CS 314 Final Review — Bipartite Graphs — **Solution**

Graphs

Write an instance method for the provided `Graph` class which determines whether a graph is bipartite. A graph is bipartite if the set of vertices can be partitioned into two disjoint sets such that no edges exist between nodes of the same subset. One way to determine if a graph is bipartite is to check if the graph can be colored using 2 colors. If each vertext can be colored with either one of two colors such that no two vertices of the same color are adjacent, then the graph is bipartite.

Assume we are only dealing with unweighted, undirected graphs.

Complete the following instance method.

```
// Determines whether or not a graph is bipartite.
// pre: vertices.size() > 0
// post: the structure of the graph is not altered by this operation
public boolean isBipartite() {

}
```

Use the following Graph implementation.

```
public class Graph {
  // The vertices in the graph.
  private Map<String, Vertex> vertices;

  // Sets scratch to 0 for all vertices
  private void clearAll(){ /* ... */ }

  private static class Vertex {
    private String name;
    private List<Edge> adjacent;
    private int scratch;
  }

  private static class Edge {
    private Vertex dest;
  }

}
```

You may create helper methods.
You may not create any new data structures.
Do not use any other Java classes or methods.

```
// Determines whether or not a graph is bipartite.
// pre: none
// post: returns true iff graph is bipartite
public boolean isBipartite(){
    clearAll();
    String start = vertices.keySet().iterator().next();
    vertices.get(start).scratch = COLOR_2;
    return bipartiteHelper(true, start);
}

private static final int COLOR_1 = 1;
private static final int COLOR_2 = -1;
public boolean bipartiteHelper(boolean color, String currName){
    Vertex curr = vertices.get(currName);
    int adjColor = color ? COLOR_1 : COLOR_2;
    for(Edge e : curr.adjacent){
        Vertex dest = e.dest;
        if(dest.scratch == 0){
            dest.scratch = adjColor;
            if(!bipartiteHelper(!color, dest.name))
                return false;
        }
        else if(dest.scratch != adjColor){
            return false;
        }
        // If vertex's color is already correct, perfect, keep checking!
    }
    return true;
}
```

This is a tricky graph and recursion problem. When checking whether a graph is bipartite, it doesn't matter which vertex we start on nor which color we assign to the first vertex. We will use the scratch field in the vertex class to keep track of which color we've assigned to the vertex. I arbitrarily chose 1 and -1 as colors and 0 signifies we haven't colored it yet. So, first thing we do is we choose arbitrarily choose a vertex and color to start on. Then, to be bipartite, all vertices adjacent to the first vertex need to be the assigned the other color.

This pattern continues recursively. We can now consider the vertices we just colored as the new starting vertices and color their neighbors the other color. We alternate current colors as we go deeper into the recursive calls (that is the reasoning behind passing `!color` into the recursive helper method when making the recursive call).

If we ever come across a vertex which has already been colored, but not with the color we expect/want it to be, then we can be sure that the graph is not bipartite. In that case, we can return `false` to signal up the call stack that the graph is not bipartite.