Homework #6
Introduction to Algorithms
601.433/633
Spring 2020

Due on: Tuesday, April 21st, 12pm

1 Problem 1 (15 points)

Let $G = (V, E)$ be an undirected graph with *distinct non-negative* edge weights. Consider a problem similar to the single-source shortest paths problem, but where we define path cost differently. We will define the cost of a simple s-t path $P_{s,t} = \{e_1, e_2, \ldots, e_k\}$ to be

$$c(P_{s,t}) = \max_{e \in P} w_e$$

The cost of a path is now just the largest weight on that path, rather than the sum of the weights on the path. Give an algorithm that takes as input an undirected graph $G = (V, E)$ with *non-negative edge weights*, and a vertex $s \in V$, and computes the path from s to every other node in G with the least cost under cost function $c(\cdot)$. That is, for each $t \in V$, find a simple path $P_{s,t} = \{(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, t)\}$ connecting s to t, such that, letting S denote the set of all simple paths connecting s and t, $c(P_{s,t}) = \min_{P \in S} c(P)$.

Your algorithm should run in $O(|E| + |V| \log |V|)$ time. Prove the correctness of your algorithm and its runtime. Hint: note the similarities between Dijkstra's algorithm and Prim's algorithm.

2 Problem 2 (15 points)

You are given a simple weighted graph (no self-edges or multi-edges) $G = (V, E)$ and its minimum spanning tree T_{mst}. Somebody added a new edge e to the graph G. Let's call new graph $G' = (V, E \cup \{e\})$. Devise an algorithm which checks if T_{mst} is also a minimum spanning tree for the graph G' or not. Your algorithm should work in $O(|V|)$ time. Prove correctness of your algorithm and provide running time analysis.

You can assume that all edge weights in G' are distinct. T_{mst} and G' are given to you as adjacency lists.

3 Problem 3 (20 points)

An airline has n flights. In order to avoid "re-accommodation", a passenger must satisfy several requirements. Each requirement is of the form "you must take at least k_i flights from set F_i". The problem is to determine whether or not a given passenger will experience "re-accommodation". The hard part is that any given flight cannot be used towards satisfying multiple requirements. For example, if one requirement states that you must take at least two flights from $\{A, B, C\}$, and a second requirement states that you must take at least two flights from $\{C, D, E\}$, then a passenger who had taken just $\{B, C, D\}$ would not yet be able to avoid "re-accommodation".

Given a list of requirements $r_1, r_2, \ldots, r_m$ (where each requirement r_i is of the form: "you must take at least k_i flights from set F_i"), and given a list L of flights taken by some passenger, determine if that passenger will experience "re-accommodation". Your algorithm should run in $O(|L|^2 m)$ time.

Hint: You should just need to show how this can be reduced to a network flow problem and then use a blackbox algorithm solving the flow problem. Prove that your reduction is correct and that the runtime of the algorithm you use will be $O(|L|^2 m)$ for the reduction you give.