CS 4349.400 Homework 7

Due Wednesday November 7th, in class

Please answer the following questions.

1. You—yes, *you*—can cause a major economic collapse with the power of graph algorithms![1] The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with \$1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with $\$(1 \cdot 120 \cdot 0.01 \cdot 1.2) = \$1.44$! The cycle of currencies $\$ \to ¥ \to € \to \$$ is called an ***arbitrage cycle***. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

   Suppose n different currencies are traded in your currency market. You are given the matrix $R[1 \,..\, n, 1 \,..\, n]$ of exchange rates between every pair of currencies: for each i and j, one unit of currency i can be traded for $R[i, j]$ units of currency j. (Do *not* assume that $R[i, j] \cdot R[j, i] = 1$.)

   (a) To start, let's design an algorithm that returns an array $M[1 \,..\, n]$, where M is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles. We'll do so using a reduction to single source shortest paths.

   Let $G = (V, E, w)$ be an edge weighted graph where $V = \{1, \cdots, n\}$ and E is the set of pairs (i, j) with $i \neq j$. What weights should we use so that a *shortest* path in G from 1 to i is also the best sequence of currency exchanges from 1 to i? *[Hint: Remember your log rules.]*

   (b) Are some of the edge weights negative?

   (c) Which single source shortest path algorithm should we run?

   (d) After running the algorithm, how do we compute each $M[i]$ given $dist(i)$?

   (e) Finally, what is the running time of our algorithm *in terms of n?*

   (f) Now, back to our original goal of finding an arbitrage cycle, if one exists. What would an arbitrage cycle correspond to in the graph G?

[1] No, you can't.

2. Suppose we are given an edge-weighted directed graph $G = (V, E, w)$. We'll assume that G contains no negative-length cycles.

   When we began discussing all-pairs shortest paths, we considered the following recursive definition of $dist(u, v)$, the length of the shortest path from vertex u to vertex v:

$$dist(u,v) = \begin{cases} 0 & \text{if } u = v \\ \min_{x \to v} \left(dist(u,x) + w(x \to v) \right) & \text{otherwise} \end{cases}$$

   Unfortunately, this recurrence doesn't work, because to compute $dist(u, v)$, we'll need to compute $dist(u, x)$ for some other vertices x. But to compute $dist(u, x)$, we may need to compute $dist(u, v)$. Fortunately, we avoid this behavior if G is a directed acyclic graph (a DAG).

   (a) Briefly explain why we will not get into an infinite loop evaluating $dist(u, v)$ if G is a DAG.

   (b) Fix a vertex s, and suppose we want to compute $dist(s, v)$ for all vertices v in the DAG G. In order to compute distances from s to every other vertex via dynamic programming, we need an *evaluation order* for the different subproblems $dist(s, v)$. Describe a simple $O(V + E)$ time algorithm to order the vertices so that all of its dependencies $dist(s, x)$ are already available when it comes time to evaluate $dist(s, v)$. Remember, reductions to algorithms seen in class are not only allowed, but encouraged.

   (c) What is the total time needed to run the algorithm from part (b) *and* to compute $dist(s, v)$ for all $v \in V$ using dynamic programming? You may assume that every vertex in G is reachable from s.

3. Let $G = (V, E, w)$ be a connected directed graph ***with non-negative edge weights***. Let H be a subgraph of G obtained by deleting some edges.

   (a) Let t be a vertex of G. Describe an $O(E \log V)$ time algorithm to compute the shortest path distances *from* each vertex v *to* t. *[Hint: Make a simple modification to the graph and run Dijkstra's algorithm.]*

   (b) Let s and t be vertices of G. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe an $O(E \log V)$ time algorithm that chooses the best edge to reinsert. *[Hint: Use the algorithm from part (a).]*