Advanced Algorithms 2019 Lecturer: Mohsen Ghaffari

Graded Homework 2

Student: First Name Last Name, ID Number *Deadline: 12/03/2019*

Note 1: Solutions must be typeset in LaTeX and should be emailed to brandts@ethz.ch by 11:59 pm on December 3, 2019. Please make sure to use the subject "Advanced Algorithms 2019: GHW1" for your email, without the quotation marks.

Note 2: This problem set was compiled at this time: 12:02, Wednesday 27th November, 2019. In the unlikely case of a future update, we will mark the new version with a new timestamp and we will inform you via email about the update.

Note 3: For this graded homework, the late submission policy is that you lose 20% of the grade, on the whole graded homework, for each day of delay, and the number of days of delay is rounded up. Hence, for instance, 2 hours of delay means your grade will be multiplied by 0.8, and 25 hours of delay means your grade will be multiplied by 0.6.

1 Uber (20 points)

Every evening, after enjoying his research and teaching at ETH, Mohsen works as an Uber driver to make some extra cash and support his luxurious lifestyle. Your task is to help him optimize his route so that he can satisfy all requests in a small amount of time.

The road network of Zurich is given as a simple connected weighted graph G on n nodes. Each edge $e \in E(G)$ has a positive integer weight $w(e)$, meaning that traveling along this edge takes $w(e)$ seconds. Mohsen is starting at a vertex $u \in V(G)$. He can fit up to C other people in his car, besides himself (C is a positive integer). He got a series of k requests (all at the same time). Each request consists of two vertices $s_i, t_i \in V(G)$, meaning that a person currently in s_i wants to move to t_i.

Your task is to find a *route* for Mohsen. This is a sequence of vertices of G starting at u with consecutive vertices connected by an edge in $E(G)$. The same vertex can appear multiple times in the sequence. Moreover, for each vertex in this sequence you should tell Mohsen whether he should pick up some person from this vertex to his car or drop some person from his car to this vertex. Note that there can be up to C people (besides Mohsen) in the car at each point in time. Since Mohsen prefers saving his time over the comfort of his customers, as many times as he wishes he can drop a person at some vertex of G and later pick the person up from this vertex again. In the end, all people should be dropped at their respective destinations. The cost of the route is the sum of costs of the edges on the route (where, to be clear, traversing an edge multiple times incurs the edge cost multiple times). The objective of this approximation algorithm's problem is to minimize this cost.

(A) Find a deterministic polynomial-time $O(1)$-approximation algorithm for the problem of minimizing the cost of the route, for the special case that the graph G is a tree. Your algorithm's approximation factor should be an absolute constant, and independent of C (i.e., treat C as a non-constant).

(B) *For the solution of this problem, you can assume that you have a solution to the previous subproblem.*

Find a randomized polynomial-time algorithm with expected $O(\log n)$ approximation guarantee for general graphs G.

2 Number of Distinct Elements (20 points)

Consider a stream $a_1, a_2, \ldots, a_m$ such that $a_i \in \{1, 2, \ldots, n\}$. For each element $i \in \{1, 2, \ldots, n\}$, define the frequency of i, say f_i, as the number of times that i is appeared in the stream. We want to approximate the number of distinct elements of the stream, say ℓ, to within a $1 \pm \epsilon$ factor, for a given constant ϵ. For convenience, assume that the number of distinct elements ℓ is greater than a desirably large constant (that can depend on ϵ).

(A) Define a random set S_k by including each $x \in \{1, 2, \ldots, n\}$ in S_k with probability $1/k$ independently and let $y_k = \sum_{i \in S_k} f_i$. We want to show that $y_k = 0$ is a good statistical test for approximating ℓ. More precisely, prove that for large enough k, we have:

- If $\ell < (1 - \epsilon)k$, then $\Pr[y_k = 0] > \frac{1}{e} + \frac{\epsilon}{3}$,
- If $\ell > (1 + \epsilon)k$, then $\Pr[y_k = 0] < \frac{1}{e} - \frac{\epsilon}{3}$.

(B) Assume that you can do the test of part (A) using $g(n, \epsilon)$ bits of memory. Design an algorithm that computes a $(1 \pm \epsilon)$-approximation of ℓ with probability at least $1 - 1/n$ using space of $O(g(n, O(\epsilon)) \log^2 n)$ bits.

3 Streaming, Median Lower Bound (20 points)

Suppose a non-decreasing stream $a_1, a_2, \ldots, a_m$ where $a_i \in \{1, 2, \ldots, n\}$ and $a_i \leq a_{i+1}$ for $i \leq m - 1$. Prove that any deterministic algorithm which can find the median of a non-decreasing stream, e.g. the element $a_{\lceil \frac{m}{2} \rceil}$, needs space of $\Omega(n)$ bits. Note that the algorithm does not know m in advance.

4 Streaming, Higher Connectivity Algorithm (20 points)

Consider a graph in the dynamic streaming setting, i.e., a dynamic stream of edge arrivals and departures, on a set V of n vertices. Let $G = (V, E)$ denote the graph at the end of this stream. Devise a streaming algorithm that uses $nk\text{poly}(\log n)$ bits of space and, with high probability, computes a collection F_1, F_2, ..., F_k of forests such that for each $i \in \{1, 2, ..., k\}$, the forest F_i is a maximal forest in the graph $G \setminus (\cup_{j=1}^{i-1} F_j)$ — this is the spanning subgraph of G resulting from removing all edges in any of F_1 to F_{i-1}.

Note: If the stream was repeated k times, you could easily compute these forests by directly applying what we saw in lecture 9, each time ignoring the edges of the stream already present in the previously constructed forests. The question asks you to compute all these forests when there is no repetition of the stream.

5 Derandomized 2-Additive Spanner (20 points)

In lecture 10, we discussed a *randomized* algorithm that constructs a spanner with additive stretch of 2 and with $O(n^{3/2} \log n)$ edges, with probability $1 - 1/\text{poly}(n)$. In this exercise, we ask you to derandomize the process and get almost the same result deterministically. In particular, recall that the only randomized step in the algorithm was to mark vertices with probability $p = O(\log n)/\sqrt{n}$, which provided the guarantee that, with probability $1 - 1/\text{poly}(n)$, each heavy vertex — i.e., any vertex with degree at least $\sqrt{n}$ — has a marked neighbor.

Devise a *deterministic* algorithm that runs in $m\text{poly}(\log n)$ time, m being the number of the edges of the original graph, and marks a set of at most $O(\sqrt{n} \log^2 n)$ vertices such that, we have the deterministic guarantee that each heavy vertex has a marked neighbor.