CS 180 Homework 2

Prithvi Kannan

405110096

1. Exercise 3 Page 107

The following algorithm finds a topological order of G if G is a DAG or a cycle if G is not a DAG.

```
To compute a topological ordering of G (not necessarily a DAG):

Find a node v with no incoming edges and order it first

    Delete v from G

    Recursively compute a topological ordering of G–{v} and append this order
after v

If there are no nodes with no incoming edges

       Initialize a list of nodes

       Pick the first element in the adj list for each node

              If unique, add to list

              If revisiting, return the set of nodes between the visits

       Return the list of nodes
```

2. Exercise 4 on Page 107

Construct a graph showing the relationship of specimens. If (i, j) is marked as "different", we say there is an edge between node i and node j. Observations marked as "same" or "ambiguous" do not need to be indicated on the graph. The following algorithm uses DFS to search for bipartiteness in G. Since G is not necessarily a connected graph (since "same" and "ambiguous" relations produce detached nodes), we must iterate over components of G.

```
For components C with more than 1 node in G

       Start at an arbitrary node

              Add to set U

              Assign all neighboring nodes to set V

              Assign all of the neighbor's neighbors to Set U

              Continue for all nodes in component C

              While assigning colors, if we find a neighbor which is in the
              same set as current vertex, then the observations do not line up.
```

The runtime of this algorithm is $O(m + n)$. Creating the graph G will take $O(m)$ for m observations. Running BFS will take $O(m + n)$.

3. Exercise 9 on page 110

We use BFS to produce a spanning tree of G starting with the root node s. The spanning tree will have levels, which will represent the length of the shortest path from s to the nodes in that level of the spanning free. We are given that the length of $s - t$ is strictly greater than $n/2$ so the node t must be past the level $n/2 + 1$. Now we look at the levels between s and t. The total number of nodes in levels 1 through $n/2$ is at most $n - 2$ (since nodes s and t don't appear in any of these levels). If each of these $n/2$ levels has 2 or more nodes, then total number of nodes in G will exceed n. Therefore, there must be at least one level in the range 1 through $n/2$ containing just one node, say v. If we remove the only node in that level, v, then we break the path from s to t.

```
Construct a BFS spanning tree of G

Construct a list of the nodes at each level of the tree

For any level i where 1 ≤ j ≤ n/2 and there is only 1 node in level 1 (called
v)

    return v
```

4. Exercise 11 on page 111

We constructed a directed graph. For a triple (C_i, C_j, t_k), we add nodes (C_i, t_k) and (C_j, t_k) with directed edges in both directions. Simultaneously, we maintain a list for C_i and C_j. If the nodes are not the first in the list, add a directed edge from the previous element in the list to the current node.

```
Initialize G and an array of linked lists representing each computer

Loop through all of the given triples

    Add nodes (C_i,t_k) and (C_j,t_k) to G

    Add dir edge from (C_i,t_k)->(C_j,t_k) and (C_j,t_k)->(C_i,t_k)

    If C_i's list not empty

        Add edge from last element in list (C_i,tj) to (C_i,t_k)

    Add node (C_i,t_k) to C_i's list

    If C_j's list not empty

        Add edge from last element in list (C_j,tj) to (C_j,t_k)

    Add node (C_j,t_k) to C_j's list
```

Start at C_a's list. Go through the list until you find the last node (C_a, x') for $x' \leq x$. Now run BFS with (C_a, x') as the root node. If a node (C_b, y') for $y' \leq y$ is part of the traversal, then we say that C_b could be infected by time y.

```
Go through C_a's linked list and find last node (C_a,x') for x'≤x
```

```
Run BFS algorithm starting at (C_a,x')
    If (C_b,y') for y'≤y is visited
        C_b can be infected by time y
```

To analyze the output of the algorithm, let's split into the two possibilities that C_a was infected and the case that C_b was not infected.

If the virus could have been introduced to C_a at time x and could have infected C_b at time y, then we expect our algorithm to indicate that. For this to happen, C_a and C_b have to be connected after time x. If C_a and C_b are connected, then there must be a path from C_a to C_b since the algorithm builds the graph as so. The connection must also happen after the time x of infection, which the algorithm enforces by starting at nodes of C_a after time x.

Now consider the situation where a virus introduced to C_a at time x cannot infect C_b at time y. This can happen with C_a and $C_b$being connected but before time x. In this case, our algorithm would give the correct output since it starts looking through C_a's list only for $x' \leq x$. If C_a and C_b are not connected in the first place, then there will be no path from C_a to C_b in the graph list, so our algorithm would give the correct output.

Consider the runtime of the algorithm. There are m triples, so it will take $O(m)$ to steps to traverse all of them. Looking through each triple runs in constant time, so we can build the graph in $O(m)$. Running DFS is linear with the size of the graph, so that takes $O(m)$, giving a total runtime of $O(m)$.

5. Exercise 12 on page 112

We construct a directed graph using the relations. For each P_i add nodes birth and death, b_i and d_i. Since every person was born and died at some point, we add the edge (b_i, d_i). For all relations where we know P_i died before P_j, we can construct a directed edge from (d_i, b_j). For relations where we know the P_i overlapped with P_j, we add edges (b_i, d_j) and (b_j, d_i). The algorithm looks like the following:

```
Loop through all the relations
    For P_i and P_j in the relations
        Add 2 nodes: b_i and d_i
        Add dir edge from b_i to d_i
    If the relation is P_i died before P_j
        Add dir edge from d_i to b_j
    If the relation is P_i overlapped with P_j
        Add dir edge from b_i to d_j
        Add dir edge from b_j to d_i
```

Now we run a topological sort algorithm on the directed graph. If the DG is a DAG, then we call the relations consistent, and the topological sort is our ordering of the people's lives. If the DG has a

cycle, then the topological sort algorithm will fail, and our inputs were inconsistent. The algorithm for the topological sort looks like this:

```
Find a node v with no incoming edges and order it first

    Delete v from G

    Recursively compute a topological ordering of G–{v} and append this order
after v

If there are no nodes with no incoming edges

    G has a cycle and therefore finding are inconsistent
```

6. An array of n elements contains all but one of the integers from 1 to n+1.

(a) Give the best algorithm you can for determining which number is missing if the array is sorted, and analyze its asymptotic worst-case running time.

```
Create two pointers: one for start and one for end

While the end is greater than the start

    Middle is (start+end)/2

    If the difference between the middle element and the start index is 1

        Set end to middle

    If the difference between the middle element and the start index is 2

        Set start to middle

Return one more than the middle element
```

This algorithm will run in $O(logn)$ time. This is because for each iteration of the while loop, we are halving the domain. In the worst case, the missing number is at the start or the end, which would require $logn$ comparisons.

(b) Give the best algorithm you can for determining which number is missing if the array is not sorted, and analyze its asymptotic worst-case running time.

```
Calculate the total sum of all numbers from 1 to n+1 which is (n+1)*(n+2)/2

For all n numbers in the array

    Subtract the number from sum

Return sum
```

This algorithm will run in $O(n)$ time. This is because we need to consider all the elements in the array at least once. This algorithm does not have a worst-case and will take n steps every time.