CS/ECE 374 A (Spring 2020)
Homework 9 (due Apr 9 Thursday at 10am)

Instructions: As in previous homeworks.

Problem 9.1: You are managing a small company with two employees Alice and Bob. You are given n requests (known ahead of time), where for each $i \in \{1, \ldots, n\}$, the i-th request requires sending one of your employees to location p_i at a specified time t_i; it takes one unit of time for Alice or Bob to handle the request. (Alice or Bob may arrive before t_i, but may only begin at time t_i and finish at time $t_i + 1$.) Let $\tau(p_i, p_j)$ denote the time needed to get from p_i to p_j. (Thus, if Alice was at location p_i and finished the i-th request at time $t_i + 1$, Alice can go to location p_j before time t_j iff $\tau(p_i, p_j) \leq t_j - t_i - 1$. Similarly for Bob.) Let v_i be the "value" (a positive number) of the i-th request. Initially, at time 0, Alice and Bob are at location p_0.

Describe an efficient algorithm to find a subset S of requests that can be handled by Alice and Bob, maximizing the total value of S. You may assume that $\tau(\cdot, \cdot)$ can be evaluated in constant time (and satisfies the triangle inequality).

(Hint: build a graph with $O(n^2)$ vertices, where each vertex corresponds to a pair of current locations of Alice and Bob... Then apply a known graph algorithm.)

Problem 9.2: We are given a directed graph $G = (V, E)$ with n vertices and m edges, where each edge e has a color $c(e) \in \{1, \ldots, C\}$ and a real weight $w(e) > 0$. We are given two vertices $s, t \in V$, and an integer $k \leq n$.

(a) (5.0 points) Describe an efficient algorithm to compute a path from s to t that changes colors at most k times. Analyze the running time as a function of n, m, and C.

For example, if the sequence of edges of a path has colors $\langle 2, 2, 4, 3, 3, 3, 4, 4, 1, 1, 1, 3, 3 \rangle$, then the path changes colors five times.

(As an application, imagine that the edges of the same color represent the route of a bus line. We want a path that does not require too many bus transfers.)

(Hint: build a new graph with $O(nC)$ vertices and apply a known algorithm.)

(b) (5.0 points) Describe an efficient algorithm to compute a path from s to t that changes colors at most k times, minimizing the total weight of the path. Analyze the running time as a function of n, m, C, and k.

(Hint: build a new graph with $O(nkC)$ vertices and apply a known algorithm.)

Problem 9.3: Let $G = (V, E)$ be an *undirected* unweighted graph with n vertices and m edges. Let L be a parameter with $4 \ln n \leq L \leq n$. We say that two vertices u and v are *far* iff the shortest-path distance between u and v is greater than L. In this problem, you will see that computing shortest-path distances for far pairs is easier than in the general case.

(a) (2.5 points) Pick an arbitrary source vertex s and let V_i be the set of all vertices with shortest-path distance at most i from s.

Prove that for any $\ell \geq 2\ln n$, there exists $i \in \{1, \ldots, \ell\}$ with $|V_i - V_{i-1}| \leq \frac{2|V_{i-1}|}{\ell} \ln n$.

(Hint: the inequality $1 + 2x \geq e^x$ for all $x \in [0, 1]$ might be useful—you may use it without proof.)

(b) (2.5 points) Prove that there exists a partition V into three subsets A, B, C with $|B| = O((|A|/L) \log n)$ and $|A| \neq 0$, such that for every far pair of vertices $u, v \in V$, any path from u to v must use a vertex in B or is entirely contained in C. Moreover, show that such a partition can be computed in $O(n + \sum_{u \in A} \deg(u))$ time.

(Hint: use (a) and a (slightly modified) BFS. Note: A path $\pi = \langle v_1, \ldots, v_k \rangle$ "uses" a vertex z if $z = v_i$ for some $i \in \{1, \ldots, k\}$.)

(c) (2.5 points) Prove that there exists a subset $X \subseteq V$ of $O((n/L) \log n)$ vertices, such that for every far pair of vertices $u, v \in V$, any path from u to v must use a vertex in X. Moreover, show that such a subset X can be computed in $O(n^2)$ time.

(Hint: use (b) iteratively.)

(d) (2.5 points) Give an algorithm to compute the shortest-path distance between u and v for all far pair of vertices u and v, in $O((n^3/L) \log n)$ total time.

(Thus, if the graph is dense and L is large, this is faster than the standard $O((m+n)n)$-time algorithm for all-pairs shortest paths that runs BFSs from all n source vertices.)

(Hint: use (c).)

(Added Clarification: your algorithm does not have to detect which pairs are far. For pairs (u, v) that are not far, the value returned by your algorithm for (u, v) can be anything; for pairs (u, v) that are far, the value returned by your algorithm for (u, v) must be equal to the shortest-path distance between u and v.)