HW 6 (due Wednesday, at noon, October 17, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.3**

Submission guidelines and policies as in homework 1.

1 (100 PTS.) Steiner tree.

Consider an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with positive weights on the edges. Here G has n vertices and m edges. We are given a set $X \subseteq \mathsf{V}$, and the task is to compute the lightest spanning tree that contains all the vertices of X. This problem is known as the ***Steiner tree*** problem, and it is NP-HARD.

Consider the following randomized algorithm. It first computes a random permutation $x_1, \ldots, x_t$ of the vertices of X. Let $\mathcal{T}_i$ be a spanning tree of $\{x_1, \ldots, x_i\}$ computed by the algorithm. Initially, $\mathcal{T}_1$ is just the vertex x_1. In the ith iteration, it connects x_i to the closest vertex of $\mathcal{T}_{i-1}$ using a shortest path ρ_i. It adds ρ_i to $\mathcal{T}_{i-1}$ to get $\mathcal{T}_i$. the algorithm then outputs $\mathcal{T}_t$ as the approximation.

1.A. (10 PTS.) Show an example where this algorithm would not output the optimal solution, independent of the permutation.

1.B. (20 PTS.) Let C be the shortest cycle that visits all the vertices of X (the cycle C is not necessarily simple). Let opt be the optimal Steiner tree of X in G. Prove that $\omega(\text{opt}) \leq \omega(C) \leq 2\omega(\text{opt})$.

1.C. (20 PTS.) Let R_i be the length of ρ_i. Prove that $\mathbb{E}[R_i] \leq 2\omega(C)/i$.

1.D. (40 PTS.) Prove that $\mathbb{E}[\omega(\mathcal{T}_t)] = O(\omega(C) \log t)$.

1.E. (10 PTS.) Prove that the above algorithm provides a $O(\log n)$ approximation to the Steiner tree problem, in expectation. (This is easier than easy.)

2 (100 PTS.) Back and forth.

2.A. (10 PTS.) Prove that for any $x_1, \ldots, x_n \geq 0$, we have $\sum_{i=1}^{n} \frac{1}{x_i+1} \geq \frac{n}{\alpha+1}$, where $\alpha = \left(\sum_{i=1}^{n} x_i\right)/n$. (Hint: Follows readily from an inequality well known to the internet.)

2.B. (20 PTS.) Consider a graph G and a random permutation $\pi_1, \ldots, \pi_n$ of its vertices, and consider the greedy algorithm that, in the ith iteration, adds π_i to the computed independent set I if none of the neighbors of π_i were already in the independent set. Prove that for a vertex $v \in \mathsf{V}$, the probability of v to be in I is at least $1/(d(v)+1)$, where $d(v)$ denotes the degree of v in G.

Prove that $\mathbb{E}[|I|] \geq \frac{n}{\delta+1}$, where δ is the average degree of a vertex in G.

2.C. (20 PTS.) Prove that in a graph G with n, there are at most $n/2$ vertices with degree larger or equal to 2δ, where δ is the average degree of a vertex in G.

2.D. (20 PTS.) A ***partial coloring*** is a coloring of some of the vertices of the graph, such that every two vertices that are colored and have an edge between them have a different color (i.e., some vertices might not be colored). Present an efficient algorithm that partially colors, say, at least half the vertices in the graph. What is the number of colors your algorithm uses in the worst case (the lower, the better).

2.E. (30 PTS.) A ***weak k coloring*** is a coloring of the vertices of the graph, such that there are few edges that are assigned the same color. In particular, for a k coloring χ of a graph G, let $f(\chi)$ be the number of edges in G that are violated (i.e., they are assigned the same color). Present an algorithm that compute in randomized polynomial time a weak k coloring that violates at most m/k edges, where m is the number of edges of G. The result of the algorithm must be correct (but the running time can be a random variable). What is the running time of your algorithm? (Faster is better – also prove the bound on the running time of your algorithm.)

(Hint: Use randomization. Start by showing an algorithm that achieves the desired coloring in expectation, and continue from there.)

3 (100 PTS.) k closest servers.

Consider an undirected graph G with n vertices and m edges, with positive weights on the edges. In addition, you are given a set S of servers. For each vertex v in the graph, we are interested in computing the set $N(S, v, k)$ – which is the set of the k closest servers to v in S (assume all shortest path distances are unique).

3.A. (20 PTS.) Show how to modify Dijkstra algorithm, such that given a set Y of vertices, one can compute for all the vertices in G their closest vertex in Y. What is the running time of your algorithm? (Faster is better.)

3.B. (20 PTS.) Let R be a random sample from S, where every vertex is chosen with probability $1/k$. Let u be a vertex in $N(S, v, k)$. Prove that with probability at least $1/(ck)$, for some absolute constant c, we have that $u \in R$, and no other vertex of $N(S, v, k)$ is in R.

3.C. (20 PTS.) Let $R_1, \ldots, R_t$ be $t = O(k \log n)$ random samples generated as above. For $i = 1, \ldots, t,$, and for each vertex v in the graph, compute its closest neighbor $n_i(v) = \text{ClosestNeighbor}(v, R_i)$ – that is, the closest vertex in R_i to v. What is the running time of your algorithm.

3.D. (20 PTS.) Let $L(v) = \{n_1(v), \ldots, n_t(v)\}$. Prove, that with high probability, for all $v \in \mathsf{V}(\mathsf{G})$, we have $N(S, v, k) \subseteq L(v)$.

3.E. (20 PTS.) Present an algorithm, as fast as possible, using the above, that computes for each vertex of G its k closest servers in S. What is the running time of your algorithm? Your algorithm should succeed with high probability.

(There is a deterministic algorithm for this problem with better running time, but it is less elegant.)