HW 5 (due Monday, at noon, March 9, 2015)

OLD CS 473: Fundamental Algorithms, Spring 2015 Version: **1.01**

You also have to do the quiz online (on moodle).

Collaboration Policy: For this homework, Problems 1–3 can be worked in groups of up to three students.

1. (40 PTS.) Heaviest edge in lightest tree.

You are given an undirected connected graph G with n vertices and m edges. We have weights on the edges of G (assume, they are all unique). Now, consider the MST T of G, and let $\omega = w(\mathsf{G})$ be the weight on the heaviest edge in T. We refer to w as the ***heavy*** edge in G. Our purpose here is to compute w in linear time.

(A) (5 PTS.) Given a number α, describe a linear time algorithm (i.e., $O(n + m)$) that decides if $\omega \leq \alpha$ or $\omega > \alpha$ (without knowing ω, naturally).

(B) (5 PTS.) Assume that you are given α and $\alpha \leq \omega$. A path in G such that all the edges are of weight at most α is ***α-light***. In particular, consider the equivalence relationship $u \approx v$, which holds if u and v are connected by a α-light path. Provide an algorithm that computes the partition of $\mathsf{V}(\mathsf{G})$ into the equivalence classes of $\approx$.

(C) (5 PTS.) Given α, as above let G/α be the graph defined over these equivalence classes (from (B)), where the edge between two new vertices $X \subseteq \mathsf{V}(\mathsf{G})$ and $Y \subseteq \mathsf{V}(\mathsf{G})$ exists if there is an edge in G between any vertex of X and any vertex of Y, and furthermore, the weight of the edge XY is the minimum weight of such an edge in G.
Describe in detail (and be careful - this is not quite as easy as it looks) an algorithm for computing G/α in $O(n + m)$ time (you are allowed to use hashing for this part).

(D) (5 PTS.) Prove that if $\alpha < \omega$, then $w(\mathsf{G}/\alpha) = w(\mathsf{G})$.

(E) (5 PTS.) Assume that $\alpha > \omega$, and consider the graph $\mathsf{G}_{<\alpha}$, which is formed by removing from G all the edges in G with weight $\geq \alpha$. Prove that $w(\mathsf{G}_{<\alpha}) = w(\mathsf{G})$.

(F) (5 PTS.) Consider the algorithm that picks a certain the edge e in G, uses the algorithm (A) to decide if $\omega \leq \alpha$, or $\omega > \alpha$, where $\alpha = w(\mathsf{e})$ is the weight of the edge e. The algorithm then computes either the graph G/α or $\mathsf{G}_{<\alpha}$, and continue computing the $w(\cdot)$ recursively on the appropriate graph. (Once the input graph has size $O(1)$, it computes $w(\cdot)$ on the input graph by computing the MST, and outputting the heaviest edge.)
Describe this algorithm in more detail (pseudo-code might be a good idea here), and show that its running time is $O((n + m) \log n)$.

(G) (5 PTS.) Show how to modify the algorithm from (F), if necessary, such that the running time of the resulting algorithm for computing $w(\mathsf{G})$ is $O(n+m)$. Prove the new bound on the running time.

2. (30 PTS.) Extracting spanning trees.

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be an undirected graph with weights on the edges. The graph G has n vertices and m edges. Consider the situation where you want to compute for G potentially many spanning trees. The reason being that if the first spanning tree fails (say, the graph corresponds to a communication network, and a connection, which corresponds to an edge, might fail). To this end, the natural thing is to compute first the minimum spanning tree T_1 of $\mathsf{G}_1 = \mathsf{G}$. Next, let $\mathsf{E}_2 = \mathsf{E}(\mathsf{G}_1) \setminus \mathsf{E}(T_1)$. Let T_2 be the MST of the graph $\mathsf{G}_2 = (\mathsf{V}, \mathsf{E}_2)$.

More generally, in the ith iteration, the algorithm computes the minimum spanning tree T_i of the graph $\mathsf{G}_i = (\mathsf{V}, \mathsf{E}_i)$, where $\mathsf{E}_i = \mathsf{E}_{i-1} \setminus \mathsf{E}(T_{i-1})$. Note, that T_i might be a minimum spanning forest for G_i, if the graph G_i is not connected. In such a case, the algorithm computes only the minimum spanning trees of G_i that have more than one vertex. If T_k is empty, the algorithm can terminate at this iteration.

Describe how to compute the sequence of edge disjoint trees $T_1, T_2, \ldots$. An algorithm with running time $O(nm)$ would give you at most 15 points. For full credit, the running time of your algorithm has to be at

most $O(m \log^{O(1)} n)$.
Hint: Think about how to run all these MST algorithms in "parallel". In particular, consider an edge that appears in T_k, and think why it does not appear in $T_1, \ldots, T_{k-1}$.

3. (30 PTS.) Pipes on a tree

You are given an un-rooted tree T with n vertices (nodes in this tree, say, have degree at most 3). You are also given a set with m paths $\Pi = \{\pi_1, \ldots, \pi_m\}$. Here a path is defined by its two unique endpoints, since the path in tree between any pair of vertices is unique. Provide an algorithm that computes the largest set $X \subseteq \Pi$, such that no pair of paths in X share a vertex.
For full credit, your algorithm should be as fast as possible.