Homework #8

Megan Ku
Data Structures and Algorithms

April 2, 2020

1. Single Minimum Spanning Tree Proof

We will prove the single minimum spanning tree by contradiction. We start by saying that all the edges have unique weights. Let's suppose that there are two optimal solutions: T_1* and T_2*. That means that there is an edge in the original graph that when added and another edge is removed, the total cost of the tree does not increase. This means that there are two edges in the graph with the same weight. But this contradicts our intial statement that all the weights are unique. QED

2. Breadth First Search

a. Proof by Contradiction

Suppose that there is a node v whose BFS path to the source i is not the shortest path. This means that there exists a path from i to v that traverse less edges than the returned BFS path. From node i, the kth iteration of BFS discovers the path to all nodes $\leq$ k edges away. If the shortest path was less than k edges away from i and the path that BFS returned was k edges away, then the BFS didn't discover the path during the first k-1 iterations. So we have a contradiction where BFS doesn't fully search all the paths it should be at each iteration. QED

b. Make a Weighted Graph Unweighted

To make a weighted graph unweighted, we can replace each edge with weight w with a string of edges and vertices that connect the original vertices together. The number of edges would be equal to w+1, and w vertices link those edges together. Then, in the new graph, an edge of zero weight would be replaced with a single unweighted edge, an edge with a weight of 1 would be replaced with two edges joined by an intermediate vertex, etc. These inserted vertices will have no value associated with them. This preserves the relative distances between the vertices as they were in the weighted graph, and since those relative distances are preserved, the shortest path remains the same. So the new graph would have the original $|V|$ vertices and $|E|$ edges, plus the sum of the following: for each weighted edge with weight w, add w edges and w vertices in between the neighbors that edge joins together. So if the sum of the weights is W, the total number of vertices in our unweighted graph is $|V| + W$, and the number of edges is $|E| + W$.

c. Shortest BFS Path for Weighted Graph

The construction of the graph takes $O(h|E|)$, where h is the maximum edge weight in the graph. The BFS of the new graph takes $O(h(|V| + |E|))$ time, so the total runtime is $O(h(|V| + 2|E|)$ which can be represented as $O(h(|V| + |E|))$. In comparison, Dijkstra's algorithm takes, with a Fibonacci heap, $O(|E| + |V|log(|V|))$. By using an unweighted graph to represent a weighted graph, the runtime becomes dependent on the most weighted edge in the original graph. So, generally, Dijkstra's algorithm would be faster because it isn't bound by the weight of the graphs. If there were only 3 edges but the heaviest edge had a value of 10,000,000, the BFS method would take significantly longer than Dijkstra's algorithm. Likewise, if all the edges already have a weight of 1, then the BFS method would run faster than Dijkstra's algorithm.