JavaScript Is Graph Bipartite?

Challenge

There is an undirected graph with `n` nodes, where each node is numbered between `0` and `n - 1`. You are given a 2D array `graph`, where `graph[u]` is an array of nodes that node `u` is adjacent to. More formally, for each `v` in `graph[u]`, there is an undirected edge between node `u` and node `v`. The graph has the following properties:

- There are no self-edges (`graph[u]` does not contain `u`).
- There are no parallel edges (`graph[u]` does not contain duplicate values).
- If `v` is in `graph[u]`, then `u` is in `graph[v]` (the graph is undirected).
- The graph may not be connected, meaning there may be two nodes `u` and `v` such that there is no path between them.
- A graph is bipartite if the nodes can be partitioned into two independent sets `A` and `B` such that every edge in the graph connects a node in set `A` and a node in set `B`.

Return `true` if and only if it is bipartite.

1st Example

```
Input: graph = [[1,2,3],[0,2],[0,1,3],[0,2]]
Output: false
```

Example continues on next page...

```
Explanation: There is no way to partition the nodes into two
             independent sets such that every edge connects
             a node in one and a node in the other.
```

2nd Example

```
Input: graph = [[1,3],[0,2],[1,3],[0,2]]
Output: true
Explanation: We can partition the nodes into
             two sets: {0, 2} and {1, 3}.
```

Constraints

- `graph.length == n`
- `1 <= n <= 100`
- `0 <= graph[u].length < n`
- `10 <= graph[u][i] <= n - 1`
- `graph[u]` does not contain `u`.
- All the values of `graph[u]` are unique.
- If `graph[u]` contains `v`, then `graph[v]` contains `u`.

Solution

```
const isBipartite = (graph) => {
    const n     = graph.length,
          color = Array(n).fill(0);
```

Solution continues on next page...

```
    for (let i = 0; i < n; i++) {
        if (color[i]) {
            continue;
        }

        const queue = [i];

        color[i] = 1;

        while(queue.length) {
            const curr = queue.shift();

            for (let next of graph[curr]) {
                if (color[next] === color[curr]) {
                    return false;
                }

                if (!color[next]) {
                    color[next] = color[curr] === 1 ? 2 : 1;

                    queue.push(next);
                }
            }
        }
    }

    return true;
};
```

Explanation

I've built a function called `isBipartite` that takes in a graph as input. Its purpose is to check whether the given graph is bipartite or not.

A bipartite graph is a graph whose vertices can be divided into two disjoint sets such that every edge connects a vertex from one set to a vertex from the other set. In other words, the graph can be colored using only two colors, such that no two adjacent vertices have the same color.

The function starts by initializing two variables: `n` which represents the number of nodes in the graph, and `color` which is an array of length `n` filled with `0`s. This array will be used to keep track of the color assigned to each node.

Next, the function enters a `for` loop that iterates over each node in the graph. Inside the loop, it checks if the current node has already been assigned a color. If it has, the function continues to the next iteration of the loop.

If the current node has not been assigned a color, it creates an empty queue and adds the current node to it. It then assigns the color `1` to the current node.

The function enters a `while` loop that continues as long as the queue is not empty. Inside the loop, it removes the first element from the queue and assigns it to the variable `curr`.

For each neighbor of the current node, the function checks if the neighbor has the same color as the current node. If they have the same color, it means the graph is not bipartite, so it returns false.

If the neighbor has not been assigned a color, it assigns a color to it based on the color of the current node (if the current node has color `1`, it assigns color `2` to the neighbor, and vice versa). It then adds the neighbor to the queue.

After the `while` loop ends, the function returns true, indicating that the graph is bipartite.