CSCI 570 - Spring 2021 - HW 2

Due Sunday Feb. 22 (by 4:00 AM)

Problem 1 (20 points)

Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the i-th element of set A, and let b_i be the i-th element of set B. You then receive a payoff on $\prod_{i=1}^{n} a_i^{b_i}$. Give an algorithm that will maximize your payoff (6 points). Prove that your algorithm maximizes the payoff (10 points) and state its running time (4 points).

solution:[Rubric] (20 points)
Giving an algorithm (6 points):
Sort both A and B in the same order (either both ascending or both descending). This takes $O(n \log n)$. What's important is that the largest element of A is matched with the largest of B, the second-largest of each are matched, and so on (greedy solution).
Prove the correctness of the algorithm (10 points):
For purpose of the proof, fix the order of A to be in ascending order, i.e., $a_1 \leqslant a_2 \leqslant ... \leqslant a_n$; we will consider all solutions in terms of how their B arranged relative to this A. My solution has B sorted ascendingly.

For any arbitrary solution (could be an optimal solution):$\{b_i\}$, where b_i is the element of B matched with a_i in this solution. Suppose there exist an inversion $b_i > b_j$, for $i > j$. Now we prove that undoing this inversion will make the result no less than the original one, and has the effect of multiplying the original solutions quantity by this value:

$$\frac{a_i^{b_j} \times a_j^{b_i}}{a_i^{b^i} \times a_j^{b_j}} = \frac{a_j^{b_i - b_j}}{a_i^{b_i - b_j}} \tag{1}$$

Because $\frac{a_j}{a_i} \geq 1$ and $b_i - b_j > 0$, the result above should be no less than 1, i.e., every inversion relative to this solution can be removed without negatively affecting the quantity, which indicates the optimality of the greedy solution.
Complexity of the algorithm (4 points): The sorting takes $O(n \log n)$, so the complexity of the algorithm is $O(n \log n)$.

Problem 2 (20 points)

Suppose you are 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 i-th 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 (6 points) and prove that your strategy yields an optimal solution (10 points). Give the time complexity of your algorithm as a function of n (4 points). Suppose the gas stations have already been sorted by their distances to USC.

solution: [Rubric] (20 points)
Giving an algorithm (6 points):
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,
Prove the correctness of the algorithm (10 points):
Let $\{g_1, g_2, ..., g_m\}$ be the set of gas stations at which our algorithm made us refuel. We next prove that our choice is optimal.
Let $\{h_1, h_2, ..., h_k\}$ 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 $h_1 \leqslant g_1$. 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 $\{g_1, h_2, ..., h_k\}$ is legal since when leaving g_1 we have at least as much fuel now as we had before swapping. Hence $\{g_1, h_2, ..., h_k\}$ is an optimal solution.
Assume that $\{g_1, g_2, ..., g_{c-1}, h_c, ..., h_k\}$ is an optimal solution (induction hypothesis). From the greedy strategy taken by our algorithm, $h_c \leqslant g_c$. If $h_c \leqslant g_c$, then by swapping g_c and h_c, we get $\{g_1, g_2, ..., g_{c-1}, g_c, h_{c+1}, ..., h_k\}$ 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. This $\{g_1, g_2, ..., g_c, h_{c+1}, ..., h_k\}$ is an optimal solution.
By induction, it thus follows that $\{g_1, g_2, ..., g_m\}$ is an optimal solution.
Complexity of the algorithm (4 points):
The running time is $O(n)$ since we at most make one computation/decision at each gas station.

Problem 3 (20 points)

Some of your friends have gotten into the burgeoning field of time-series data mining, in which one looks for patterns in sequences of events that occur over time. Purchases at stock exchanges-what's being bought-are one source of data with a natural ordering in time. Given a long sequence S of such events, your friends want an efficient way to detect certain "patterns" in them—for example, they may want to know if the four events

buy Yahoo, buy eBay, buy Yahoo, buy Oracle

occur in this sequence S, in order but not necessarily consecutively.

They begin with a collection of possible events (e.g., the possible transactions) and a sequence S of n of these events. A given event may occur multiple times in S (e.g., Yahoo stock may be bought many times in a single sequence S). We will say that a sequence S' is a subsequence of S if there is a way to delete certain of the events from S so that the remaining events, in order, are equal to the sequence S'. So, for example, the sequence of four events above is a subsequence of the sequence

buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo, buy Oracle

Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of S. So this is the problem they pose to you: Give an algorithm that takes two sequences of events-S' of length m and S of length n, each possibly containing an event more than once-and decides in time $O(m+n)$ whether S' is a subsequence of S. Prove that your algorithm outputs "yes" if S' is indeed a subsequence of S (hint: induction).

Solution: [Rubric] (20 points)

Let the two input sequences be $S = (s_1, s_2, \ldots, s_n)$ and $S' = (r_1, r_2, \ldots, r_m)$.

Correctness of the algorithm (10 points)
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".

Show the algorithm outputs "yes" (7 points) 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.

Analyze the runtime (3 point) The running time of the algorithm is $\mathcal{O}(n + m)$ as we examine each element in the sequences at most once

Problem 4 (20 points)

You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box that arrived after his make it to Boston faster. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking. Maybe one could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed.

Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed. Your proof should follow the type of analysis we used for the Interval Scheduling Problem: it should establish the optimality of this greedy packing algorithm by identifying a measure under which it "stays ahead" of all other solutions.

Solution: [20 points]

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: **[Using induction (2 point)]**

- For $k = 1$: the greedy fits as many boxes into one truck as possible, it is clear that no other algorithm can fix more boxes into the truck, thus, $i \leq j$. **[4 point]**

- 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 $i \leq j$. **[6 points]**
- 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 fit all the boxes $b_{j'+1}, \ldots, b_i$ into the k-th truck, and it may be able to fit more. **[8 points]**