# CSCI 570 - Fall 2019 - HW 4 Solutions

## 1 Graded Problems

1. Suppose you were to drive from USC to Santa Monica along I-10. Your gas tank, when full, holds enough gas to go $p$ miles, and you have a map that contains the information on the distances between gas stations along the route. Let $d_1 < d_2 < ... < d_n$ be the locations of all the gas stations along the route where $d_i$ is the distance from USC to the gas station. We assume that the distance between neighboring gas stations is at most $p$ miles. Your goal is to make as few gas stops as possible along the way. Give the most efficient algorithm to determine at which gas stations you should stop and prove that your strategy yields an optimal solution. Give the time complexity of your algorithm as a function of $n$.

   **Solution**: The greedy strategy we adopt is to go as far as possible before stopping for gas. That is when you are at the $i^{th}$ gas station, if you have enough gas to go the $i + 1^{th}$ gas station, then skip the $i^{th}$ gas station. Otherwise stop at the $i^{th}$ station and fill up the tank.

   Let $\{g_1, g_2, \ldots, g_m\}$ be the set of gas stations at which our algorithm made us refuel. We next prove that our choice is optimal.

   Let $\{h_1, h_2, \ldots, h_k\}$ be an optimal solution.

   Since it is not possible to get to the $g_1 + 1^{th}$ gas station without stopping, any solution should stop at either $g_1$ or a gas station before $g_1$, thus $h_1 \leq g_1$. If $h_1 < g_1$, then we can swap its first stop with $g_1$ without changing the number of stops. The new solution we obtain $\{g_1, h_2, \ldots, h_k\}$ is legal since when leaving $g_1$ we have at least as much fuel now as we had before swapping. Hence $\{g_1, h_2, \ldots, h_k\}$ is an optimal solution.

   Assume that $\{g_1, g_2, \ldots, g_{c-1}, h_c, \ldots, h_k\}$ is an optimal solution (induction hypothesis). From the greedy strategy taken by our algorithm, $h_c \leq g_c$. If $h_c < g_c$, then by swapping $g_c$ and $h_c$, we get $\{g_1, g_2, \ldots, g_{c-1}, g_c, h_{c+1}, \ldots, h_k\}$ which is indeed a legal solution. The legality follows from the same reasoning as above. That is, when leaving $g_c$ we now have at least as much fuel as we did before swapping and can hence reach the destination. Thus $\{g_1, g_2, \ldots, g_c, h_{c+1}, \ldots, h_k\}$ is an optimal solution.

By induction, it thus follows that $\{g_1, g_2, \ldots, g_m\}$ is an optimal solution.

The running time is $\mathcal{O}(n)$ since we at most make one computation/decision at each gas station.

2. Consider the following modification to Dijkstra's algorithm for single source shortest paths to make it applicable to directed graphs with negative edge lengths. If the minimum edge length in the graph is $-w < 0$, then add $w + 1$ to each edge length thereby making all the edge lengths positive. Now apply Dijkstra's algorithm starting from the source s and output the shortest paths to every other vertex. Does this modification work? Either prove that it correctly finds the shortest path starting from $s$ to every vertex or give a counter example where it fails.

   **Solution 1**: No, the modification does not work. Consider the following directed graph with edge set $\{(a, b), (b, c), (a, c)\}$ all of whose edge weights are $-1$. The shortest path from $a$ to $c$ is $\{(a, b), (b, c)\}$ with path cost $-2$. In this case, $w = 1$ and if $w + 1 = 2$ is added to every edge length before running Dijkstra's algorithm starting from a, then the algorithm would output $(a, c)$ as the shortest path from $a$ to $c$ which is incorrect.

   **Solution 2**: Another way to reason why the modification does not work is that if there were a directed cycle in the graph (reachable from the chosen source) whose total weight is negative, then the shortest path is not well defined since you could traverse the negative cycle multiple times and make the length arbitrarily small. Under the modification, all edge lengths are positive and hence clearly the shortest paths in the original graph are not preserved.

3. Solve Kleinberg and Tardos, Chapter 4, Exercise 3.

   **Solution**: Assume the greedy algorithm currently in use fits boxes $b_1, b_2, \ldots, b_j$ into the first $k$ trucks. We prove that no other algorithm can fit more boxes into $k$ trucks, i.e., if an algorithm fits boxes $b_1, b_2, \ldots, b_i$ into the first $k$ trucks, then $i \leq j$. We prove this claim by induction on $k$:

   - For $k = 1$: the greedy fits as many boxes into one truck as possible, it is clear that no other algorithm can x more boxes into the truck, thus, $i \leq j$.
   - Assume it holds for $k - 1$, i.e., if the greedy algorithm fits boxes $b_1, b_2, \ldots, b_j$ into the first $k - 1$ trucks, and the other algorithm fits boxes $b_1, b_2, \ldots, b_i$ into the first $k - 1$ trucks, then $i \leq j$.
   - For $k$: the greedy algorithm fits $j'$ boxes into the first $k - 1$ trucks, the other algorithm fits $i'$ boxes into the first $k - 1$ trucks, and $i' \leq j'$; now, for the $k$-th truck, the other algorithm packs in boxes $b_{i'+1}, \ldots, b_i$; since $i' \leq j'$, the greedy algorithm is able at least to t

all the boxes $b_{j'+1}, \ldots, b_i$ into the $k$-th truck, and it may be able to fit more

4. Solve Kleinberg and Tardos, Chapter 4, Exercise 5.

   **Solution**:

   (a) Sort the list of houses from west to east.

   (b) Place a base station exactly 4 miles east of the first house in the list. Remove all the houses in the list that are covered by the base station and then repeat step 2.

   The algorithm terminates and every house is covered since we remove houses only after they are covered and at each iteration of step 2 at least one house is covered. The running time of the algorithm is dominated by the sorting in step 1 and is hence $\mathcal{O}(n \log n)$.

   Let $\{s_1, s_2, \ldots, s_k\}$ denote the list of base station locations as determined by our algorithm and let $\{t_1, t_2, \ldots, t_m\}$ denote the choice of locations made by an optimal solution. Assume that both the lists are sorted in order from west to east.

   We claim that for all $i \in \{1, 2, \ldots, m\}$, either $t_i = s_i$ or $t_i$ is to the west of $s_i$.

   This is true when $i = 1$ for otherwise the western most house is not covered. Assume that the claim is true for $i = c - 1$ (induction hypothesis).

   All the houses to the west of $s_{c-1}$ are covered by $\{s_1, s_2, \ldots, s_{c-1}\}$ by the construction of our algorithm. By the induction hypothesis, $t_{c-1}$ is either $s_{c-1}$ or $t_{c-1}$ is further west of $s_{c-1}$. Thus the first house eastward of $s_{c-1}$ is not covered by $\{t_1, t_2, \ldots, t_{c-1}\}$. This implies that either $t_c = s_c$ or $t_c$ is to the west of $s_c$ for otherwise the first house eastward of $s_{c-1}$ is not covered by the optimal solution. Our claim thus follows by induction.

   The claim implies in particular that the number of base station in our solution to the west of any point is at most as many as the number of base stations in the optimal solution to the west of that point. Thus $k \leq m$ and our solution is optimal.

# 2 Practice Problems

1. Solve Kleinberg and Tardos, Chapter 4, Exercise 4.

   **Solution**: Let the two input sequences be $S = (s_1, s_2, \ldots, s_n)$ and $S' =$

$(r_1, r_2, \ldots, r_m)$.

We propose the following greedy algorithm. Find the first event in $S$ that is the same as $r_1$. If you cannot find such an event, output "no" and terminate. Say $s_{i_1}$ is the first event in $S$ that is the same as $r_1$. Remove the first $i_1$ events from $S$, that is $S = (s_{i_1+1}, s_{s_1+2}, \ldots, s_n)$. In the second iteration, find the first event in $S$ that is the same as $r_2$. If you cannot find such an event, output "no" and terminate. Say $s_{i_2}$ is the first event in $S$ that is the same as $s_2$. Set $S = (s_{i_2+1}, s_{i_2+2}, \ldots, s_n)$ and so on. If the algorithm runs successfully for $m$, iterations then output "yes".

Clearly, if the algorithm runs successfully for $m$ iterations, then $S'$ is indeed a substring of $S$ (since $(s_{i_1}, s_{i_2}, \ldots, s_{i_m}) = S'$) and thus and our algorithm is correct.

The harder direction is to prove that if $S'$ is a substring of $S$, then our algorithm indeed outputs "yes". We prove the following slightly stronger claim.

Claim: If $(s_{k_1}, s_{k_2}, \ldots, s_{k_m}) = S'$, then $s_{i_j} = s_{k_j}$ and $i_j \leq k_j$ for all $j \in \{1, 2, \ldots, m\}$

We prove the claim by induction on $j$. The case $j = 1$ follows from the first step of our algorithm. Assume (induction hypothesis) that the claim holds for $j = c - 1$. The induction hypothesis implies that the algorithm ran successfully for at least $c - 1$ steps. Since $s_{k_c} = r_{k_c}$ and $k_c > k_{c-1} \geq i_{c-1}$, the $c^{th}$ step of the algorithm ran successfully and found $s_{i_c}$ such that $s_{i_c} = s_{k_c}$. Further since $i_c$ is the smallest index greater than $i_{c-1}$ such that $s_{i_c} = r_{i_c}$, it is true that $k_c \geq i_c$ and the claim follows.

The running time of the algorithm is $\mathcal{O}(n+m)$ as we examine each element in the sequences at most once.

2. Solve Kleinberg and Tardos, Chapter 4, Exercise 8.

   **Solution**: Suppose by way of contradiction that $T$ and $\hat{T}$ are two distinct minimum spanning trees for the graph. Since $T$ and $\hat{T}$ have the same number of edges but are not equal, there exists an edge $e \in T$ such that $e \notin \hat{T}$. Adding $e$ to $\hat{T}$ results in a cycle $C$. As edge weights are distinct, take $e'$ to be the uniquely most expensive edge in $C$. By the cycle property, no minimum spanning tree can contain $e'$, contradicting the fact that the edge $e'$ is in atleast one of the trees $T$ or $\hat{T}$.

3. Solve Kleinberg and Tardos, Chapter 4, Exercise 22.

   **Solution**: Consider the following counterexample. Let $G = (V, E)$ be a weighted graph with vertex set $V = \{v_1, v_2, v_3, v_4\}$ and edges $(v_1, v_2), (v_2,$

$v_3), (v_3, v_4), (v_4, v_1)$ of cost 2 each and an edge $(v_1, v_3)$ of cost 1. Then every edge belongs to some minimum spanning tree, but a spanning tree consisting of three edges of cost 2 would not be minimum.