CSE 101 Homework 3

Spring 2017

Question 1 (Single Sink Shortest Paths, 10 points). The shortest path algorithms we have discussed tend to compute single source shortest paths. Given a (potentially weighted and directed) graph \( G \) and a source vertex \( s \), they compute the length of all the shortest paths from \( s \) to each other vertex \( v \). Show how such an algorithm can be used to in essentially the same runtime, compute all the single sink shortest paths. That is for some vertex \( t \) compute the lengths of all the shortest paths from \( v \) to \( t \) for each other vertex \( v \).

Solution:
To compute the shortest paths from a vertex \( s \) to each other vertex \( v \) in a graph \( G \), we can run a single source shortest path algorithm (BFS for unweighted or Dijkstra/Bellman for weighted graphs) starting \( s \).

The problem requires us to calculate the shortest paths of each vertex \( v \) to some vertex \( t \). For this, we create a graph \( G' \) by reversing the edges of \( G \). Now, run the same single source shortest path algorithm on \( G' \) starting at vertex \( t \) as the source.

Time Complexity: \( G' \) can be created in \( O(|V| + |E|) \). The runtime of single source shortest path algorithm remains same as for graph \( G \).

Question 2 (Price Functions for DAGs, 15 points). Recall from class that if we are trying to compute shortest paths in a graph with some price function \( \ell(v, w) \), we can obtain a new shortest paths problem by letting \( \ell'(v, w) = \ell(v, w) + f(w) - f(v) \) for some function \( f : V \to \mathbb{R} \). Furthermore, we showed that the shortest paths obtained for the distance function \( \ell \) and the shortest paths for \( \ell' \) are always the same.

It is of some interest to attempt to find functions \( f \) so that the resulting weight function \( \ell' \) is always non-negative. If \( G \) is a DAG, find a linear time algorithm to compute such an \( f \). Hint: we just need that \( f(w) \) is much bigger than \( f(v) \) for all edges \((v, w)\). You can figure out how to do this by topologically sorting \( G \).

Solution:
If we let \( r(v) \) be the rank of \( v \) in the topological ordering, we note that for each edge \((v, w)\) that \( r(w) \geq r(v) + 1 \).
Since we want \( f(w) \) to be much bigger than \( f(v) \), it suffices to take \( f \) proportional to \( r \). In particular, if we let \( C \) be a large enough constant so that \( \ell(v, w) \geq -C \) for all edges \((v, w)\), we can simply take \( f(v) = C v \). Then for each edges \( \ell'(v, w) = \ell(v, w) + C(r(w) - r(v)) \geq -C + C = 0 \).

Question 3 (Roadtrip Planning, 50 points). Jane is returning from a roadtrip, but is low on funds. She currently has \( N \) dollars on hand and is located in Sourcetown and need to get to Sinkville. She has a road map of the country she is driving in given by a graph \( G \), with vertices \( s \) and \( t \) corresponding to Sourcetown and Sinkville respectively. Additionally, each directed edge of \( G \) has an associated cost which is how much she must pay to travel along that edge. However, she has some friends willing to pay her to make some journeys, so some of these costs may be negative. She would like to have an algorithm that given \( G, s, t, N, \) and the cost function to determine whether or not there is a path from \( s \) to \( t \) so that at no point along the path would her total amount of money fall below 0 dollars.

(a) Devise an \( O(|V| + |E|) \) time algorithm for this problem under the assumption that \( G \) has no negative weight cycles. Hint: Find a way to modify Bellman-Ford to disallow paths that would put her total money below 0. [25 points]
(b) Adapt this algorithm to work even if $G$ is allowed to have negative weight cycles. Hint: Find a way of detecting whether or not Jane can exploit such a cycle in her path. [25 points]

Solution:

(a) We would modify the update rule for Bellman Ford Algorithm. In this case, $d_k(v)$ would be defined as the cost required to reach vertex $v$ using at most $k$ roads. While computing the value of $d_k(v) = \min_w(d_{k-1}(w) + l(w, v))$, $d_k(v)$ is updated only if $d_k(v) \leq N$, else it’s default value is $\infty$. This constraint ensures that the cost from the $s$ is updated only when it is possible to reach vertex $v$ if the total money doesn’t drop below $\$N$ during the journey. Runtime is same as that of Bellman Ford of $O(|V||E|)$.

(b) 1. First run DFS on the reverse graph to find all vertices from which it is possible to reach $t$. Remove from the graph any vertices that $t$ cannot be reached from (as going to these cities will never allow us to get to our final destination).

2. Run the same algorithm (modified Bellman ford) as in part (a). If there is a path without repeating vertices from $s$ to $t$ satisfying the money constraint that the path never exceeds cost of $N\$, return True.

3. Update all the edges once again. If shortest path for any vertex changes, there is a negative weight cycle. Not only that, but there must be a negative weight cycle that Jane can reach and traverse once without running out of money. This means that Jane can traverse this cycle as many times as she likes in order to get an arbitrarily large amount of money. Since we already removed all vertices from which $t$ could not be reached, after doing this enough times, she will have enough money to reach $t$, so in this case, you can return True.

4. Otherwise there is no such path from $s$ to $t$ which satisfies the money constraint, return False

**Time Complexity:** Step 1 is a DFS and runs in time $O(|V| + |E|)$. Step 2 is the same as part (a), it runs in $O(|V||E|)$. Step 3 takes $O(|V| + |E|)$.

**Question 4** (Related Divide and Conquer Recurrences, 25 points). We consider some runtime recurrences similar to those found in divide an conquer algorithms.

(a) Show that if $T(1) = 1$ and $T(n) = 2T(n-1)$ for $n > 1$ that $T(n)$ grows exponentially. [5 points]

(b) Show that if $T(1) = 1$ and $T(n) = T(n-1) + T([n/2])$ for $n > 1$ that there exist constants $C > c > 0$ so that for all $n > 10$ we have that $n^{C \log_2(n)} > T(n) > n^{c \log_2(n)}$. Hint: First show that $T$ is increasing. For the upper bound, show that $T(n) \leq nT(n/2)$. For the lower bound, show that $T(n) \geq (n/2)T(n/4)$, and use induction in either case. [20 points]

Solution:

(a)

$$T(n) = 2T(n-1)$$
$$= 2^2T(n-2)$$
$$= 2^3T(n-3)$$
$$\vdots$$
$$= 2^kT(n-k)$$

Setting $k = n - 1$, we get $T(n) = 2^{n-1}$
(b) First we prove by induction on \( n \) that \( T(x) \) is increasing for \( x \leq n \). This clearly holds for \( n = 1 \), and if \( T \) is increasing on \([0, n]\) and \( 0 < x < y \leq n + 1 \) then
\[
T(x) = T(x - 1) + T(x/2) \leq T(y - 1) + T(y/2) = T(y).
\]
This completes the inductive hypothesis and proves that \( T \) is increasing.

We’ll first prove upper bound on \( T(n) \) and then we’ll prove lower bound on \( T(n) \).

(1)
\[
T(n) = T(n - 1) + T(n/2) \\
= T(n - 2) + T(n/2) + T((n - 1)/2) \\
= ... \\
= T(n/2) + T((n - 1)/2) + T((n - 2)/2) + ... + T(1) \\
\leq T(n/2) + T(n/2) + ... + T(n/2) \quad (\text{As } T(n) \text{ is a monotonic function.}) \\
\leq nT(n/2)
\]

Now,
\[
T(n) \leq nT(n/2) \\
\leq n(n/2)(n/4)...(n/2^{\log_2(n)+1})T(1) \\
\leq n^{\log_2(n)}/(2^{\log_2(n)(\log_2(n)-1)/2}) \\
\leq n^{\log_2(n)}/(n^{\log_2(n)-1/2}) \\
\leq n^{\log_2(n)/2+1/2} \\
\leq n^{C\log_2(n)}
\]

(2)
\[
T(n) = T(n - 1) + T(n/2) \\
= T(n - 2) + T(n/2) + T((n - 1)/2) \\
= ... \\
= T(n/2) + T((n - 1)/2) + T((n - 2)/2) + ... + T((n/2 + 1)/2) \\
\geq T(n/4) + T(n/4) + T(n/4) + ... + T(n/4) \quad (\text{As } T(n) \text{ is a monotonic function.}) \\
\geq (n/2)T(n/4)
\]

Now,
\[
T(n) \geq (n/2)T(n/4) \\
\geq (n/2)(n/8)T(n/16) \\
\geq ... \\
\geq (n/2)(n/8)...n^{1/2}T(n^{1/2}/2) \\
\geq (n^{1/2})^{\log_4(n^{1/2})} \\
\geq n^{c\log_2(n)}
\]

It is easy to see that \( C > c > 0 \).

**Question 5** (Extra credit, 1 point). *Approximately how much time did you spend working on this homework?*