
}
```