# Homework 2

Shreenidhi Hegde

## Problem 1

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 $\prod_{i=1}^{n} {a_i}^{bi}$ . Give an algorithm that will maximize your payoff .Prove that your algorithm maximizes the payoff and state its running time. Prove that your algorithm maximizes the payoff and state its running time.

## Solution :

Consider two sets A and B where $a_i$ is the $i^{th}$ element of Set A and $b_i$ be the $i^{th}$ element of set B. We do reorder both the sets in the decreasing order of positive integers.
Now we have set A where $a_1 \geq a_2 \geq \ldots. \geq a_n$ and set B where $b_1 \geq b_2 \geq \ldots. \geq b_n$ .
Now if we receive a payoff on $\prod_{i=1}^{n} {a_i}^{bi}$ , this gives us the maximum payoff.

### Complexity:

Sorting both the sets in decreasing order of positive integers takes *O(n log n)* times. Hence our complexity of the algorithm is *O(n log n)* and calculating the payoff takes *O(n).* Therefore overall time complexity is *O(n + n log n) = O(n log n).*

### Proof :

Let's S be our greedy solution in which $a_1$ is paired with $b_1$ and $a_q$ is paired with $b_q$ . Consider there is another solution S' , in which $a_1$ is paired with $b_q$ and $a_q$ is paired with $b_1$ . $All\ the$ other terms are paired same as that of solution S. Now , if we take the ration of both the terms in set S and S', we get :

$$\frac{\Pi_S {a_i}^{b_i}}{\Pi_{S'} {a_i}^{b_i}} = \frac{(a_1)^{b_1}\ (a_q)^{b_q}}{(a_1)^{b_q}\ (a_q)^{b_1}}$$
$$= \frac{(a_1)^{b_1 - b_q}}{(a_q)^{b_1 - b_q}}$$
$$= \left(\frac{a_1}{a_q}\right)^{b_1 - b_q}$$

Since we know that $a_1 \geq a_q$ and $b_1 \geq b_q$ as we have sorted them in the decreasing order. This gives us the ratio of payoffs of *S* to *S′* greater than 1. i.e $\frac{\Pi_S {a_i}^{b_i}}{\Pi_{S'} {a_i}^{b_i}} \geq 1$. This contradicts the statement that S' is the optimal solution. Therefore $a_1$ should be paired with $b_1$ to maximise the payoff. Hence we prove that our solution *S* is the optimal solution.

## Problem 2:

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 $d1 < d2 < ... < d_n$ 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 neighbouring 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. Suppose the gas stations have already been sorted by their distances to USC.

## Solution :

We use greedy solution to be the optimal solution.

First, we start with full tank. Now our vehicle can travel up to *p* miles. As we already have the map with distances between the gas stations, we will go till the farthest gas station which we can reach within *p* miles. At first stop, we will fill the tank full and start again. Travel until next farthest gas station within *p* miles and so on. Basically, as we have the distances given, we first go to the farthest station within *p* miles. Then we add up *p* distance to already travelled distance and keep checking at each stations whether we can go till the next available gas station. We skip the current gas station , if we can go until next gas station. Otherwise, we fill up the gas there.

### Time Complexity:

As the gas stations have been already sorted according to their distances from USC, Our runtime complexity will be *O(n)* as at max we will have to stop at each gas stations.

### Proof:

This is a greedy algorithm and has an optimal substructure. We can prove this by induction: Let us consider $S_1$, $S_2$, $S_3$, be the stops our greedy algorithm choses and $T_1$ , $T_2$ , $T_3$ ,…., $T_k$ be the stops in the optimal solution.

### Base Case :

Let the first stop be $S_1$ for our greedy solution. The optimal solution will either choose the station which is before $S_1$ or $S_1$ as it can anyway go till $S_1$ without getting out of gas. Thus $T_1 \leq S1$. Hence our greedy solution is true for $1^{st}$ stop.

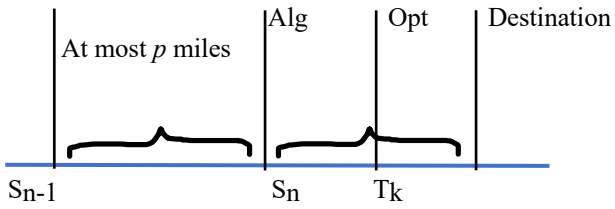
### Inductive Hypothesis :

Let us assume that our greedy solution is true for n-1 stops i.e both greedy solution and the optimal solution will choose same number of gas stations till n-1.

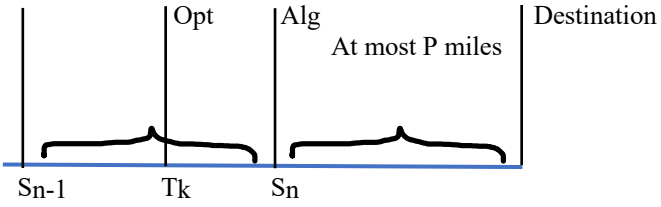
Inductive Step :

- We need to prove that our greedy solution gives the optimal result for n stops.
- Now we know that from the inductive hypothesis, our algorithm is true for $S_1$ , $S_2$ , $S_3$ ,…., $S_{n-1}$ gas stations. Now we have to consider $Sn^{th}$ gas stop.

- If suppose, the *o*ptimal solution has stopped at the gas station $T_k$ beyond $S_n$. $S_n < T_k$. Our greedy always chooses last possible stations before running out of fuel. the vehicle will run out of fuel to reach $T_k$ as it can be more than *p* miles. So, it is not possible.




In the next scenario, consider if $T_k$ comes before $S_n$. Then, it would require one more station to fill up the gas for the vehicle to reach the destination as the distance to cover will be more than *p* to reach the destination.




This proves that our greedy solution cannot be worse than the optimal solution.

The solution also has the greedy choice property. Consider there are *g* gas stations within first *p* miles. Our greedy solution will choose the $g^{th}$ station as it can travel farther most till *g* and no other station after *g* is considered. If it had chosen any other station say *k* which is less than *g* i.e $k < g$ , then it will fill less gas to make it full than if it had chosen the $g^{th}$ station and leaves with same amount of gas as it would have left from station *g*. It could have gone till $g^{th}$station either way, in that way our solution will minimise the stops.

## **Problem 3**:

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 1

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 :

Let S be the sequence of all the events and S' be the subsequence of S. Consider S consist of sequences S1 , S2 , S3 ,…., Sn and S' consist of S'1 , S'2 , S'3 ,…., S'm sequences. We choose greedy algorithm to solve this problem. Our algorithm will first check the first sequence of S' and look for it in S. If it finds the match it matches it
.It keeps continuing to find the next sequences of S' in S and so on until it has traversed till the end of both the lists.

Initialize i =0 and j = 0

While i < n and j < m

If S[i] matches with S'[j]

i = i+1

j = j+1

else
i = i+1

End While

If j == m

Output YES

Time Complexity :

The algorithm takes O(m+n) time as in the worst case, our algorithm scans through the events in both the sequences of length m and n only once.

Proof of Correctness

We can prove this through induction. Let $m_1, m_2, m_3 \ldots$ denote the events of $S'$ which are matched with S till now. Now we have to prove that $m_j \leq n_j$ for j-1…$S'$ length.
Base Case : Consider j=1. We have found the first match of $S'$ in S and our algorithm will output YES. We now have $m_1 \leq n_1$.

Inductive Hypothesis : Let's consider our algorithm has found $m_j - 1$ matches and we have $m_j\text{-}1 \leq n_j - 1$.

Inductive step : From the induction, we know that $m_j\text{-}1 \leq n_j - 1$. Now our algorithm has found match at $n_j$. And $n_j\text{-}1 < n_j$. Hence $m_j\text{-}1 \leq n_j - 1 < n_j$. As $m_j$ is the smallest index after $m_j$-1, $m_j\text{-}1 < m_j$. Combining the last two equalities, we get $m_j \leq n_j$. As it has found all the matches now, our algorithm will output "YES". Hence we have proved our claim, thus proving the correctness of the algorithm.

**Problem 4:**

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 :**

Let S be the greedy algorithm currently in use and fits $a_1, a_2, a_3, \ldots, a_n$ boxes without disrupting the order and uses up to $k_i$ trucks. Consider there exists another optimal solution $S'$ which fits these boxes up to $k_j$ trucks where $i$ and $j$ are the indexes of the trucks. We have to prove that there is no other better solution which can fit all the boxes in less trucks than our solution. i.e $i \leq j$.

Base Case : Consider there is only one truck. Our greedy algorithm will put as many boxes as it can in that truck up to maximum weight 'W' by maintaining the order of the boxes.

Inductive Hypothesis :

Assume it holds true for *k-1* trucks. If our greedy algorithm S fits boxes $a_1$ , $a_2$ , $a_3$ ,…., $a_p$ in *k-1* trucks and $S'$ fits $a_1$ , $a_2$ , $a_3$,…., $a_q$ boxes in *k-1* trucks. Consider $S'$ plans to put less boxes sometimes to allow the next trucks to be better packed. So, it might have less boxes in those *k-1* trucks than solution S. we can say that $a_q \leq a_p$.

Inductive Step:

For *k* : The greedy algorithm puts next set of boxes in $k^{th}$ truck up to maximum weight 'W'. Now solution $S'$ will have more or same number of boxes left to put in the $k^{th}$ truck as it might have had put less boxes until *k-1* trucks i.e $a_n - a_q \geq a_n - a_p$. This makes $S'$ to either fit all those boxes in $k^{th}$ truck or use one more truck *k+1* to fit the exceeding boxes. That gives us $i \leq j$. This contradicts our claim that $S'$ is optimal as it uses same or more trucks than our greedy solution. Hence we proved that our greedy solution will use minimum number of trucks to send the boxes and stays ahead of all other solutions.