601.433/633 Introduction to Algorithms **Fall 2025**
Homework #4 **Due:** Nov 3, 2025, 11:59pm

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 Word Segmentation (33 points)

A number of languages, both ancient and modern, are written without spaces between the words. In these languages, an important problem is word segmentation: given a string of characters, divide the string into consecutive words. In English, the analogous problem would consist of taking a string like "meetateight" and deciding that the best segmentation is "meet at eight" (and not "me et at eight" or "meet ate ight", etc.).

A natural approach to this problem is to find a segmentation that maximizes the cumulative "quality" of its individual constituent words. Thus, suppose you are given a black box that, for any string of letters $x = x_1x_2x_3 \dots x_k$, will return a positive number $quality(x)$. Larger quality values correspond to more plausible words. (So $quality(\text{“}me\text{”})$ would be large, while $quality(\text{“}ght\text{”})$ would be small.) By convention, we will set the quality of the empty string to be 0, i.e., $quality(\text{“”}) = 0$.

Given a long string of letters $y = y_1y_2 \dots y_n$, a segmentation of y is a partition of its letters into contiguous blocks of letters; each block corresponds to a word in the segmentation. The *total quality* of a segmentation is the sum of the qualities of the blocks.

Give an algorithm that takes a string y and computes the maximum total quality of any segmentation (i.e., return the largest number α such that there is a segmentation of total quality α and no segmentation has total quality larger than α). Your algorithm should run in $O(n^2)$ time. As always, prove correctness and running time. You can treat a single call to the black box computing $quality(x)$ as a single computational step, no matter how long x is.

2 Submatrices (34 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) (17 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 $O(nm)$.

3 Reachable nodes (33 points)

Let G be a directed graph represented by an adjacency list. Suppose each node u has a weight $w(u)$, which might be different for each node. Give an algorithm that computes, for every node u, the maximum weight of any node that is reachable from u. So, for example, if G is strongly connected then every node can reach every other node, so for every node the maximum reachable weight is the same (the largest weight of any node in the graph).

More formally, for each vertex u let $R(u)$ denote the vertices reachable from u. Then when your algorithm is run on G, it should return an array of values where the value for node u is $\max_{v \in R(u)} w(v)$.

Your algorithm should run in $O(m + n)$ time. Prove correctness and running time.