Jennifer Manning

Project 6

1)

```
bool Wrestlers( Graph G, Vertex root )
        queue Q;
        bool state = true; //this coresponds to good
        root.setType = “good”
        Q.enqueue( root );

        while !Empty ( Q )
                t = Q.dequeue();
                state = !state;
                t.setType = state ? “good” : “bad”;
                for all vertices V in t.adj
                        u = adj[iterator];
                        if (u type is not set)
                                u.SetType = state ? “good” : “bad”;
                        else if u.GetType == t.GetType
                                return false;
        return true;
```

In this example, each wrestler is a vertex, and it's realtionship with another wrestler is an edge. Therefore, each adjacent wrestler must be opposite type as the previous. We must assume that the root, in this case I assumed a good guy. Then we iterate through the neighbors, and label them as bad guys. Then each of their neighbors must be good; however, each neighbor must be checked if they are already labeled. If a good guy's neighbor is already a good guy then it is impossible to create the graph described.

2)

a. { (3, A, B), (2, B, E), (1, E, I), (2, E, H), (3, B, F), (2, F, G), (1, C, G), (4, A, D), (7, I, J)}
b. { (1, C, G), (1, E, I), (2, B, E), (2, E, H), (2, F, G), (3, A, B), (3, B, F), (4, A, D), (7, I, J)}

3)

```
void Graph::DFS(Vertex v, unsigned &time){
v.visited = true;
v.discovered = ++time;
foreach Veretex w adjacent to v
if(!w.visited)
        edge.set(tree)
        dfs(w, time);
else if( w.explored == 0)
        edge.set(back)
else if (v.discovered < w.discovered)
        edge.set(forward)
else
        edge.set(cross)
 break //for
 v.explored = ++time;
}
```

4)

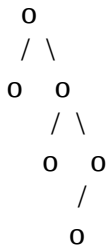
Assume there is some intial vertex, Vi such that deg(Vin) != deg(Vout).

Case1: deg(Vin)>deg(Vout)
After all simple paths involving Vi have been traversed and the current vertex is Vi,
deg(Vout) =0
however, deg(Vin)>0
so, it is impossible to leave Vi,
therefore, there are edges adjacent to Vi that are unable to be traversed
which does not follow the definition of a Euler path

Case2: deg(Vin)<deg(Vout)
After all simple paths involving Vi have been traversed
and the current vertex is not Vi,
deg(Vin) =0
however, deg(Vout)>0
so, it is impossible to return to Vi which still has outgoing nodes
therefore, there are edges adjacent to Vi that are unable to be traversed
which does not follow the definition of a Euler path

5)
a) An example of a graph that has n-1 vertices in the discovered state would be a graph in which every vertex is adjacent to the first node to be explored. During the exploration of Vo, every node but itself would be discovered but not explored.

b) A graph is which Θ(n) vetices are in the discovered state is one in which there is a linear graph, but every vertex has one other vertex. Such as:



In this case, exactly ½ will be discovered.

c) A graph in which Θ(n) vertexes are in the undiscovered state, Θ(n) vertexes are in the completely explored state, and only Θ(1) vertexes are discovered is one in which half the vertices are in a linear line and ½n -1have been completely explored and then the following vertex is connected to every node remaining. All of those vertices would be undiscovered, the only discovered vertex would be the next one in the linear line.