CSCI 570 : HW-02 (Spring 2021)

February 26, 2021

Question 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 Q 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. Prove that your algorithm maximizes the payoff and state its running time.

Answer

Algorithm:

$P = \prod_{i=1}^{n} {a_i}^{b_i}$ will give the maximum payoff where a_i and b_i denote the element at i^{th} index in ascending ordered sets of A and B respectively.

Steps:

1. Sort sets A and B in ascending order. **Time Complexity:** $\mathcal{O}(nlogn)$
2. Find product of all element using $P = \prod_{i=1}^{n} {a_i}^{b_i}$. **Time Complexity:** $\mathcal{O}(n)$

Proof of Correctness

We are claiming that $P = \prod_{i=1}^{n} {a_i}^{b_i}$ where A and B are re-ordered in ascending order will maximize our payoff.

$$P = {a_1}^{b_1} \cdot {a_2}^{b_2} ..... {a_i}^{b_i} ..... {a_j}^{b_j} ..... {a_{n-1}}^{b_{n-1}} \cdot {a_n}^{b_n}$$

Suppose for the purpose of contradiction that P is not optimal, Then there is an optimal solution O with $O \neq P$

Since $O \neq P$, some pairs might be out of order. For example

$$O = {a_1}^{b_4} \cdot {a_2}^{b_7} ..... {a_i}^{b_j} ..... {a_j}^{b_i} ..... {a_{n-1}}^{b_1} \cdot {a_n}^{b_2}$$

Now we will perform following steps:

- Let pick any out of order pair of B from O and swap them while doing the product to match P. Lets call this new product of O'.

- Since B is sorted, then $b_j > b_i \implies b_j = b_i + c$ where c is a positive constant. We are ignoring the case where $b_j = b_i$, since swapping them will not make a difference.

- Since A is sorted, then $a_j > a_i$, also again we are ignoring case where $a_j = a_i$, because swapping their exponents will not make any difference.

$$\begin{aligned} {a_i}^{b_j} \cdot {a_j}^{b_i} &= {a_i}^{b_i+c} \cdot {a_j}^{b_j-c} \\ &= {a_i}^{b_i} \cdot {a_i}^{c} \cdot {a_j}^{b_j} \cdot {a_j}^{-c} \\ &= {a_i}^{b_i} \cdot {a_j}^{b_j} \cdot (\frac{a_i}{a_j})^{c} \end{aligned}$$

$$\text{Since c is positive constant and } a_i < a_j, \implies {a_i}^{b_j} \cdot {a_j}^{b_i} < {a_i}^{b_i} \cdot {a_j}^{b_j}$$

- This swap makes O' strictly better than O. Therefore O is not optimal.

- This is a Contradiction, therefore our assumption was wrong, P is actually optimal.

Time Complexity

Sorting the two sets A and B will take $\mathcal{O}(nlogn)$ time in worst case. And computing product will take linear time $\mathcal{O}(n)$. **Therefore overall time complexity would be** $\mathcal{O}(nlogn)$.

Question 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 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.

Answer

Here we are given locations of all gas stations $d_1 < d_2 < d_3... < d_n$ along the route where d_i is distance from USC. Gas tank can hold gas for atmost p miles and distance between any 2 gas station is atmost p miles. We are assuming that we starting at USC which is at distance 0 and Santa Monica is at most p miles away from last gas station.

Algorithm:

We will start with full gas tank at USC (distance 0) and will stop at the gas station which is p miles away if it exists else gas station just before that. We will make our gas tank full and repeat the same process again, till the last gas station can be reached. From our last stop we will see if distance to Santa Monica is greater than p, if so then we will make stop at last gas station too. This strategy ensures that we will stop at minimum gas station to reach Santa Monica from USC.

PsuedoCode

```
d <- list of distances of gas station from USC
stops <- initialize empty list which will indexes of gas station to stop
travelled = d[1]

for i in 2...n {
    if travelled >= p{
        append i-1 to stops
        reset travelled to d[i]-d[i-1]
    }
    else{
        add d[i]-d[i-1] to travelled
    }
}

if distance of Santa Monica from last stop is greater than p{
    append n to stops
}
```

Proof of Optimality

Suppose our algorithm (ALG) is not optimal and there exists another algorithm (OPT) which is optimal.

ALG : $i_1, i_2, ..i_r, ..i_k$ *where,* i_t *is* t^{th} *stop chosen by our algorithm*

OPT : $j_1, j_2, ..j_r, ..j_m$ *where,* j_t *is* t^{th} *stop chosen by optimal algorithm*

To show: $k = m$ or "ALG" selects same number of stops at gas stations as "OPT" does.

Part - 1

We will prove by induction that algorithm "ALG" will always choose the k^{th} gas station which is same or after the k^{th} gas station chosen by "OPT" or $i_k \geq j_k$.

Base Case : For just 1 gas station in between, we have to no option but to select that gas station if Santa Monica is more than p miles from USC. Thus base case holds.

Inductive Hypothesis : Lets assume this is true for first $r - 1$ gas stations chosen by "ALG" and "OPT" $i_{r-1} \geq j_{r-1}$.

Inductive Step : Lets proof this for first r gas stations.

Since our gas tank is full at i_{r-1} gas station, so we can go at most p miles from i_{r-1} before needing to refill. Similarly we can go atmost p miles from j_{r-1} without needing to refill.

Using inductive hypothesis and above statement, we can say that we always have option of choosing i_r gas station which lies after or at j_r, where i_r is at most p miles away from i_{r-1}, because

$$i_{r-1} \geq j_{r-1}\ (IH) \implies i_{r-1} + p \geq j_{r-1} + p \implies i_r \geq j_r$$

Here, "ALG" will always try to choose the farthest possible gas station.

Thus, using this inductive step we have shown $i_r \geq j_r$.

Part - 2

We will proof by contradiction that our algorithm "ALG" is optimal and will give minimum number of stops to reach Santa Monica.

Lets say for particular stop (r^{th}), "ALG" will give i_r gas station and "OPT" will give j_r gas station. Using Part 1 we can say $i_r \geq j_r$.

Suppose for next gas station "ALG" selects a station i_{r+1} which is p miles away from i_r. Now we have 2 options of choosing next gas station j_{r+1} for "OPT".

1. If we choose $j_{r+1} > i_{r+1}$, then this will make "OPT" incorrect as we can go only p miles without refilling. This is because $i_r \geq j_r$ which makes j_{r+1} more than p miles away form j_r.

2. If we choose $j_{r+1} < i_{r+1}$, then this will make "OPT" less or same optimal as "ALG". This is because for every r^{th} stop our algorithm "ALG" is ahead of "OPT".

Thus, we can deduce that "ALG" will never choose more number of stops than what is chosen by optimal algorithm "OPT" and hence $k = m$. **This is a contradiction and we can now say that our algorithm "ALG" is indeed an optimal algorithm.**

Time Complexity

We are here just doing a linear pass over all the gas stations distances to make decisions about stops which makes our algorithm linear in terms of n. **So we can say time-complexity of our algorithm is $\mathcal{O}(n)$**

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

buy Yahoo, buy eBay, buy Yahoo, buy Oracle

They begin with a collection of possible events (e.g., the possible transac- tions) 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 substring of S (hint: induction).

Answer

Algorithm:

Let i and j be the 2 pointers pointing to start of sequence of S (length n) and S' (length m) respectively. We will increment pointers using following rules:

Rules:

1. If S[i] = S'[j], increment i and j by 1.
2. If S[i] ≠ S'[j], increment i by 1.

If $j > m$, then our sequence S' is subsequence of S and we will output 'yes', else if $i > n$ then it is not a subsequence.

Psuedo code

```
set i and j as 1
while i <= n and j <= m {
    if S[i]==S'[j] increment i and j by 1
    else increment i by 1
}

if j > m {
    output "yes"
}

Complexity: O(m+n). Since, we increment i and j pointers atmost n+m times.
```

Proof of Correctness

Firstly lets assume our algorithm "ALG" is not correct and there exists another algorithm "OPT" which is correct.

Let above algorithms selects following indices of S to match each element of S'.

$ALG : i_1, i_2, ..i_r, ..i_m$ *where,* $i_1 < i_2 < ..i_r.. < i_m$,

$OPT : j_1, j_2, ..j_r, ..j_m$ *where,* $j_1 < j_2 < ..j_r.. < j_m$

Part - 1

We will proof by induction that algorithm "ALG" will always stays with or ahead of "OPT" in matching S'[index] wih element in S. To Proof : $i_r \leq j_r$.

Base Case: Lets consider base case where sequence S' is of length 1 (m=n=1),

Here, $i_1 \leq j_1$, If S' is a subsequence of S then "ALG" will try to match that element (S'[1]) with first available same element in S. Thus, base case holds.

Inductive Hypothesis: Lets suppose for $(r-1)^{th}$ element in S', $i_{r-1} \leq j_{r-1}$

Inductive Step: Lets proof this for r^{th} element.

Since our inductive hypothesis says algorithm "ALG" will match $(r-1)^{th}$ element is S' with a element in S, at the same index or before the optimal algorithm "OPT" does. Thus, it will be true for r^{th} element also because of the nature of "ALG" which will always try to match current element of S' with next first available match in S.

Thus, by this inductive step we can show $i_r \leq j_r$ and hence our algorihtm "ALG" will always stays with or ahead of optimal algorithm in matching elements from subsequence S'.

Part - 2

We will use contradiction to show that our algorithm is correct and will tell if S' is a subsequence of S if it so.

Lets say for particular index (r+1) of S', our algorithm "ALG" fails to find any match in S (length n) and hence $i_{r+1} > n$ but our optimal algorithm does find a match $j_{r+1} \leq n$ because it is correct.

But using Part 1, we can say that this is not possible because $i_r \leq j_r$ and if "OPT" succeeded in finding a match of S'[r+1] then there exists a match in sequence S after index j_r for S'[r+1]. Since our "ALG" greedily searches for next match, it should also succeed in finding that match. **This is a Contradiction which shows our initial assumption was wrong and our algorithm "ALG" is indeed a correct algorithm and will tell if S' is a subsequence of S**.

If S' is not a subsequence of S, then suppose upto particular index (r-1), our algorithm "ALG" was able to find match and after that there is no match with S. This will make $i_r > n$ and here algorithm will say S' is not a subsequence of S.

Question 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.

Answer

We are given set of boxes where i^{th} box weighs w_i, these boxes has to sent in n number of trucks of each having capacity W. Here company is trying to minimize number of trucks (n) with a constraint that boxes should be sent in order they arrive.

We have to prove that their current greedy strategy is optimal and will minimize the number of trucks.

Proof of Optimality

Lets suppose their current greedy strategy "ALG" is not optimal and their exists a strategy "OPT" which allows them to send less number of trucks.

$ALG : A_0^0, A_0^1, A_0^2, A_1^3, ..., A_i^r, ...A_k^n, \quad A_i^r$ *denote* r^{th} *box is sent in* i^{th} *truck* (A_i) *by greedy strategy.*

$OPT : O_0^0, O_0^1, O_0^2, O_1^3, ..., O_j^r, ...O_m^n, \quad O_j^r$ *denote* r^{th} *box is sent in* j^{th} *truck* (O_j) *by optimal strategy.*

To Proof: $k \leq m$ or "ALG" uses less or equal number of trucks as compared to "OPT"

Part 1:

We will prove by induction that for r^{th} box, if ALG sends that box in truck A_i and OPT sends that box in truck O_j, then $i \leq j$.

Base Case: For only one box ($n = 1$), both ALG and OPT send that in first truck. Hence base case holds.

Inductive Hypothesis: Lets assume for box $(r-1)$ where $r > 2$, ALG sends $(r-1)^{th}$ box in A_{i-1} and OPT sends it in O_{j-1}, where $(i-1) \leq (j-1)$.

Inductive Step: Lets consider for r^{th} box.

Their greedy approach "ALG" will try to put r^{th} box in A_{i-1} if it can be fitted in $(i-1)^{th}$ truck else that box will go in i^{th} truck (A_i).

Optimal algorithm also have 2 options either to put r^{th} box in O_{j-1} or O_j.

In any case, using inductive hypothesis we can see for r^{th} box, ALG will try to put that box in same truck as OPT does or any truck before that because it is always looking to make current truck more packed if possible. So, we can conclude that "ALG" will always "stays ahead" or "stays with" "OPT" in sending boxes. Thus $i \leq j$.

Part 2:

We will now use contradiction to prove that greedy strategy "ALG" is optimal and will minimize the number of trucks.

Lets says for particular r^{th} box, using strategy "ALG" we will put that box in i^{th} truck (A_i) and "OPT" will put that box in j^{th} truck (O_j).

Now, lets see possibilities for next box i.e. $(r+1)^{th}$ box, greedy algorithm "ALG" will try to put that in current truck (A_i) if possible else in next truck A_{i+1}.

Now for algorithm "OPT", lets consider all that cases that $(r+1)^{th}$ box could go.

1. In $(j-1)^{th}$ truck (O_{j-1}), it is not possible as it will make the boxes r and $(r+1)$ arrive out of order and make the algorithm incorrect.

2. In j^{th} truck (O_j), if there is enough space it can go in this truck, nevertheless using **Part 1**, we can say it will be less than or equal to truck number selected by "ALG".

3. In $(j+1)^{th}$ truck (O_{j+1}), it can go in this truck just to make this or any next truck to be packed better, however this may cause the algorithm "OPT" to select more number of trucks.

Using above 3 cases we can see there is no chance that "OPT" uses less number of trucks however it is possible for it to use more number of trucks which will make "OPT" not optimal. **This is a contradiction, hence our assumption was wrong and we can conclude that greedy strategy "ALG" that is currently in use is actually an optimal strategy.**