CS 180, Fall 2019
Problem Set 3
Due October 23, 2019

Tian Ye
UID: 704931660

October 23, 2019

Exercise 10 Page 110

We will use a BFS to search for the number of shortest paths from v to w; we know that all the shortest paths will share the same level. Hence, we will identify that level and increment a counter each time we encounter w from that level.

```
let count = 0
let v.level = 0
let layer_depth = INT_MAX

Queue Q
Q.enqueue(v)
discovered[v] = true

while Q is not NULL
        let node = Q.dequeue()
        for each neighbor node' of node
                if node' == w and node.level < layer_depth
                        layer_depth = node.level
                        count++
                else if discovered[node'] == false
                        Q.enqueue[node']
                        node'.level = node.level + 1
                discovered[node'] = true

return count
```

Proof. The algorithm runs in $O(m+n)$ time complexity as it is essentially a BFS with the only difference being that it increments a counter every time it encounters w from the correct layer. □

Exercise 6 Page 108

Proof. Assume there exists some edge e that exists in graph G but does not exist in graph T, as well as two arbitrary nodes u and v.

If e makes no difference in the distance from node w to u and from w to v, the BFS will not add e, since it already has an edge from w to the parents of both u and v, meaning it will not be added to T. For the DFS, if e connects two nodes that were already previously visited, it does not add e to T. Otherwise, it will be added to T. This is contradictory, as the in the one of the cases, the final tree T will differ for the BFS and DFS.

On the other hand, if e makes the distance from w to u or from w to v shorter, the BFS will include e as it finds the shortest path. For the DFS, if e connects two previously visited nodes, it will not be included. Otherwise, it will include it to T. Once again, the final tree T from the BFS and DFS do not necessarily match.

Therefore, by contradiction, the statement assuming there is an edge e not in T must be false. $\square$

Exercise 7 Page 108

Proof. Assume for the purposes of a proof by contradiction that there exist two nodes u and v within graph G that are not connected with each other. This indicates that there are no shared nieghbors between either node, and hence they and their neighbors can be broken down into subgraphs of U and V.

Let us say that U contains n_u nodes and V contains n_v nodes. From this, we can say that the maximum number of edges U can contain is $\frac{(n_u-1)(n_u-2)}{2}$. Similarly, the maximum number of edges V can contain is $\frac{(n_v-1)(n_v-2)}{2}$. Since $n_u + n_v = n$, we can combine the maximum number of edges of $U + V$ and simplify to see that G has at most $\frac{n^2-2n_u(n-n_u)-3n-4}{2}$ edges.

This contradicts the claim, since if every vertex within G has a degree of at least $\frac{n}{2}$, the number of edges within G must be at least $\frac{n^2}{2}$. Since $1 < u < n$, $\frac{n^2-2n_u(n-n_u)-3n-4}{2} < \frac{n^2}{2}$.

Therefore, if every vertex has a degree of at least $\frac{n}{2}$, G must be connected. $\square$

Exercise 3 Page 189

Proof. If we were to not use the given greedy algorithm, and choose to not always pack our trucks to the maximum capacity, there exist two possible scenarios:

- The truck that arrives can carry our extra box. Adding the box to the next truck will result in a solution that is identical to the current greedy algorithm in terms of efficiency.
- The truck that arrives can not carry our extra box. In order to carry our box, they will have to unload a box and then use an additional truck past the first one. This is worse in terms of efficiency than the given greedy solution.

Hence, the greedy algorithm is the most efficient way to approach this problem. □

Exercise 6 Page 131

Proof. For the optimal solution, we want to combine each individual's running and biking time, and organize them from slowest to fastest combined time. We will have the contestants use the pool one at a time in that order.

We will show that this is the fastest solution using the following:

Every contestant has to swim and then run and bike afterwards. As the time spent in the pool for all the contestants is a constant, our goal is to minimize the remaining run and bike time. To do so, we will place the contestant with the shortest run and bike time last to minimize the time spent in the competition. We then do this for all the remaining contestants, arriving at the initial proposed solution. □

Exercise 6a

Proof. We will use a DFS starting from every single node n, resetting the adjacency list with every iteration. This ensures that we are able to backtrack with every single new iteration starting from a new node n. We will keep a global variable that contains the current maximum path; every time a new longer path is found, we update that global variable. After the DFS is performed on every node, we return the longest path.

This algorithm runs in $O(n!)$, as when it performs the modified DFS, it will go through every single path, not vertex. This means that each call of the DFS will iterate through up to n nodes, then $n-1$, then $n-2$, etc. Hence the time complexity is $O(n!)$. □

Exercise 6b

We will start by performing a topological ordering on the graph. For each topological ordering, we will start at the root and go to the adjacent nodes. We will update the adjacent node's path if the current node's path plus the adjacent node's path is greater than what the adjacent node's path already is. We then traverse backwards from the longest node path to find the longest path. At the end, we will return the list of the nodes with the longest path.

```
Initialize the graph G with edge counter to 0 for each node
For each node n in G
        Set n's distance to 0
        For each outgoing edge from n to another node v
                Increment v's incoming edge counter by 1
For each node n in G
        If the incoming edge is 0
                Add n into list L
While L is not empty
        Get node n from L and delete n from L
        For each outgoing edge e from node n to v
                Decrement v's incoming edge counter by 1
                If v's distance == n's distance
                        Increment v's distance by 1
                        Set v's previous node to n
                If v's incoming edge counter is 0, add v to L
Search all nodes for node n' with the greatest distance
Create a list starting with node n'
        Get the previous node of n'
        Append the previous node of n' to the list
        Repeat using the previous node of n'
Return the final list
```

Proof. This solution holds for all, as even if there is a disconnected vertex, that can serve as just another component. The time complexity is $O(n+m)$ for the topological ordering, and $O(n)$ for the linear scan, resulting in an overall time complexity of $O(n+m)$. □