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

CS 473: Algorithms, Fall 2018 Version: **1.31**

Submission guidelines and policies as in homework 1.

1 (100 PTS.) Steiner tree.

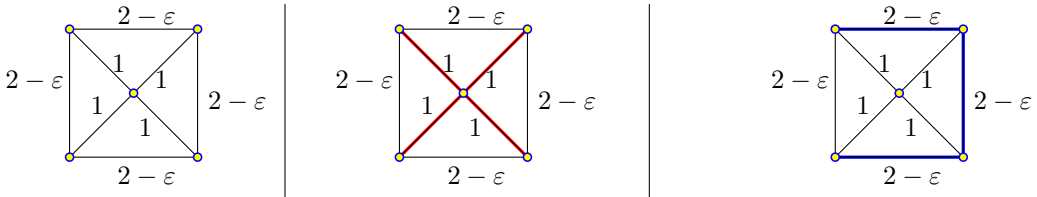
Consider an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with positive weights on the edges. Here G has n vertices and m edges. We are given a set $X \subseteq \mathsf{V}$, and the task is to compute the lightest spanning tree that contains all the vertices of X. This problem is known as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

Solution:

Consider the following graph, where the set X are the four corners.



Graph | Optimal solution of cost 4 | One possible solution of value $6 - 3\varepsilon$.

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 $w(\text{opt}) \leq w(C) \leq 2w(\text{opt})$.

Solution:

The cycle C spans X, which readily implies that $w(\text{opt}) \leq w(C)$, as opt is the cheapest spanning subgraph with this property.

As for the other part, observe that opt is a tree. As such, duplicating each of its edges, results in an Eulerian graph J. The Eulerian cycle of J has weight at most $2w(\text{opt})$, and it implies the claim.

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

Solution:

We use backward analysis. Let U be the first i vertices in the permutation π, which we fix. The introduction of U breaks C into i consecutive paths along C: $\tau_1, \ldots, \tau_i$ that their endpoints are at vertices of U, and no path has a point of U in its interior.

We now randomized over the possible ordering of the elements of U. Let $U = \{p_1, \ldots, p_i\}$. Clearly, p_t is the last point, then it can connect itself either to the point of U following it in C or to the previous one, which ever is cheaper. The path ρ_i might only be shorter than these two candidate paths. In particular, for each of these i points, select the path of C that is shorter and adjacent to it. Let $\sigma_1, \ldots, \sigma_i$ be these paths.

Clearly, $\sum_j w(\sigma_j) \leq 2w(C)$, since only two paths in the collection can correspond to a path τ_j, for some j. The claim now readily follows, since

$$\mathbb{E}[R_i \mid U] \leq \sum_j \frac{1}{i} w(\sigma_j) \leq \frac{2w(C)}{i}.$$

Now, we have

$$\mathbb{E}[R_i] = \mathbb{E}\Big[\mathbb{E}[R_i \mid U]\Big] \leq \mathbb{E}\left[\frac{2w(C)}{i}\right] \leq \frac{2w(C)}{i}.$$

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

Solution:

$$\mathbb{E}[w(\mathcal{T}_t)] = \mathbb{E}\left[\sum_i R_i\right] \leq \sum_{i=1}^{t} \frac{2w(C)}{i} = O(w(C) \log t).$$

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.)

Solution:

Well, the expected approximation quality is

$$\mathbb{E}\left[\frac{w(\mathcal{T}_t)}{w(\mathrm{opt})}\right] = O(\log t) = O(\log n),$$

by the above.

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.)

Solution:

The AM-GM-HM inequality, states that for any numbers $y_1, \ldots, y_n$, positive, we have

$$\frac{y_1 + \cdots + y_n}{n} \geq \sqrt[n]{y_1 \cdots y_n} \geq \frac{n}{\frac{1}{y_1} + \cdots + \frac{1}{y_n}}.$$

Which implies that

$$\frac{1}{y_1} + \cdots + \frac{1}{y_n} \geq \frac{n}{\frac{y_1 + \cdots + y_n}{n}},$$

which is what we need for $y_i = x_i + 1$.

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.

Solution:

Let $N(v)$ denote the $d(v)$ neighbors of v. If v is the first in the permutation out of $\{v\} \cup N(v)$, then it is definitely going to be in I. As such, we have that $\mathbb{P}[v \in I] \geq \frac{1}{|N(v)|+1}$, since this is the probability of v to be first in a random permutation of itself and its $d(v)$ neighbors.

The second part is now immediate. Define an indicator variable $X_v = 1 \iff v$ is in I. We have that

$$\mathbb{E}[|I|] = \sum_{v \in \mathsf{V}(\mathsf{G})} \mathbb{E}[X_v] \geq \sum_{v \in \mathsf{V}(\mathsf{G})} \frac{1}{d(v)+1} \geq \frac{n}{\delta + 1},$$

where the last part follows from (A).

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.

Solution:

If not, then we have

$$\sum_{v \in \mathsf{V}(\mathsf{G})} d(v) \geq \sum_{v \in \mathsf{V}(\mathsf{G}) : d(v) \geq 2\delta} d(v) \geq (n/2+1)2\delta > n\delta,$$

which is a contradiction, since $\delta = \sum_{v \in \mathsf{V}(\mathsf{G})} d(v)/n$.

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).

Solution:

The algorithm repeatedly pick a vertex of minimum degree that was not colored yet, and assign it to lowest color available out of $1, 2, \ldots$. The algorithm stops as soon as the algorithm tries to assign a vertex a color $2\delta + 2$. Observe, that by the above, all vertices of degree at most 2δ are colored by this algorithm, since there is always a color available that is valid for such vertices. By the above, this colors at least half the vertices in the graph.

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.)

Solution:

Assign every vertex a random color from $\{1, \ldots, k\}$. Observe that the probability of an edge to be rejected is $1/k$. As such, the expected number of rejected edges is **exactly** m/k. Let X be the number of rejected edges by the algorithm.

The algorithm next computes X. If $X > m/k$ then the algorithm tries again. Assume m/k is an integer. We have, by Markov's inequality, that

$$\mathbb{P}[X > m/k] = \mathbb{P}[X \geq m/k + 1] \leq \frac{\mathbb{E}[X]}{m/k+1} \leq \frac{m/k}{m/k+1} \leq \frac{m}{m+k}$$

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.)

Solution:

Add a super source vertex s, connect it to all the vertices of S by edges of cost zero, and then run Dijkstra from s. Clearly, this computes the shortest path from s to all the vertices in G, which is clearly the desired distance. It is now easy to modify Dijkstra such that it remembers, for each vertex v, the vertex $z \in S$ the shortest path arriving to v, passed through z. This does not effect the running time. All one need to do, is to remember together with the distance, an additional label to the gateway vertex. Now, whenever Dijkstra reassigns a distance to a vertex, it also copies the label from the source of the edge. In addition, all the vertices of S are initialized with their own label.

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.

Solution:

Let $N = N(S, v, k) = \{u_1, \ldots, u_k\}$. Assume that $u = u_i$. The probability that $u \in R$ is $1/k$, and the probability that any element $x \in N \setminus \{u\}$ is not in R is $1 - 1/k$. Observe that for two elements $x, y \in N$, the events of whether or not they are in R are independent. As such, the desired probability is

$$\alpha = \mathbb{P}\left[(u \in R) \cap \bigcap_{x \in N-u} (x \notin R)\right] = \mathbb{P}[u \in R] \dot{\prod}_{x \in N-u} \mathbb{P}[x \notin R] = \frac{1}{k}\left(1 - \frac{1}{k}\right)^{k-1} \geq \frac{1}{ek},$$

since $(1 - 1/k)^{k-1} \geq 1/e$ which is a well known fact. If you don't know it, observe that for $k > 1$, we have $1 - 1/k \geq \exp(-2/k)$ (see lemma below), and as such

$$(1 - 1/k)^{k-1} \geq (1 - 1/k)^k \geq \exp(-2/k \cdot k) = 1/e^2 \geq 1/8.$$

As such, $\alpha \geq \frac{1}{8k}$.

Lemma 0.1. *$1 - x \geq \exp(-2x)$, for $x \in (0, 1/2)$.*

Proof: Set $f(x) = (1-x) - \exp(-2x)$, and observe that $f(0) = 0$. Furthermore, we have

$$f'(x) = -1 + 2\exp(-2x) \qquad \text{and} \qquad f''(x) = -4\exp(-2x).$$

Solving for $f'(x) = 0$, we have $\exp(-2x) = 1/2 \iff -2x = \ln(1/2) \iff x = (\ln(2))/2 \approx 0.346$. Namely, $f(x)$ increases in the range $[0, 0.346...]$. After that point $f(\cdot)$ start decreasing. In particular, computing $f(1/2) \approx 0.132$, which implies that $f(x) \geq 0$ holds for all $x \in [0, 1/2]$. ■

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.

Solution:

The running time is $O((k \log n)(n \log n + m)) = O(kn \log^2 n + km \log n)$, since this is just running Dijkstra $O(k \log n)$ time. Specifically, we use the algorithm of part (A), on each of the $O(k \log n)$ random samples $R_1, \ldots, R_t$.

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 $N(S, v, k) \subseteq L(v)$.

Solution:

Consider an element $u \in N(S, v, k)$. By part (B), with probability at least $1/8k$, a sample R_i contains u and no other element of $N(S, v, k)$. But if this happens, then $u \in L(v)$. The probability of this NOT happening, over all samples, is at most $\beta = (1 - 1/8k)^{O(k \log n)} < 1/n^{10}$, by choosing the constant in the O to be sufficiently large. As such,

$$\begin{aligned}
&\mathbb{P}\Big[\forall v \in \mathsf{V}(\mathsf{G}), \forall u \in N(S, v, k) \qquad u \in L(v)\Big] \\
&\qquad = 1 - \mathbb{P}\Big[\exists v \in \mathsf{V}(\mathsf{G}), \exists u \in N(S, v, k) \qquad u \notin L(v)\Big] \\
&\qquad \geq 1 - nk\beta \\
&\qquad = 1 - 1/n^8.
\end{aligned}$$

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.

Solution:

Compute the lists as described above. Using hashing remove duplicate from each lists $L(v)$ so that it contains no repetitions. This takes $O(nk \log n)$ time. Next, for each list find the k smallest numbers (the numbers are the distances of these vertices to v) – this can be done in time proportional to the list size. Now just output the lists. The correctness is implied by the above. As for running time we have

$$O\big((k \log n)(n \log n + m) + nk \log n\big) = O\big(nk \log^2 n + mk \log n\big),$$

as desired.

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

Solution:

See the paper https://arxiv.org/abs/1607.07818.