Homework 3: CSCI 6212 Algorithms, due: Beginning of class, October 18, 2019

Collaboration Policies: This is not a group homework, you are required to do this homework yourself. However, you are *allowed* to use or search any passive online resource for information about how to solve the problem, and discuss the problems with classmates. You are *not allowed* to use active web resources (like Stack Overflow) where you post questions and ask for responses, post these questions to a "work for hire" site where someone else does them for you, or take *any* written notes from discussions with classmates or others. Problems:

1. Let G(V,E) be undirected, weighted graph, and let T be a minimum spanning tree of G. Suppose you are given *both* G and T, and also another edge e not already in the graph. Let G' be the graph G + the new edge. What can we learn about T', a minumum spanning tree for G'?

   (a) True or False: the total weight of T is less than or equal to total weight of T'. Explain why in 1 or 2 sentences? *[2 point T,F, 3 for explaination] False. The weight of T' can be smaller than the weight of T.*

   (b) Give an $O(n)$ algorithm to find T'. Explain your algorithm in Pseudocode, and give an argument why it is correct. *[10 points: (5 algorithm (fewer points if not O(n), or if it is unclear). 5 analysis – 3 points for argument about cycle, 2 points for argument that you only have to worry about the cycle.)]*

   Algorithm: Add new edge e to T and find the one cycle in the graph with edges $T\{e\}$. If e is heavier than all edges on the cycle, then there is no change, otherwise add e to T and drop the heaviest edge on the cycle to get the new tree T'.

   Runtime: Can find this cycle using DFS of tree T, starting at u and stopping when node v is discovered. Because T only has $n-1$ edges, this takes $O(n)$ time. Finding the heaviest edge on the cycle also takes $O(n)$ time.
   Argument for Correctness:

   Part 1: We add edge e and create a cycle. Removing the heaviest edge from this cycle gives our tree T'.

   Let's consider how Kruskal's algorithm runs on G (the original graph) and G' (the graph that includes the new edges e).

   For all edges with weight less than $w(e)$ the algorithm runs the same because it is considering the same edges with the same weights in the same order.

   If edge e is not added by Kruskal's algorithm that must be because it's endpoints are already connected, and adding it makes a cycle, and since all edges are added in order of weight, then e would be the heaviest edge in the cycle (because it was added last).

   If edge e is added by Kruskal's algorithm then there are two components that are linked in the MST of G' that are not linked in the MST of G (at the same point in the algorithm). The edge that links those components in G comes after e and that ed *linking two components that were not linked by the MST algorithm that ran on graph G, so there will be some other edge that would have linked those components that will now not be added because they*

that must be because it's endpoints are already connected, and adding it makes a cycle, and since all edges are added in order of weight, then e would be the heaviest edge in the cycle (because it was added last).

edge (u,v) connected

The nodes are the cycle were connect in T, and we re-connect with less weight in T' using the new edge.

2. Let $G = (V, E)$ be a weighted graph and suppose that you have already used Dijkstra's Algorithm to compute the shortest paths to all nodes from a source s. Suppose every edge in the graph has its weight increase by 2. Do the shortest paths all remain the same? If so, argue why, if not, show a (small) example graph where the paths change. *2 points for "False", 2 points for "correct example", 1 point for "easy to interpret" correct example.*

Good Practice Problems (BUT NOT GRADED)

- Give an example of a directed graph with negative-weight edges for which Dijkstras algorithm produces incorrect answers. Show why Dijkstras algorithm fails.
- Edge-disjoint paths Given a directed graph $G = (V, E)$ with vertices $s, t \in V$, give an algorithm that finds the maximum number of edge-disjoint paths from s to t.
- 4) Alice wants to send Bob a file over a wired network. Their computers are connected to each other by multiple routers and multiple different routes. Given the bandwidth of each connection (edge weights), Alice needs to calculate the route that can send the file the fastest. How do you modify Dijkstras algorithm to find the path with the highest (minimum) bandwidth?