CSCI 570 – Homework 2
Due: Feb 22nd, 2021

Name: Rohan Mahendra Chaudhari
USC ID: 6675-5653-85
Email ID: rmchaudh@usc.edu

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 (6 points). Prove that your algorithm maximizes the payoff (10 points) and state its running time (4 points).

Solution:

Algorithm:

a. Sort set A in nondecreasing order $a_1 >= a_2 >= \ldots.. >= a_n$
b. Sort set B in nondecreasing order $b_1 >= b_2 >= \ldots.. >= b_n$
c. Return $\prod_{i=1}^{n} a_i^{bi}$ (Pair a_i with b_i)

Proof:

Assuming the above algorithm fails to provide an optimal payoff, we consider 2 solutions S and S'.

Let S be an optimal solution where a_1 is paired with b_k and a_k is paired with b_1. Here, $a_1 > a_k$ and $b_1 > b_k$.

The other solution S' is the solution where a_1 is paired with b_1 and a_k is paired with b_k and the rest of the pairs are the same as in S.

Hence,

Payoff(S)/Payoff(S') = $(\prod_S a_i^{b_i})/(\prod_{S'} a_i^{b_i}) = ((a_1)^{b_k}(a_k)^{b_1}) / ((a_1)^{b_1}(a_k)^{b_k}) = (a_1/a_k)^{b_k - b_1}$

Since $a_1 > a_k$ and $b_1 > b_k$, then it can be inferred from the above equation that Payoff(S)/Payoff(S') < 1.

This contradicts the assumption that S is an optimal solution. Thus a_1 should be paired with b_1.

Repeating the argument for remaining elements completes the proof.

Running Time:
Assuming 2 sets A and B given above are already sorted, in which case the time complexity would be O(n).

If the 2 given sets A and B are not sorted, then we would have to sort them first. This would increase the time complexity of the algorithm to O(n logn).

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 $d_1 < d_2 < ... < d_n$ 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 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 (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:

Here, we make use of a greedy algorithm to go as far as possible before stopping for gas. Let C_i be the city with distance d_i from USC.

Algorithm:

```
A = NULL
dest = 0
for i = 1 to n:
    if (d_i – dest) > p:
            A = A U {C_i-1}
            dest = t_i-1
```

Proof:

Greedy Choice Property – Let A be an optimal solution. Suppose that it has a sequence of stops $a_1,a_2,\ldots,a_k$ where a_i is the stop corresponding to distance t_i. Suppose that b is the first stop made by the above greedy algorithm. We now show that there is an optimal solution with the first stop at b.

If a_1 = b then A is such a solution. Now suppose that a_1 != b, since the greedy algorithm stops at the latest possible city and follows the premise that s_1 is before b. We now argue that A' = $(b,a_2,a_3,\ldots..,a_k)$ is an optimal solution.

First note that |A'| = |A|

Second, we argue that A' is legal (i.e. you never run out of gas).

By definition of greedy choice you can reach b.

Finally, since A is optimal and the distance between b and a_2 is no more than the distance between a_1 and a_2, there is enough gas to get from b to a_2. The rest of A' is like A and thus legal.

Optimal Substructure Property – Let Q be the original problem with an optimal solution A. Then after stopping at the station b at distance d_i the subproblem Q' that remains is given by $d_{i+1},\ldots..,d_n$ (i.e. you start at the current city instead of USC).

Let A' be an optimal solution to Q'. Since cost(A) = cost(A')+1, clearly an optimal solution to Q includes within it an optimal solution to Q'.

Time Complexity:

Clearly the above algorithm can compute minimum stops with time complexity of O(n).

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

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 subsequence's 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:

Here,
Sequence = sequence S
Subsequence = subsequence S'

Algorithm:

```
//main function
Main():
    if isSubsequence(sequence, subsequence):
            print("S' is subsequence of S")
    else:
            print("S' is not a subsequence of S")


// function to check for subsequence in a given string
isSubsequence(sequence,sunsequence):
    n = sequence.length()
    m = subsequence.length()
    j = 0
    for i = 0 to n:
            // break if all characters of subsequence are checked
            if(j == m):
                    break;
            if(subsequence[j] == subsequence[i]):
```

```
            j++
// subsequence is found if j is equal to length of subsequence
if(j==m):
        return true;
return false;
```

My proof of correctness of optimality by induction:

Let $S=(j_1,\cdots,j_m)$ be the sequence found by our greedy solution and hence return true, and let $S'=(l_1,\cdots,l_n)$ be the sequence found by an optimal solution and also returns true.

We prove by induction that the greedy algorithm will succeed in returning true if a match was found and will ensure that $j_m >= l_n$, showing that the greedy is as optimal as the optimal solution.

Base Case
Consider the case where m=1 and n=1, then let j_1 of S and l_1 of S' be the first events found and hence $j_1 >= l_1$.

Inductive Hypothesis
Now consider the case where m>1 and n>1, and assume m-1 > S.length and n−1 < S'.length.
By the Inductive hypothesis, we have found events matching the subsequence and hence j_m-1 and l_n-1, has a match, giving m−1 >= n−1.
Knowing that the next solutions would be at j_m and l_n , thus S=S', and $j_m >= l_n$.

Hence greedy is as good as optimal.

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:

Consider the solution given by the greedy algorithm as a sequence of packages represented by indexes 1, 2, 3, ... n. Each package i has a weight, w_i, and an assigned truck t_i. $\{ t_i \}$ is a non-decreasing sequence (as the k^{th} truck is sent out before anything is placed on the $(k+1)^{st}$ truck). If $t_n = m$, that means our solution takes m trucks to send out n packages.

If the greedy solution is non-optimal, then there exists another solution $\{ t'_i \}$, with the same constraints, such that, $t'_n = m' < t_n = m$.

Consider the optimal solution that matches the greedy solution as long as possible, so for all $i < k$, $t_i = t'_i$, and $t_k \mathrel{!}= t'_k$.

There are 2 cases :

Case 1: $t_k = 1 + t'_k$

i.e. the greedy solution switched trucks before the optimal solution.

But the greedy solution only switches trucks when the current truck is full. Since $t_i = t'_i$, $i < k$, the contents of the current truck after adding the $(k – 1)^{st}$ package are identical for the greedy and the optimal solutions.

So, if the greedy solution switched trucks, that means that the truck couldn't fit the k^{th} package, so the optimal solution must switch trucks as well.

So this situation cannot arise.

Case 2: $t'_k = 1 + t_k$

i.e. the optimal solution switches trucks before the greedy solution.

Construct the sequence $\{ t''_i \}$ such that,

$t''_i = t_i$, $i <= k$ &

$t''_k = t'_i$, $i > k$

This is the same as the optimal solution, except package k has been moved from truck t'_k to truck (t'_{k-1}). Truck t'_k cannot be overpacked, since it has one less packages than it did in the optimal solution, and truck (t'_{k-1}) cannot be overpacked, since it has no more packages than it did in the greedy solution.

So $\{ t''_i \}$ must be a valid solution. If k = n, then we may have decreased the number of trucks required, which is a contradiction of the optimality of $\{ t'_i \}$. Otherwise, we did not increase the number of trucks, so we created an optimal solution that matches $\{ t_i \}$ longer than $\{ t'_i \}$ does, which is a contradiction of the definition of $\{ t'_i \}$.

So the greedy solution must be optimal.