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.

Solution 1

`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 disconne
cted 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;
    }
}
```

written by flagbigoffer original link here

Solution 2

```
    def isBipartite(self, graph):
        color = defaultdict(lambda: -1)
        return all(self.dfs(graph, v, edges, 0, color) for v, edges in enumerate(gra
ph) if color[v] == -1)

    def dfs(self, graph, v, edges, cur_color, color):
        if color[v] != -1: return color[v] == cur_color
        color[v] = cur_color
        return all(self.dfs(graph, e, graph[e], int(not cur_color), color) for e in
edges)
```

written by johnyrufus16 original link here

Solution 3

We have a graph, and we need to create two sets of nodes. eg: [[1,3], [0,2], [1,3], [0,2]]
We can rewrite this in terms of adjacency list as :
Node => adacency list
0 => [1,3]
1 => [0,2]
2 => [1,3]
3 => [0,3]

Let the final set be first , second and initialized to empty.
Now, for each node in the graph, check following conditions :
1. if node is first set, then all the adjacency nodes are to be added to second list. If any of the adjacent nodes are in first set then return false.
2. if node is second set, then all the adjacency nodes are to be added to first list. If any of the adjacent nodes are in first set then return false.
3. if node is not present neither in first nor second set, then check if any of the nodes in the adjacent list are in first or second.
a. If any of the node is in second list, then add node to first list and remaining nodes to second list.
b. If any of the node is in first list, then add node to first list and remaining nodes to first list.

If all nodes are processed then return true.

```python
class Solution:
    def isBipartite(self, graph):
        """
        :type graph: List[List[int]]
        :rtype: bool
        """
        first  = []
        second = []
        for index,i in enumerate(graph):

            f = False
            s = False
            # case 3
            if index not in first and index  not in second:
                for j in i :
                    if j in first:
                        f= True
                        s= False
                        break
                    elif j in second:
                        f= False
                        s= True
                        break
                if f:
                    second.append(index)
                else:
                    first.append(index)
            # case 1
            if index in first :
                for j in i:
                    if j in first :
                        return False
                    second.append(j)
            # case 2
            if index in second :
                for j in i :
                    if j in second:
                        return False
                    first.append(j)

        return True
```

written by venkatagogu original link here