CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani **HW 4**

Due February 15, 11:59pm

1. (15 pts.) Short Questions

For these questions, either provide a short justification for your answers (one or two sentences should be enough) or provide a counterexample (please make your counterexample as small as possible, for the readers' sake).

(a) We want to use the FFT to multiply two polynomials of degree 62 and 83, respectively. What would be the size n of the FFT needed? What is the corresponding ω? What is ω^{64}? (It is a very simple complex number.)

(b) Let G be a dag and suppose the vertices $v_1, \ldots, v_n$ are in topologically sorted order. If there is a path from v_i to v_j in G, are we guaranteed that $i < j$?

(c) Let $G = (V, E)$ be any strongly connected directed graph. Is it always possible to remove one vertex (as well as edges to and from it) from the graph so that the remaining directed graph is strongly connected?

(d) Consider the array `prev` computed by Dijkstra's algorithm ran on a connected undirected graph, starting at any vertex s. Is the graph formed by the edges $\{u, \texttt{prev}(u)\}$ a tree?

(e) Give an example where the greedy set cover algorithm does not produce an optimal solution.

2. (15 pts.) All roads lead to Rome

You are the chief trade minister under Emperor Caesar Augustus with the job of directing trade in the ancient world. The Emperor has proclaimed that *all roads lead to (and from) Rome*; that is, all trade must go through Rome. In particular, you are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a particular node $v_0 \in V$ (Rome). Give an efficient algorithm for finding shortest path between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0 (Rome). Make your algorithm as efficient as you can, perhaps as fast as Dijkstra's algorithm.

(a) Give the efficient algorithm.

(b) Occasionally, Augustus will ask you for the (smallest) distance between two vertices. You want to do this as quickly as possible, so that Augustus does not have your head.

This is called a *distance query*: Given a pair of vertices (u, v), give the the distance of the shortest path that passes through v_0. Describe how you might store the results such that you require $O(|V|)$ storage and from your data structure you can compute the result in $O(1)$ time. For your answer, a clear description of the data structure and its usage is sufficient.

(c) On the other hand, the traders need to know the paths themselves.

This is called a *path query*: Given a part of vertices (u, v), give the shortest path itself, that passes through v_0. Describe how you might store the results such that you require $O(|V|)$ storage and from your data structure you can compute the result in $O(|V|)$ time. Again, a clear description of the data structure and its usage is sufficient.

3. (15 pts.) Unique Shortest Path

Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V|+|E|)\log|V|)$ time.

Input: An undirected graph $G=(V,E)$; edge lengths $l_e > 0$; starting vertex $s \in V$.

Output: A Boolean array `usp[·]`: for each node u, the entry `usp[u]` should be `true` if and only if there is a *unique* shortest path from s to u. (Note: `usp[s] = true`.)

4. (20 pts.) 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 i, j of currencies, 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.

(a) 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*.) Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$ and two specific currencies s, t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. We recommend that you represent the currencies and rates by a graph whose edge lengths are real numbers.

(b) In the economic downturn of 2016, the FEMO had to downsize and let Oski go, and the currencies are changing rapidly, unfettered and unregulated. As a responsible citizen and in light of what you saw in lecture this week, this makes you very concerned: it may now be possible to find currencies $c_{i_1}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \times r_{i_2,i_3} \times \cdots \times r_{i_{k-1},i_k} \times r_{i_k,i_1} > 1$. 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.

You decide to step up to help out the World Bank. Given an efficient algorithm for detecting the presence of such an anomaly. You may use the same graph representation as for part (a).

5. (20 pts.) Hyperloop construction

Elon Musk has perfected Hyperloop technology, and would like to shorten travel time between Palo Alto and Hawthorne. There is already a network of (two-way) roads $G=(V,E)$ connecting a set of cities V. Each road $e \in E$ has a (nonnegative) travel time $\ell(e)$ needed to cross it.

(a) Mr. Musk has enough cash in his checking account to finance the construction of *one* Hyperloop route, out of a set H of possible routes. Like normal roads, each Hyperloop route h would connect two cities (in the same set V) and have a travel time $\ell(h)$. As Mr. Musk's assistant, you are asked which of these Hyperloop routes, if added to the existing network G, would result in the maximum decrease in the driving time between two fixed Palo Alto (s) and Hawthorne (t). Design an efficient algorithm for this problem.

In particular, the problem is:

Input: an undirected graph $G=(V,E)$, a set of candidate edges H (between the same set of vertices V), with both $e \in E$ and $h \in H$ have nonnegative edge lengths $\ell(\cdot)$, and two vertices s, t

Output: the edge $h_{best} \in H$ such that adding it to G reduces the distance from s to t as much as possible.

Clarifications: There might not be an $s-t$ path in G, but there is definitely an $s-t$ path in $G \cup h$ for at least one $h \in H$. Edges in H may connect the same two vertices as edges in E, and their lengths could be greater or smaller than the parallel already-existing edge.

(b) Mr. Musk is feeling more generous, and is willing to bestow even more Hyperloop routes upon California! Rather than paying for one Hyperloop route, he is now willing to pay to build n of them. The rest of the problem is the same as before: design an efficient algorithm to find the set of n Hyperloop routes that will decrease the driving time between Palo Alto and Hawthorne the most.

Note: It's likely that extending your algorithm from part (a) to this problem will result in an exponentially slow algorithm. When we encounter a problem that has extra features that a standard algorithm (Dijkstra's in this case) cannot handle, our choices include using the standard algorithm as a subroutine in a more complex algorithm; modifying the standard algorithm to take into account the extra features; or modifying the problem instance to incorporate additional information so that the standard algorithm will return the correct result. In this case, we recommend you try the last strategy: construct a graph such that every node also includes information about how many Hyperloop routes have been taken to reach it.

6. (15 pts.) Proof of correctness for greedy algorithms

This question guides you through writing a proof of correctness for a greedy algorithm. A doctor's office has n customers, labeled $1, 2, \ldots, n$, waiting to be seen. They are all present right now and will wait until the doctor can see them. The doctor can see one customer at a time, and we can predict exactly how much time each customer will need with the doctor: customer i will take $t(i)$ minutes.

(a) We want to minimize the average waiting time (the average of the amount of time each customer waits before they are seen, not counting the time they spend with the doctor). What order should we use? You do not need to justify your answer for this part. (Hint: sort the customers by ____)

(b) Let $x_1, x_2, \ldots, x_n$ denote an ordering of the customers (so we see customer x_1 first, then customer x_2, and so on). Prove that the following modification, if applied to any order, will never increase the average waiting time:

- If $i < j$ and $t(x_i) \geq t(x_j)$, swap customer i with customer j.

(For example, if the order of customers is $3, 1, 4, 2$ and $t(3) \geq t(4)$, then applying this rule with $i = 1$ and $j = 3$ gives us the new order $4, 1, 3, 2$.)

(c) Let u be the ordering of customers you selected in part (a), and x be any other ordering. Prove that the average waiting time of u is no larger than the average waiting time of x—and therefore your answer in part (a) is optimal.

Hint: Let i be the smallest index such that $u_i \neq x_i$. Use what you learned in part (b). Then, use proof by induction (maybe backwards, in the order $i = n, n-1, n-2, \ldots, 1$, or in some other way).

7. (Bonus 0 pts.) Two-vertex Disjoint Paths

(Note: Only solve this problem if you want an extra challenge. We will not grade this problem.)

Find a polynomial-time algorithm to solve the following problem:

Input: A dag G, and vertices s_1, s_2, t_1, t_2.

Question: Does there exist a path from s_1 to t_1 and a path from s_2 to t_2, such that no vertex appears in both paths?

Your algorithm should have running time $O(|V|^c)$ for some constant c (e.g., $c = 4$ or something like that).