Ryan Mysliwiec
HW 11-1

a) Proof of statement

Base Case: A Tree, G, has zero nodes so the game would immediately end. A Tree, G, has one node, so the game would end. If there is a cycle, G' = G.

Induction Hypothesis: A Tree, G, has k nodes. If the tree was built using BFS, each node's children would consist of its neighbors. Each episode, the node with the least number of children would be deleted, with its children being rearranged to their other friendships.

If the tree was built using DFS, then the node's children reflect the path from the initial node to the last node it can reach. At each episode, the shortest path's parent node will be removed and the tree will be slightly re-arranged, similar to the way the BFS tree would.

Both of these methods will eventually work their way down to being a tree of just, at most, a tree of 2 vertices, where G' would be empty, like in the example given in the assignment description.

Inductive Step: When there are k+1 nodes, the DFS and BFS modes work the same way. There will be another child of a node for BFS and one of the paths will be 1 node longer for DFS. Otherwise, it works exactly the same way.

b) **Survivor Graph**

Pseudocode

```
DFS(G)
      For each vertex u in G.V //G.V is that set of vertexes in G
            u.color = white //white=unvisited, grey=visited,
                            black=finished
            u.pred = nil    //predecessor of current node
      time = 0   //counter initialization
      survivors = []
      for each vertex u in G.V
            if u.color == white
                  DFS-Visit(G,u)

      for each n in G.V
            if n.keep == 1
                  survivors.append(n)

       return survivors

DFS-Visit(G,u)
```

```
      time = time + 1
      u.start = time
      u.color = grey

      for each v in G.Adj[u]      //G.Adj[u] is simply a
                                  collection of the neighbors of u
            if v.color == white
                  v.pred = u
                  DFS-Visit(G,v)
            Else if v.color == grey and u.pred != v
                  v.keep = 1
                  Find-Cycle(G,u,v)
       u.color = black
       time = time + 1
       u.finish = time


Find-Cycle(G,u,v)
      u.keep = 1
      if u.pred != v and u.pred != nil
            Find-Cycle(G,u.pred,v)
```

Description

This is an augmented version of the DFS algorithm in the book. It initializes all the nodes' color to white, the predecessors to nil, a list of survivor nodes for the survivor graph to empty and the time to 0. It then runs the traversal across every node in the graph.

In the traversal algorithm, the time is increased and set as the node's start time, with the color being set to grey. For each of the node's neighbors, the algorithm checks the color neighbor. If white, the neighbor's predecessor is set to the current node and the traversal algorithm is recursively called on the neighbor. Otherwise, if the neighbor's color is grey and the current node's predecessor is not a neighbor (meaning there is already a cycle), the neighbor is marked as a node to keep in the survivor list and a function to check for a cycle is called. Finally, the current node's color is set to black, and the time is increased and set to be the current node's finishing time.

The function to find a cycle sets the current node to be kept, and if there is no cycle (like previously mentioned) and the predecessor is not nil, it will call the cycle function again, but on the predecessor and neighbor of the current node.

Finally, the initial code will run through the graph again and check for which nodes have been marked to keep. It adds all of these to a list that is returned.

Proof of Correctness

Base Case: A Tree, G, has zero nodes so the game would immediately end. A Tree, G, has one node, so the game would end. A Tree, G, has two nodes, so both nodes would be deleted and the game would end. If there is a cycle, G' = G.

The algorithm will be run on every node in the graph, and will thus traverse all possible paths. Because it traverses all possible paths, it also checks a cycle every time it is necessary. This shows that the algorithm will correctly create a list of nodes for G'.

Running Time

Each of the for loops in the DFS algorithm run over all n nodes in the graph, so it runs in O(3n)=O(n) time. In the traversal functions, the for loop runs over every edge connected to the node it is called on, which is at most m edges. This runs in O(m) time. Together, the entire algorithm runs in a total of O(m + n) time.