Sreeram Danda
CS577 Homework #5
Problem #4

**#4a) Prove that a graph is bipartite iff it can be colored with two colors.**

1) If a graph is bipartite then it is colored with two colors:

*Proof:* Let G be a bipartite graph that has partitions A, B. Let the colors be the set S = {R, G}. Then, we can create a function *f* such that *f(v) = R,* if $v \in A$. Similarly, *f(v) = B,* if $v \in B$. Therefore, if a graph is bipartite then it can be colored with two colors.

2) If a graph is two colored, then it is bipartite:

*Proof:* Let G be a two colored graph where R and G and no vertex is connected to a vertex of the same color. Since, there cannot be an edge between G-G or R-R, one can group the vertices into a set A for G vertices and set B for R vertices. Since, there are two groupings with no edges within each group the graph is bipartite.

**#4b) Two Coloring Algorithm**

```
color_bipartite (G)
  |   s = random vertex from G
  |   s.visited = true
  |   s.color = left
  |   Q = a queue
  |   Q.enqueue(s)
  |   while(Q is not empty)
  |     |   v ← dequeue( )
  |     |   for (All neighbors w of v)
  |     |     |   if (w.visited == false)
  |     |     |     |   Q.enqueue(w)
  |     |     |     |   w.color = opposite of v
  |     |     |     |   w.visited = true
  |     |     |   else if(w.visited == true && w.color == v.color)
  |     |     |     |   return NO
```

Time Complexity: The algorithm performs a breadth first search traversal which has a runtime of O(N) and for each vertex the algorithm performs constant time operations. Therefore, the runtime complexity for this algorithm is O(N).

Correctness: The vertices/edges are visited in breadth first search ordering ie,. in order of distance from the start vertex. Moreover, every non-tree edge found in the traversal is either between vertices at the same or different levels. If the algorithm does not return NO, then every vertex has been assigned a coloring and there are no edges between vertices on the same side. If the algorithm does return NO, this is because the colors set by the algorithm depend only on the parity of the corresponding vertex's level. It must be the case that there is an edge between two vertices on the same level. Let x be the least common ancestor of w, v. Let i be x's level and j be the level w and v are on. There are j - i paths from x to both w and v. If we add the edge (w, v) to these two paths, we get a cycle x → w → v → x of length 2(j - i)

+ 1. Since this is an odd length cycle it cannot be colored with two colors based on the proof provided in **4a**. Therefore, the algorithm correctly returns NO.

**#4c) Number of Colors for One Odd Cycle**

If a graph only has one cycle of odd length then it requires 3 colors to color the entire graph. Alternating a color for each vertex in that cycle one can color all vertices such that its neighbors are different colors than itself. This is because for two vertices they can be colored with two colors. So repeating the coloring n times we can cover 2n vertices with two colors. If however we have an odd number of vertices in the cycle, upon reaching the last vertex one of the neighbors will be one color and another a different color. In order to have a distinct color from its neighboring vertices the last vertex must be a different color than both.

**#4d) Efficient Algorithm Two Coloring w/ One Odd Cycle**

```
color_with_odd_cycle (G)
  |   s = random vertex from G
  |   s.visited = true
  |   s.color = red
  |   Q = a queue
  |   Q.enqueue(s)
  |   while(Q is not empty)
  |     |   v ← dequeue( )
  |     |   for (All neighbors w of v)
  |     |     |   if(w.visited == false && v.color == null)
  |     |     |     |   w.color = v.parent.color
  |     |     |   else if (w.visited == false)
  |     |     |     |   w.color = opposite of v (if red -> blue, Vice Versa)
  |     |     |   else if(w.visited == true && w.color == v.color)
  |     |     |     |   v.color = null
  |     |     |   Q.enqueue(w)
  |     |     |   w.visited = true
```

Time Complexity: The algorithm performs a breadth first search traversal which has a runtime of O(N) and for each vertex the algorithm performs constant time operations. Therefore, the runtime complexity for this algorithm is O(N).

Correctness: The proof of correctness is exactly the same as **4b** except, rather than returning NO when there is a cycle of odd length found, the algorithm removes the color of the vertex that has conflict and continues normally. Now the problem is to address what occurs to the children of the conflict vertex. Since the color is assigned via the level of the vertex, the algorithm correctly assigns the color as the color of the parent of the conflict vertex. Since, there are only two colors, a color at level i will repeat at i+2n levels. Thus, the children's vertices are colored correctly.