NAMES: John Karuntzos / Dakota Krogmeier / Kristopher Hoce

Worksheet 19: Work with either ONE or TWO other students to complete this worksheet. You must work with at least one other student on this assignment, and should submit one paper.

1. Either prove or give a counter-example (by drawing a relevant directed graph and explaining why your graph is relevant by clearly showing that the hypothesis is met but the conclusion fails to hold): if there is a path from u to v in a **directed** graph G, and if d[u] < d[v] in a **depth-first search** of G, then v is a descendant of u in the depth-first forest produced. (2 pts)

   if there is a path from u to v in a directed graph and d[u] < d[v], then v is a descendent of u. We have to finish our dfsVisit(v) before we finish our dfsVisit(u), so that the finishing time of v is is earlier than the finishing time of u, otherwise our nodes are disjointed. Because we know that v is a descendent that means we discovered v while expanding u, so we know that d[u] < d[v].

2. Describe an algorithm that determines if an undirected graph G = (V, E) contains a cycle. Your algorithm should take, as input, a graph (you should specify how the graph is represented), and returns True if the graph contains a cycle, and False if it does not. (4 points)

KEY IDEA:

When making this program, the goal is to use a helper function that will use recursion to will go over all the nodes and if one node is visited twice then it says it's a cycle. Then this helper function will be used in the main function to test the cycle where it will return true or false depending on the helper functions findings. To start, mark all the nodes as unvisited in the undirected graph. Then after choosing the starting vertex, mark that node only of being visited to. After that, there will be recursive checks to all the adjacent nodes to the starting vertex. If the node has not been traveled to, go to that new node and do the same recursive check to all the nodes connected to that. During the recursive checks to adjacent nodes, if the node comes across a node that has already been traveled to, that isn't the parent node, then that counts as a cycle shows that there is a cycle in the undirected graph. This helper function will then be called in the main hascycle function, where it will make all nodes not visited again and then iterate through the graph to return true or false based on whether it has a cycle.

Algorithm HasCycle(G):

Input: Graph G of vertices and edges

Output: True if G has a cycle, false otherwise

PROCESS:

```
HasCycle(G):
    visited <— []
    recursed <— []
    for i <— 0 to V:
        if helper(i, visited, recursed) = true:
            return true
    return false


helper(i, visited, recursed):
    if recursed[i] = true:
        return true
    if visited[i] = true:
        return false
    visited[i] <— true
    recursed[i] <— true
    children <— adjacent[ i ]    //All adjacent vertices of i

    for c in children:
        if helper(c, visited, recursed) = true:
            return true
    recursed[ i ] <— false
    return false
```