601.433/633 Introduction to Algorithms **Fall 2024**
Homework #5 **Due:** October 15, 2024, 9am

Remember: you may work in groups of up to three people, but must write up your solution entirely on your own. Collaboration is limited to discussing the problems – you may not look at, compare, reuse, etc. any text from anyone else in the class. Please include your list of collaborators on the first page of your submission. You may use the internet to look up formulas, definitions, etc., but may not simply look up the answers online.

Please include proofs with all of your answers, unless stated otherwise.

1 Submatrices (33 points)

Let $A \in \{0,1\}^{n \times m}$ be a matrix with n rows, m columns, and where every entry is either 0 or 1. We will let A_{ij} denote the entry in row i and column j, so for example A_{11} is the top-left entry, A_{n1} is the bottom-left entry, A_{1m} is the top-right entry, and A_{nm} is the bottom-right entry. Suppose that we want to find the largest integer k such that A contains a $k \times k$ contiguous submatrix consisting of all 0's. In other words, we want to find the largest k such there exist values i, j such that $A_{xy} = 0$ for all $i - k < x \leq i$ and $j - k < y \leq j$.

We will design a dynamic programming algorithm that runs in $O(nm)$ time for this problem.

(a) (17 points) For every $i, j \in \mathbb{N}$ with $1 \leq i \leq n$ and $1 \leq j \leq m$, let $S(i,j)$ denote the maximum value of k such that there is a $k \times k$ contiguous submatrix of A consisting of all 0's whose bottom-right corner is at (i,j) (row i, column j). Write a recursive formula for $S(i,j)$, and prove that your formula is correct.

Note: you will need to use this formula in the next part to get an $O(nm)$-time algorithm, so make sure that your formula is not too big/slow.

(b) (16 points) Give a dynamic programming algorithm based on your solution to part (a), and prove that it correctly finds the largest possible value of k and runs in time $O(nm)$.

2 Completing Homeworks (33 points)

Suppose (hypothetically) that you were taking a class, possibly called "Introduction to Algorithms", in which the homeworks were extremely difficult. After enough complaining, the professor decided to make the following changes. There are two homework assignments each week rather than one, an "easy" assignment and a "hard" assignment. The hard assignment is worth more points, but it is in fact so difficult that you can only complete it if you're completely rested and prepared, meaning that you cannot have done either of the assignments the week before.

More formally, let n be the number of weeks in the class, let h_i be the number of points for the hard assignment in week i, and let e_i be the number of points for the easy assignment in week i. Note that h_i does not have to be equal to h_j for $i \neq j$ (although it might be), and similarly with e_i and e_j. Assume that you know all of these values in advance. Then the goal is compute a schedule which in each week tells you whether to do nothing, the easy assignment, or the hard assignment and maximizes the total number of points, subject to the restriction that if you do a hard assignment in week i you cannot have done *any* assignment in week $i - 1$.

(a) (11 points) One obvious approach would be to choose a hard assignment in week i if we get more points than if we completed the easy assignments for weeks i and $i-1$. This would be the following algorithm:

```
$i = 1$
while($i < n$) {
   if($h_{i+1} \geq e_{i+1} + e_i$) {
      choose no assignment in week $i$,
      choose the hard assignment in week $i+1$,
      $i = i + 2$
   }
   else {
      choose the easy assignment in week $i$,
      $i = i + 1$
   }
}
```

Give an instance in which this algorithm does not return the optimal solution. Also say what the optimal solution is (and its value) and what the algorithm finds instead.

(b) (22 points) Give an algorithm with $O(n)$ worst case running time which takes as input the values $e_1, \ldots, e_n$ and $h_1, \ldots, h_n$ and returns the value of the optimal schedule. Prove its correctness and running time.

3 Mobile Business (34 points)

Let's say that you have a great idea for a new food truck, and in order to save money you decide to run it out of your RV so you can live where you work. Each day i there is some demand for your food in Baltimore and some demand in Washington – let's say you would make B_i dollars by being in Baltimore and W_i dollars by being in Washington. However, if you wake up in one city (due to being there the previous day) and want to serve in the other city, it costs you M dollars to drive there.

The goal in this problem is to devise a maximum-profit schedule. A schedule is simply an assignment of locations to days – for each day i, the schedule says whether to serve in Baltimore or Washington. The profit of a schedule is the total profit you make, minus M times the number of times you have to move between cities. For the starting case, you can assume that on day 1 you wake up in Baltimore.

For example, let $M = 10$ and suppose that $B_1 = 1, B_2 = 3, B_3 = 20, B_4 = 30$ and $W_1 = 50, W_2 = 20, W_3 = 2, W_4 = 4$. Then the profit of the schedule ⟨Washington, Washington, Baltimore, Baltimore⟩ would be $W_1 + W_2 + B_3 + B_4 - 2M = 100$, where one of the M's comes from driving from Baltimore to Washington on day 1, and the other comes from driving from Washington to Baltimore on day 3. The profit of the schedule ⟨Washington, Baltimore, Baltimore, Washington⟩ would be $W_1 + B_2 + B_3 + W_4 - 3M = 50 + 3 + 20 + 4 - 30 = 47$.

Given the fixed driving cost M and profits $B_1, \ldots B_n$ and $W_1, \ldots, W_n$, devise an algorithm that runs in $O(n)$ time and computes the profit of an optimal schedule. As always, prove correctness and running time.