H2.BRIANMCMANUS

- 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, inyour 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.

I talked with Lewis Arnsten about the problems

PROBLEM 1 (25 POINTS) *Solve exercise 3 in Chapter 4 in the Kleinberg-Tardos textbook. (trucking)*

Solution: Say n boxes arrive in the order $b_1, \ldots, b_n$. To pack the boxes into N trucks preserving the order is to assign each box to one of the trucks $1, \ldots, N$ such that no truck is overloaded and the order is preserved. We will prove that the greedy algorithm stays ahead of the ideal solution. We will prove the claim through induction.

Let $k =$ enumerate the number of trucks used. We want to show that the greedy algorithm packs at least as many boxes after k trucks as the optimal algorithm. The base case of $k = 1$ is clear, as the greedy algorithm fits as many boxes as possible. If we assume that the greedy algorithm is at least as good as the best algorithm after k trucks, that is the greedy algorithm has packed i boxes and the best algorithm j and $i \geq j$. For the $k+1$ truck the best solution packs in $b_{j+1}, \ldots, b_k$, and because $i \geq j$, the greedy algorithm is able to fit at least $b_{i+1}, \ldots, b_k$ boxes, and potentially more, so the greedy algorithm is at least as good, and the claim of optimality is proven by induction.

PROBLEM 2 (25 POINTS) *Solve exercise 5 in Chapter 4 in the Kleinberg-Tardos textbook. (cell phone towers)Please note! Here and elsewhere, when the authors (or your instructors) say "given algorithm" without further instructions, they mean "give an algorithm AND prove that it is correct and runs in polynomial time". This should be assumed henceforth.*

Solution:

Define the position of each house on the road as the number of miles from the eastern end. Consider a greedy algorithm in which We will place the first base station at the largest position such that all houses between 0 and this point are covered by the the tower, which we will denote g_1. To place the next tower repeat the process, placing the $i+1$ tower at the largest position such that all houses between it and g_i are covered. This generates a set $\{g_1, \ldots, g_n\}$.

Now we will compare the greedy solution to the best solution, denoted $\{b_1, \ldots, b_m\}$. If the greedy solution is optimal then $m = n$. To show this we will use the same technique as the previous problem. Claim: $\forall i, g_i \geq b_i$. The claim is true for $i = 1$, because the first station is placed as far east as possible. Next assume that the claim holds for some j, that is $g_j \geq b_j$. If we consider the next station, b_{j+1} cannot cover a house that g_{j+1} does not cover, as we choose g_{j+1} to have the largest possible position. If the position of b_{j+1} was larger than the position of g_{j+1} there would necessarily be an uncovered house by choice of b_{j+1}. Therefore $g_{j+1} \geq b_{j+1}$, and the claim is proven by induction.

To conclude, if the greedy solution used more stations than the best solution (i.e. $n > m$) then $\{g_1, \ldots, g_m\}$ would fail to cover every house, but from above $g_m \geq b_m$ and the best solution would also fail to cover a house.

This algorithm runs in polynomial time because for n houses each station covers at minimum 1 house, and so you will only every have to place n stations, and thus 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:

We will approach this with a greedy algorithm. We will use the i, a_i, b_i from the problem, and introduce a counter c.

---

initially $i = 1, c = 0$
while $i \le n$ **do**
  if $a_i + c < 0$ **then**
    let $b_i = 1$
  else
    let $b_i = 0$
  end if
  $C = C + a_i + b_i$
  $i = i + 1$
end while
return $b_1, \ldots, b_n$

To prove correctness we first need to consider the constraint. We will show by induction that the constraint is never violated. For the base case consider $i = 1$. As $c = 0, a_1 + c = a_1$. If $a_1 = -1$ then $b_1 = 1$ and $a_1 + b_1 = 0 \ge 0$. If $a_1 = 1, b_1 = 0 \implies a_1 + b_1 = 1 \ge 0$. Next assume that $\sum_{i=1}^{j} a_i + b_i \ge 0$ for some j. Then clearly $C = \sum_{i=1}^{j} a_i + b_i$ as it counts $(a_i + b_i)$ for each $i \le j$. Next consider $j + 1$. If $a_j = 1$ clearly $a_j + C \ge 0$, and if $a_j = -1$ then we choose $b_j = 1 \implies a_j + b_j = 0 \implies C \ge 0$ and therefore the constraint is not violated.

Now we will compare b_i to the optimal sequence B_i, and claim that $\forall i, b_i = B_i$, and again we will work by induction. For $i = 1, B_1 = b_1$, because if $a_1 = -1$ then the only valid choice is $B_1 = b_1 = 1$, and in the case that $a_i = 1$ then $B_1 = b_1 = 0$ clearly. Next assume that $\forall i \le j, b_i = B_i$. Then consider the case of $j + 1$. Clearly $\sum_{i=1}^{j} a_i + b_i = \sum_{i=1}^{j} a_i + B_i$. If $\sum_{i=1}^{j} a_i + b_i > 0$ then $B_{j+1} = b_{j+1} = 0$, as it is clearly optimal for B_{j+1} and our algorithm chooses such a b_{j+1}. If $\sum_{i=1}^{j} a_i + b_i = 0$ but $a_{j+1} = 1$ then also $B_{j+1} = b_{j+1} = 0$. If $\sum_{i=1}^{j} a_i + b_i = 0$ but $a_{j+1} = -1$ then we choose $b_{j+1} = 1$ but $B_{j+1} = 1$ also or else it violates the constraint. Therefore the algorithm produces the optimal solution.

For complexity, for each loop of the while loop i is only modified to be $i = i + 1$, and therefore the loop will terminate after i iterations, and so the complexity 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 $O(n)$.*

Solution: I will begin by describing a greedy algorithm. i, A are as in the problem description, and c counts the number of hops.

```
initially i = 0, n = 0
while i < n − 4 do
    if A[i + 4] then
        if A[i + 3] then
            if A[i + 2] then
                if A[i + 1] then
                    return "Cannot Win"
                else
                    i = i + 1
                end if
            else
                i = i + 2
            end if
        else
            i = i + 3
        end if
    else
        i = i + 4
    end if
    c = c + 1
end while
c = c + 1
return c
```

For a proof of correctness we have to consider 3 failure modes: the algorithm fails when it shouldn't, and it returns when it shouldn't, and it returns the wrong value.

For the first failure the only way that the algorithm terminates early is when $A[i+1],\ldots,A[i+4]$ are all true, which is the case when there are 4 obstacles in a row, which is impossible to pass.

For the second failure, the only times when a failure should occur is when there are 4 obstacles in a row, which is already accounted for.

For the third failure, we use induction to compare this solution to the optimal solution. Consider the sequence a_j which is all the values that i takes in the algorithm and compare it to b_j which is the values that i takes in the best algorithm. The claim is that $\forall j, a_j = b_j$. Consider $j = 0$. Clearly $a_j = b_j = 0$. Next assume that $a_k = b_k$ for some position k in the sequence of is, and then we will calculate the next hop. The algorithm first checks if $i+4$ is valid, and then failing that it checks $i+3$ and so on. This guarantees that a_{k+1} is the largest i such that $A[i] == False$, and thus it is impossible for $a_{k+1} < b_{k+1}$ as any such b_{k+1} would not be a valid hop. The while loop terminates early to avoid out of bounds, and the last hop is handled separately. If $i \geq n-4$ then there is always 1 hop left as the final space never has an obstacle and so can always be jumped to, so that is the source if the final $c = c+1$.

For complexity note that in each iteration of the while loop i is modified only once, and in the worst case i is incremented by 1 so the loop iterates as many as $n-4$ times, yielding a complexity of $O(n)$.