HW 6

Due on **Wednesday, October 13, 2021 at 10am**

CS 473: Algorithms, Fall 2021 Version: **1.2**

Submission guidelines and policies as in homework 1.

16 (100 PTS.) Coloring for long path.

You are given an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges. Computing a *simple* path containing exactly k vertices is NP-HARD if k is large. Indeed, for $k = n$ this is just Hamiltonian Path, and furthermore, it is not obvious how to get a running time better than $O(n^k)$ for this problem. This exercise presents a beautiful technique that gets you running time (roughly) $O(c_k n^{O(1)})$, where c_k is some horrible constant that depends only on k.

16.A. (25 PTS.) You are performing an experiment that succeeds with probability $p > 0$. Let $\delta \in (0, 1)$ be a parameter. Provide an upper bound, as small as possible, on how many times you have to repeat the experiment till it succeeds (at least once) with probability $\geq 1 - \delta$, as a function of p and δ. (This process of repetition is known as *amplification*.)

16.B. (25 PTS.) Consider a path $v_1 \to v_2 \to \cdots \to v_k$ of k vertices in G, and consider a random coloring $\chi : \mathsf{V} \to [\![k]\!] = \{1, \ldots, k\}$, where the color of each vertex is picked independently, uniformly, and randomly from $[\![k]\!]$. What is *exactly* the probability that $\chi(v_i) = i$, for all i?

16.C. (25 PTS.) You are given a coloring χ, as above, of the vertices of G. Describe an algorithm, as fast as possible, that computes, a path $v_1 \to v_2 \to \cdots \to v_k$ such that $\chi(v_i) = i$, if such a path exists. What is the running time of your algorithm?

16.D. (25 PTS.) Using the above, describe an algorithm, as fast as possible, that computes a simple path of length k in G if such a path exists, with success probability $\geq 7/8$. What is the running time of your algorithm?

17 (100 PTS.) Computing the k closest servers.

You are given an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges, a set $S \subseteq \mathsf{V}$ of servers, and a parameter $k > 0$. There are weights $\omega(\cdot) \geq 0$ on the edges. The task at hand, is to compute for each vertex in $v \in \mathsf{V}$, the k closest vertices to it in S.

Let $d(u, v)$ denote the shortest path distance in G between vertices u and v.

17.A. (25 PTS.) Given a set $Y \subseteq S$ of vertices, describe how to compute for each vertex $v \in \mathsf{V}$, the closest vertex to it in Y in $O(n \log n + m)$ time overall (for simplicity, you can assume all the distances you are dealing with are unique). Specifically, for all vertices $v \in \mathsf{V}$, the algorithm should compute

$$d(v, Y) = \min_{y \in Y} d(v, y) \qquad \text{and} \qquad nn(v, Y) = \arg\min_{y \in Y} d(v, y).$$

17.B. (25 PTS.) Consider a random sample $R \subseteq S$, where each vertex is picked with probability $1/k$. Consider a vertex $v \in \mathsf{V}$, and let $\Xi(v)$ be the k closest vertices to v in S. Pick any vertex $u \in \Xi(v)$. Argue that with probability $\Omega(1/k)$, u is the only vertex of $\Xi(v)$ in R.

17.C. (50 PTS.) Consider the algorithm that, for $i = 1, \ldots, M$, picks a random sample $R_i \subseteq S$, as described above, and computes the distances from R_i to all the vertices of V. For every vertex v, the algorithm computes, using **17.B.**, the set

$$C(v) = \left\{ \left(nn(v, R_i), d(v, R_i) \right) \;\middle|\; i = 1, \ldots, M \right\}.$$

The algorithm then computes for each vertex $v \in \mathsf{V}$, the k (distinct) closest vertices to it in $C(v)$, and let $S(v)$ be this set of k servers.

Describe how to implement this algorithm efficiently (as fast as possible). What is the running time of your algorithm as a function of n, m, k, M?

Provide an upper bound, as small as possible (up to a constant), on the minimum value of M, such that the algorithm computes $S(v)$ correctly for all the vertices of V (i.e., $S(v)$ are indeed the k closest servers to it in S), with probability of success at least (say) $\geq 1 - 1/n^{10}$. Prove that this value of M indeed provides the desired probability of success.

What is the overall running time of your algorithm for this choice of M?

(This is not the fastest algorithm for this problem, but it is definitely the most elegant. And it is a nice example of randomization.)

18 (100 PTS.) Fast decision tree.

Consider a rooted tree of height h. Here, every leaf is at distance h from the root. The root, as well as any internal node, has 5 children. Each leaf has a boolean value associated with it. Each internal node returns the value returned by the at least three of its children. Thus, if the values of the five children are $0, 1, 0, 1, 0$, the node would return the value 0. The evaluation problem consists of determining the value of the root. The algorithm can read a value stored in a leaf in constant time. The natural algorithm evaluates the tree recursively, and takes $O(n)$ time, where $n = 5^h$. Surprisingly, if one uses randomization, one can do much better.

18.A. (50 PTS.) Show that for any deterministic algorithm, there is an instance (a set of boolean values for the leaves) that forces it to read all $n = 5^h$ leaves.

[Hint:

I. Consider an adversary argument, where you provide the algorithm with the minimal amount of information as it request bits from you. In particular, devise such an adversary algorithm.

II. Start with $h = 1$, then extend to $h = 2$, etc. This part is somewhat harder, and you might want to do the other part first.]

18.B. (50 PTS.) Consider the recursive randomized algorithm that evaluates the children of a node in random order (recursively). As soon as three computed values agree, it returns it as the value of this node.

Show the expected number of leaves read by the algorithm, when called on the root of three, on any instance, is at most $O(n^c)$, where $c < 1$ is some constant bounded away from 1. What is the value of c? Prove your bound on c. (The value of c is going to be pretty close to one. Don't worry about getting the best constant.)

(Hint: Consider the case $h = 1$, and bound the expected number of leafs being evaluated in this case. Then extend it to $h = 2$ [using the $h = 1$ case], etc.)