CSCI 570 - Summer 2015 - HW 3

1. Solve Kleinberg and Tardos, Chapter 4, Exercise 8.

   Assume that T and $\hat{T}$ are two distinct minimum spanning trees for the the graph. This implies that $\exists e \subset T$ such that $e \notin \hat{T}$. Adding e to $\hat{T}$ results in a cycle C. Since the edge weights are distinct, there is a unique maximum weight edge e_{max} in C. By the cycle property (Prop. 4.20 in text) no minimum spanning tree can contain e_{max}. This contradicts the fact that e_{max} is either in T or in $\hat{T}$.

   NOTE : Problems 2 and 3 can be trivially solved by ignoring T and computing an MST for G from scratch. The objective of these exercises is therefore to come up with algorithms that run faster than computing from scratch. For problem 3, you can assume that d is a constant independent of the number of vertices in G.

2. Assume that you are given a graph G and a minimum spanning tree T of G. A new edge e is added to G (without introducing any vertices) to create a new graph $\bar{G}$. Design an algorithm that given G, T and e, finds a minimum spanning for $\bar{G}$.

   We add the edge e to the minimum spanning tree T and this creates a (unique simple) cycle C. Let e_{max} be an edge of maximum weight in the cycle C. We claim that $\bar{T} := T \cup \{e\} \setminus \{e_{max}\}$ is a minimum spanning tree for the new graph $\bar{G}$.

   Proof of claim:

   Since $\bar{T}$ is a tree and has $|V| - 1$ edges, it clearly spans $\bar{G}$.

   If $w(e) = w(e_{max})$, then by the cycle property (page 147, 4.20 in text) T is an MST of G and since in this case $\bar{T}$ has the same weight at T, $\bar{T}$ is an MST of $\bar{G}$ as well.

   We now deal with the case $w(e) < w(e_{max})$. Since $w(\bar{T}) = w(T) + w(e) - w(e_{\max})$, we see that $w(\bar{T}) < w(T)$. Assume that $\bar{T}$ is not an MST of $\bar{G}$. This implies that there exists $\bar{T}_{opt}$ such that $w(\bar{T}_{opt}) < w(\bar{T})$.

Observe that it is necessary that e is contained in T_{opt}, otherwise T_{opt} would be a spanning tree of G that is of weight lesser than the MST T.

If we remove e from T_{opt}, then we get two disjoint connected components (call them A and B). Since T is a spanning tree, there exists a unique edge e_1 in T that connects A and B. Further, by construction, e_1 is in the cycle C. Construct the spanning tree $T_2 := T_{opt} \cup \{e_1\} \setminus \{e\}$ which is actually contained in G.

$$\begin{aligned} w(T_{opt} \cup \{e_1\} \setminus \{e\}) &= w(T_{opt}) + w(e_1) - w(e) \\ < w(T + w(e_1) - w(e)) &= w(T) + w(e) - w(e_{max}) + w(e_1) - w(e) \\ &= w(T) + w(e_1) - w(e_{max}) \end{aligned}$$

Since $w(e_{max}) \leq w(e_1)$, $w(T_2) < w(T)$ and this contradicts the fact that T is an MST of G. Hence our assumption is wrong and T is indeed an MST of G.

Running Time: All we need to compute is the edge e_{max}. Construct the unique path (call P) in T that connects the two ends of e. This can be accomplished using BFS in $\mathcal{O}(n)$ time. The cycle C is P concatenated with e and we compute e_{max} as a maximum weighted edge in C.

3. Assume that you are given a graph $G = (V, E)$ and a minimum spanning tree T of G. A new edge v is added to G (along with d edges that connect v to G) to create a new graph G. Design an algorithm that given G, T,v and the set of edges connecting v to G, finds a minimum spanning for G.

   Pick an edge f that crosses the cut $(V, \{u\})$. Clearly $T \cup \{f\}$ is a minimum spanning tree of the graph $\hat{G} := (V \cup \{u\}, E \cup \{f\})$. Observe that G can be obtained from $\hat{G}$ by a sequence of edge additions. But from the solution to problem 2, we know how to update the minimum spanning tree when an edge is added ! Hence we have an algorithm to compute a minimum spanning tree for G by a sequence of updates. The degree of u is d and the cycle C that appears in each update step could be big (as big as $\mathcal{O}(|V|)$). Thus the total running time is $\mathcal{O}(d|V|)$.

   Fun Problem: There is an algorithm that updates the MST in $\mathcal{O}(|V|log|V|)$ time even if d is as big as $O(|V|)$. Can you think of such an algorithm ?

4. You are given a weighted directed graph $G = (V, E, w)$ and the shortest path distances $\delta(s, u)$ from a source vertex s to every other vertex in G. However, you are not given $\pi(u)$ (the predecessor pointers). With this information, give an algorithm to find a shortest path from s to a given vertex t in $O(|V| + |E|)$ time.

Solution 1:

(a) Starting from node t, go through the incoming edges to t one at a time to check if the condition $\delta(s,t) = w(u,t) + \delta(s,u)$ is satisfied for some $(u,t) \subset E$.

(b) There must be at least one node u satisfying the above condition, and this node u must be a predecessor of node t.

(c) Repeat this check recursively, i.e. repeat the first step assuming node u was the destination instead of node t to get the predecessor to node u, until node s is reached.
Complexity Analysis: Since each directed edge is checked at most once, the complexity is $O(|V| + |E|)$. Note that constructing the adjacent list of G^{rev} in order to facilitate access to the incoming edges is needed, which is still bounded by $O(|V| + |E|)$ complexity.

5. Given a directed graph with positive edge lengths and a source node s and a sink node t, design an algorithm to compute the number of shortest paths from s to t.

We modify Dijkstra's algorithm (page 138) to accomplish this. For a vertex u, let $Count(u)$ denote the number of shortest paths from s to u. Initialize $Count(s) = 1$ and for all $u \neq s$, $Count(u) = 0$.

When an edge (u,v) with $u \subset S$, $v \notin S$ is relaxed, there are three cases to consider for updating $Count$.

If $d(v) < d(u)$, do nothing.

If $d(v) > d(u) + \ell(u,v)$, then update $Count[v] = Count[u]$ since we have found a potential shortest path to v through u and u could be reached in $Count[u]$ ways.

If $d(v) = d(u) + \ell(u,v)$, then set $Count(v) = Count(v) + 1$ since we have found another way to get to v with the same length.

6. Consider the following modification to Dijkstras algorithm for single source shortest paths to make it applicable to directed graphs with negative edge lengths. If the minimum edge length in the graph is $w < 0$, then add $w + 1$ to each edge length thereby making all the edge lengths positive. Now apply Dijkstras algorithm starting from the source s and output the shortest paths to every other vertex. Does this modification work? Either prove that it correctly finds the shortest path starting from s to every