CS 161 Fall 2017: Section 6 Solutions

Currency Exchange

Suppose the various economies of the world use a set of currencies $C_1, \dots, C_n$—think of these as dollars, pounds, bitcoins, etc. Your bank allows you to trade each currency C_i for any other currency C_j, and finds some way to charge you for this service (in a manner to be elaborated in the subparts below). We will devise algorithms to trade currencies to maximize the amount we end up with.

(a) Suppose that for each ordered pair of currencies (C_i, C_j), the bank charges a flat fee of $f_{ij} > 0$ dollars to exchange C_i for C_j (regardless of the quantity of currency being exchanged). Devise an efficient algorithm which, given a starting currency C_s, a target currency C_t, and a list of fees f_{ij} for all $i, j \in \{1, \dots, n\}$, computes the cheapest way (that is, incurring the least in fees) to exchange all of our currency in C_s into currency C_t. Justify the correctness of your algorithm and its runtime.

Build the complete graph on the currencies with weights equal to the corresponding fees; i.e. $G = (V, E, w)$ where $V = \{C_1, \dots, C_n\}$, $E = \{(C_i, C_j) : i \neq j\}$, and $w(C_i, C_j) = f_{ij}$. Run Dijkstra's algorithm from C_s; return a shortest $C_s \to C_t$ path.
Correctness: Notice that any path from C_s to C_t in G indicates a sequence of exchanges, and that its total weight is precisely the sum of the fees needed to perform those exchanges. Thus it suffices to find a $C_s \to C_t$ path in G of minimum weight. Note that the f_{ij}'s are all positive, so this is a valid input to Dijkstra's algorithm.
Running time: $O(n^2)$ total. Since all exchange pairs are possible, we have $m = \binom{n}{2} = \Theta(n^2)$ edges—this is also how long it takes to build G. Dijkstra's algorithm therefore takes $O(n^2 + n \log n) = O(n^2)$ time using the Fibonacci heap implementation. Other implementations are possible; for example, using a red-black tree or binary heap leads to $O((m + n) \log n) = O(n^2 \log n)$.

(b) Consider the more realistic setting where the bank does not charge flat fees, but instead uses exchange *rates*. In particular, for each ordered pair (C_i, C_j), the bank lets you trade one unit of C_i for $r_{ij} > 0$ units of C_j. Devise an efficient algorithm which, given starting currency C_s, target currency C_t, and a list of rates r_{ij}, computes a sequence of exchanges that results in the greatest amount of C_t. Justify the correctness of your algorithm and its runtime.[Hint: How can you turn a product of terms into a sum?]

Build $G = (V, E, w)$ as in part (a), but with weights $w(C_i, C_j) = -\log(r_{ij})$. Run Bellman-Ford from C_s and return a shortest $C_s \to C_t$ path.
Correctness: As in part (a), a sequence of exchanges corresponds to a path in G. However, here we want a path $C_{i_1} = C_s, C_{i_2}, \dots, C_{i_k} = C_t$ that maximizes $\prod_{\ell=1}^{k-1} r_{i_\ell, i_{\ell+1}}$. Since log is a monotonically increasing function (i.e. if $a \geq b$, then $\log(a) \geq \log(b)$), this is the same as maximizing $\log\left(\prod_{\ell=1}^{k-1} r_{i_\ell, i_{\ell+1}}\right) = \sum_{\ell=1}^{k-1} \log(r_{i_\ell, i_{\ell+1}})$. Finally, this is equivalent to minimizing $\sum_{\ell=1}^{k-1} -\log(r_{i_\ell, i_{\ell+1}}) = \sum_{\ell=1}^{k-1} w(C_{i_\ell}, C_{i_{\ell+1}})$, which is the shortest path objective. Note that we must use Bellman-Ford rather than Dijkstra's algorithm, since these weights may be negative.
Running time: $O(n^3)$ total. G can be built in time $O(n^2)$ time, and Bellman-Ford takes $O(n^3)$ time since we have $\Theta(n^2)$ edges.

(c) Due to fluctuations in the markets, it is occasionally possible to find a sequence of exchanges that lets you start with currency A, change into currencies, B, C, D, etc., and then end up changing back to currency A in such a way that you end up with more money than you started with—that is, there are currencies $C_{i_1}, \ldots, C_{i_k}$ such that

$$r_{i_1 i_2} \times r_{i_2 i_3} \times \cdots \times r_{i_{k-1} i_k} \times r_{i_k i_1} > 1.$$

Devise an efficient algorithm that finds such an anomaly if one exists. Justify the correctness of your algorithm and its runtime.

Run Bellman-Ford on the same graph as in part (b); then execute one more iteration of Bellman-Ford to check if there is a negative cycle in G. If there is, the cycle is the anomaly—trading currencies along the cycle will result in a profit.
Correctness: A currency anomaly $\prod_{\ell=1}^{k-1} r_{i_\ell, i_{\ell+1}} > 1$ implies (by the same log manipulations we did in part (b)) that $\sum_{\ell=1}^{k-1} w(C_{i_\ell}, C_{i_{\ell+1}}) = \sum_{\ell=1}^{k-1} -\log(r_{i_\ell, i_{\ell+1}}) < 0$. Thus there is a negative cycle in G, which can be found by an extra iteration of Bellman-Ford.
Running time: $O(n^3)$ total. We are doing the same thing as in part (b), plus one extra iteration which takes $O(|E|) = O(n^2)$ time.

Traveling Across the Country

We have a graph representation of the country, where nodes u_i are on the east coast and nodes v_j are on the west coast, with n nodes in total. We also have $|E|$ undirected edges representing distances between these cities.

(a) Design an efficient algorithm to compute the shortest path starting at *any* city on the east coast and ending at *any* city on the west coast.

Add two nodes to the graph: one node E connected to all cities on the east coast with edge weight 0, and one node W connected to all cities on the west coast also with edge weight 0. Run Dijkstra's on the new graph starting at E, and return the path/length to W. The asymptotic runtime is the same as that of Dijkstra's on the same graph.

(b) This time, we start from a specific city u_i and end at a specific city v_j. However, we impose an additional restriction that we must traverse one of the edges between two cities 3 times: that is, for some w, w', we must traverse $w \to w'$, $w' \to w$, and then $w \to w'$ again. Design an efficient algorithm to find the shortest path from u_i to v_j with this additional constraint.

Run Dijkstra's once starting at u_i and once starting at v_j. Now, we have the distance/path from every city to u_i and v_j. For each edge (a, b) in the graph, check the path length given by $D(u_i, a) + 3W_{ab} + D(v_j, b)$ and $D(u_i, b) + 3W_{ab} + D(v_j, a)$. Take the minimum length path. The asymptotic runtime is the same as that of Dijkstra's because we run Dijkstra's twice and iterate over all edges once.