CMSC 27200 - Winter 2021 — Andrew Drucker, Lorenzo Orecchia

due tuesday, january 26 at 12 noon via gradescope

H2.

- The assignment is due at Gradescope on Tuesday, January 26 at 12 noon.
- You can either type your homework using LaTex or scan your handwritten work. We will provide a LaTex template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to study and discuss the problems with 2 other students (per problem; any section). However, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.

Lewis Arnsten, Collaborator: Brian McManus

Problem 1: Exercise 3 in in Chapter 4 in the Kleinberg-Tardos textbook

We know each box X_i has a weight w_i and the maximum weight a truck can carry is W. The current greedy algorithm involves packing boxes in the order that they arrive (X_1,....,X_n) and sending off a truck when it cannot fit any more boxes (total weight of boxes < W). Thus, to preserve the order of arrivals, if box X_i is assigned to truck x and box X_j is assigned to truck y where x < y, then box X_i arrived at the trucking station before box X_j (i < j).

Proof that this greedy algorithm minimizes the number of trucks that are needed:

Consider another solution that fits X_1,....,X_i boxes into the first k trucks (out of trucks 1,...,N). The original greedy algorithm fits X_1,....,X_j boxes into the first k trucks. In order for the original greedy algorithm to stay ahead of all other solutions $i \leq j$. To prove that it does this we will use induction. The base case (k = 1) is proved as the greedy algorithm will fit as many boxes as possible into the first truck. Assume true for the k case: the original greedy algorithm fits n boxes in the first k trucks and the other solution fits $m \leq n$ boxes. For k + 1 trucks, for the first k trucks the two solutions fit X_1,....,X_n and X_1,....,X_m boxes. Then for the k + 1 truck, the other solution fits X_{m+1},....,X_i boxes and the original greedy algorithm fits X_{n+1},....,X_i boxes. Since $m \leq n$, $X_{m+1} \leq X_{n+1}$, and X_{m+1},....,X_i contains more boxes than X_{n+1},....,X_i, meaning the original greedy algorithm can fit at least as many boxes in k + 1 trucks as the other solution, and possibly more.

The algorithm runs in polynomial time as it is simply a loop, updating the input set of trucks when the current truck cannot carry more weight. The algorithm is O(n).

Problem 2: Solve exercise 5 in Chapter 4 in the Kleinberg-Tardos textbook

Start on one end of the road and move towards the other. Assuming we choose to start on the eastern end of the road, we would move west until the first house X is exactly 4 miles east. This point (A_0), is the furthest a base station can be placed such that it will cover all the houses from the starting point. Place the next station (A_1) as far as possible from A_0, ensuring all the houses between the two towers are covered. The algorithm is continued by placing the $i+1^{th}$ station as far as possible from the i^{th} station such that all the houses in between are covered.

Proof that this greedy algorithm is the optimal solution:

Consider the optimal solution where $T = \{T_1,....,T_n\}$ is the set of base stations. Let $B = \{B_1,....,B_m\}$ be the set of base stations produced by the greedy algorithm. To show that the greedy algorithm uses the fewest possible base stations we will use induction. Our claim is that B is the best possible solution, meaning $B_i \geq T_i$ for all i's. The base case is true (i = 1) because we start on one end of the road, so T_1 cannot be greater than B_1. Assume the claim is true for value k: all the houses covered by base stations $\{T_1,....,T_k\}$ are necessarily covered by base stations $\{B_1,....,B_k\}$ ($B_k \geq T_k$). Since our greedy algorithm maximizes the area between house and base station, for i+1 steps, $B_{i+1} \geq T_{i+1}$ and $B_i \geq T_i$ (by assumption) for all i's, proving our claim. If $B_{i+1} \leq T_{i+1}$ the set T would have to leave a house uncovered, for T_i cannot cover a house that B_i does not cover.

If the optimal solution used fewer stations than our greedy algorithm, $n < m$, then $\{B_1,....,B_m\}$ would not cover every house. However, since $B_n \geq T_n$ (proved by induction) $\{T_1,....,T_n\}$ would also not cover all the houses. This is impossible for the optimal solution, thus it is not possible for $n < m$. $n=m$, so the greedy algorithm is the optimal solution.

This algorithm runs in polynomial time because it will at most place n stations, where n is the number of houses. The algorithm is O(n).

PROBLEM 3 (25 POINTS) *The Running Sums problem is defined as follows:*

Input: *a sequence* $(a_1, \ldots, a_n)$*, where each* a_i *is either* 1 *or* -1*.*

Desired output: *a sequence* $(b_1, \ldots, b_n)$*, where each* b_i *is either* 0 *or* 1*.*
Your goal is to ***minimize*** $\sum_{1 \le i \le n} b_i$*, subject to the* ***constraint*** *that*

$$\sum_{1 \le i \le j} (a_i + b_i) \;\ge\; 0\,, \qquad \text{for all} \quad j \in \{1, 2, \ldots, n\}\,.$$

Give an algorithm for this problem that, for full credit, should run in $O(n)$ *steps.*

Solution: Your solution goes here.

Problem 3: Running sums problem

The greedy algorithm for this problem must keep running sums of a_i and b_i. We will keep track of these sums using a variable S, initialized to 0. Our algorithm will take the first value of $\{a_1,....,a_i\}$, add it to our running sums variable S, and then set b_1 to 1 if $S < 0$ and 0 if $S \geq 0$. Lastly, the new value b_1 will be added to S and a_1 will be removed from set a. To find b_i this process will be repeated until $\{a_1,....,a_i\}$ is empty, updating S with every iteration.

We will use induction to prove this algorithm yields the optimal solution:

For the base case i=1, S=0. If $a_1 = 1$ then S = 1, $b_1 = 0$ and S is set to $a_1 + b_1 = 1 \geq 0$. If $a_1 = -1$ then S = -1, $b_1 = 1$ and S is set to $a_1 + b_1 = 0 \geq 0$. Now assume $\sum_{i=1}^{k} a + b \geq 0$, which means our running sums variable $S \geq 0$ up to some index k. Considering k+1, if $a_{k+1}=1$ then $S=a_{k+1}+ S \geq 0$ and $b_{k+1}=0$. If $a_{k+1}=-1$ then if $S=a_{k+1}+ S< 0$ we choose $b_{k+1}=1$ so $a_{k+1}+ b_{k+1}= 0$. Since we know $\sum_{i=1}^{k} a + b \geq 0$ and $a_{k+1}+ b_{k+1}= 0$ the constraint has been followed.

Now consider the optimal sequence n_i. Our claim is that b_i and n_i are equal sets. For the base case i=1, $b_1 = n_1$ because if $a_1 = -1$ b_1 and n_1 must be 1 and if $a_1 = 1$ $b_1=n_1=0$. We assume that for any value $i \leq k$, $b_i=n_i$. Considering k+1, we know $\sum_{i=1}^{k} a + b = \sum_{i=1}^{k} a + n$. Additionally, we know $b_{k+1}=n_{k+1}$ because when $\sum_{i=1}^{k} a + b = 0$ and $a_{k+1}= 1$ or $\sum_{i=1}^{k} a + b > 0$ $b_{k+1}=n_{+1}=0$. If $\sum_{i=1}^{k} a + b = 0$ and $a_{k+1}= -1$ or $\sum_{i=1}^{k} a + b < 0$ then $b_{k+1}=n_{+1}=1$ or the constraint is violated. Thus, the greedy algorithm is correct and produces the optimal solution.

This algorithm consists of a for loop that will end after k iterations. In other words it is a while loop where k is modified with addition after each iteration, thus the algorithm is O(n).

PROBLEM 4 (25 POINTS) *In the Hopping Game, there is a sequence of n spaces. You begin at space 0 and at each step, you can hop 1,2,3, or 4 spaces forward. However, some of the spaces have obstacles and if you land on an obstacle, you lose.*

Give a greedy algorithm which, given an array $A[1, \ldots, n-1]$ of Boolean values with $A[i]$ indicating the presence/absence of obstacle at position $i \in [1, n-1]$, find the minimum number of hops needed to reach space n without losing, if it is possible to do so. (We assume spaces 0 and n are obstacle-free, and are not part of the input.) Prove that your algorithm is correct. For full credit, your algorithm should run in time $O(n)$.

Problem 4: Hopping game

This greedy algorithm consists of hopping forward the maximum number of spaces i (i between 1 and 4) where A[i] is not an obstacle. By repeating this action over the entire obstacle set O = $\{O_1,....,O_n\}$ and counting each hop you will find the minimum number of hops needed to reach space n.

Here is this greedy algorithm as code:

```
hops = 0
l = length of A
i = 0
while i < l:
    if l - i <= 4:
        hops = hops + 1
        return hops

    if A[i + 4]:
        if A[i + 3]:
            if A[i + 2]:
                if A[i + 1]:
                    return 'Lose'
                else:
                    i=i+1
            else:
                i=i+2
        else:
            i=i+3
    else:
        i=i+4

    hops = hops + 1
```

This algorithm will always return a correct value, or it will terminate early if there are four obstacles in a row. To prove correctness we will use induction and compare this greedy algorithm to the optimal solution. Consider the solution X_i produced by this algorithm, consisting of values between 1-4 for each hop. Consider also the optimal solution H_i. Our claim is that $X_i=H_i$ for all i's. The base case is true, for when i=0, $X_i=H_i=0$. Now we assume that $X_j=H_j$ for a position j in the solution sequence. To calculate the next hop (j+1), the algorithm will check if there is an open space 4 spots ahead, if not it looks 3 spots ahead, and so on. Thus, X_{j+1} is the largest value i such that A[prev_spot + i] is false. It is impossible for $X_{j+1}< H_{j+1}$ as H_{j+1} would not be valid. Therefore for all j's $X_j=H_j$ and $X_{j+1}\geq H_{j+1}$ and we have found the optimal solution. This solution is also correct, as the final hop is added when the index i is within 4 of the length of the obstacle set. Since the final spot of the set is always open, when there are less than 5 spots left there is guaranteed to be exactly one valid hop left.

This algorithm consists of a while loop where i is modified once per iteration and the loop is executed less than L (length of A) times. Thus the algorithm is O(n).