CS 170 HW 4

Due **2020-09-28, at 10:30 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Counting Shortest Paths

This question is a solo question.

Given an undirected unweighted graph G and a vertex s, let $p(v)$ be the number of distinct shortest paths from s to v. We will use the convention that $p(s) = 1$ in this problem. Give an $O(|V| + |E|)$-time algorithm to compute $p(v)$ for all vertices. Only the main idea and runtime analysis are needed.

(Hint: For any v, how can we express $p(v)$ as a function of other $p(u)$?)

3 Dijkstra Tiebreaking

This question is a solo question.

We are given a directed graph G with positive weights on its edges. We wish to find a shortest path from s to t, and, among all shortest paths, we want the one in which the longest edge is as short as possible. How would you modify Dijkstra's algorithm to this end? Just a description of your modification is needed.

(If there are multiple shortest paths where the longest edge is as short as possible, outputting any of them is fine).

4 Bounded Bellman-Ford

Modify the Bellman-Ford algorithm to find the weight of the lowest-weight path from s to t with the restriction that the path must have at most k edges. Just a description of your modification is needed.

5 Arbitrage

Shortest-path algorithms can also be applied to currency trading. Suppose we have n currencies $C = \{c_1, c_2, \ldots, c_n\}$: e.g., dollars, Euros, bitcoins, dogecoins, etc. For any pair of currencies c_i, c_j, there is an exchange rate $r_{i,j}$: you can buy $r_{i,j}$ units of currency c_j at the price of one unit of currency c_i. Assume that $r_{i,i} = 1$ and $r_{i,j} \geq 0$ for all i, j.

The Foreign Exchange Market Organization (FEMO) has hired Oski, a CS170 alumnus, to make sure that it is not possible to generate a profit through a cycle of exchanges; that is, for any currency $i \in C$, it is not possible to start with one unit of currency i, perform a series of exchanges, and end with more than one unit of currency i. (That is called *arbitrage*.)

More precisely, arbitrage is possible when there is a sequence of currencies $c_{i_1}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$. This means that by starting with one unit of currency c_{i_1} and then successively converting it to currencies $c_{i_2}, c_{i_3}, \ldots, c_{i_k}$ and finally back to c_{i_1}, you would end up with more than one unit of currency c_{i_1}. Such anomalies last only a fraction of a minute on the currency exchange, but they provide an opportunity for profit.

We say that a set of exchange rates is arbitrage-free when there is no such sequence, i.e. it is not possible to profit by a series of exchanges.

(a) Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$ which is *arbitrage-free*, and two specific currencies s, t, find the most profitable sequence of currency exchanges for converting currency s into currency t. That is, if you have a fixed amount of currency s, output a sequence of exchanges that gets you the maximum amount of currency t.

Hint: represent the currencies and rates by a graph whose edge weights are real numbers.

(b) Oski is fed up of manually checking exchange rates, and has asked you for help to write a computer program to do his job for him. Give an efficient algorithm for detecting the possibility of arbitrage. You may use the same graph representation as for part (a).

For both parts, give a three-part solution.

6 Updating a MST

This question is a solo question.

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch.

There are four cases. In each, give a description of an algorithm for updating T, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases these may be quite brief. For simplicity, you may assume that no two edges have the same weight using both w and $\hat{w}$.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) < w(e)$

(d) $e \in E'$ and $\hat{w}(e) > w(e)$

7 Job Scheduling

There are n computing jobs to be run on two supercomputers. Each supercomputer can only run one job at a time, and the jth job has processing time $t_j \geq 0$. We want to come up with a *schedule* for these jobs. That is, we want to assign these n jobs to one of the two supercomputers, and decide in what order each supercomputer will run the jobs assigned to it.

For each job i let J_i be the set of jobs run before job i on the supercomputer job i is assigned to. Then job i has *waiting time* $\sum_{j \in J_i} t_j$. Our goal is to minimize the total waiting time of all jobs, $\sum_{i \in [n]} \sum_{j \in J_i} t_j$.

For example, if the jobs have processing times 1, 2, 3, 4, 5, and we assign the first supercomputer the jobs with processing times 1, 3, 2 in that order, and the second supercomputer the jobs with processing times 4, 5 in that order, the total waiting time is $0 + 1 + 4 = 5$ for jobs on the first supercomputer and $0 + 4 = 4$ for jobs on the second supercomputer, so the total waiting time is 9.

(a) Suppose each supercomputer must process exactly $n/2$ jobs, assuming n is even. Given the processing times t_j for each job, give an efficient algorithm for assigning jobs to each supercomputer and ordering the jobs assigned to each.

(Hint: Recall the service scheduling problem from discussion)

(b) Now, you can assign any number of jobs to each supercomputer. Show how we can solve this problem by constructing an instance of part (a) that has the same optimal cost.

Just a main idea and proof of correctness are needed for both parts.

8 (Extra Credit) Min-Sum Multiplication

In the all-pairs shortest path (APSP) problem, we are given an undirected connected graph G with positive edge weights. We want to find the length of the shortest path between *every* pair of vertices.

Closely related to shortest path problems is the min-sum product. The dot product of two vectors a, b is $\sum_i a_i \cdot b_i$. The min-sum product of two vectors is instead $\min_i(a_i + b_i)$. The min-sum product of two matrices is defined analogously to the normal product of two matrices - for two n-by-n matrices A, B, their min-sum product's (i, j)-th entry is the min-sum product of the ith row of A and the jth column of B (instead of their dot product).

Suppose we can compute the min-sum product of two n-by-n matrices in $T(n)$ time. Give an efficient algorithm for the APSP problem, and express your runtime in terms of $T(n)$. Assume $n^2 = O(T(n))$, i.e. the time to write down an n-by-n matrix is not more than the time to multiply two of them. **Give a three-part solution**.