Alex Greene CPE349 10/15/2015 LAB #2

Description

My algorithm works to detect bipartiteness by determining the two-colorability of a graph. By assigning a color to an arbitrary source vertex, then exploring all neighbors of that vertex and assigning them the alternate of 2 colors, the algorithm attempts to color the entire graph, alternating colors with each call. If the algorithm reaches a point where a vertex is going to be colored the same as its neighbor, we know the graph is not bipartite. The algorithm also returns whether or not a graph is connected, by checking if multiple iterations of the algorithm are necessary to reach all of the vertices.

Pseudocode

```
Given a graph, G
DepthFirstSearch(G)
    For each vertex, v, in G,
        If this is not the first iteration, we know the graph is not connected
        If vertex color has not yet been set, assign RED
        Explore(v, BLUE)

Explore(v, color)
    For each neighboring vertex, n, of v
        If the color of n is the same as the color of v
            We know the graph is not bipartite
        Else if the color of n has not been set
            Set n's color to the color passed in by 'color' param
            Explore(n, alternate color to 'color' param)
```

Testing

For testing my program, I ran the three provided test cases as well as four others that tested connected but not bipartite, bipartite but not connected, bipartite and connected, and not bipartite and not connected.