CSCE629 Analysis of Algorithms

Homework 5

Texas A&M U, Fall 2019 *09/27/19*

Lecturer: Fang Song *Due: 10am, 10/04/19*

Instructions.

- Typeset your submission by LaTeX, and submit in PDF format. Your solutions will be graded on *correctness* and *clarity*. You should only submit work that you believe to be correct, and you will get significantly more partial credit if you clearly identify the gap(s) in your solution. You may opt for the "I'll take 15%" option (details in Syllabus).
- You may collaborate with others on this problem set. However, you must **write up your own solutions** and **list your collaborators and any external sources** for each problem. Be ready to explain your solutions orally to a course staff if asked.
- For problems that require you to provide an algorithm, you must give a precise description of the algorithm, together with a proof of correctness and an analysis of its running time. You may use algorithms from class as subroutines. You may also use any facts that we proved in class or from the book.

This assignment contains 3 questions, 4 pages for the total of 40 points and 10 bonus points. A random subset of the problems will be graded.

1. (20 points) (Longest forward-backward contiguous substring) Describe and analyze an efficient algorithm to find the length of the longest *contiguous* substring that appears both *forward* and *backward* in an input string $T[1, \ldots, n]$. The forward and backward substrings must NOT overlap. Here are several examples.

   - Given the input string `ALGORITHM`, your algorithm should return 0.
   - Given the input string `RECURSION`, your algorithm should return 1, for the substring `R`.
   - Given the input string `REDIVIDE`, your algorithm should return 3, for the substring `EDI`. (The forward and backward substrings must not overlap!)
   - Given the input string `DYNAMICPROGRAMMINGMANYTIMES`, your algorithm should return 4, for the substring `YNAM`. (It should not return 6, for the subsequence `YNAMIR`, because it's not contiguous.).

2. (20 points) (Word segmentation) If English were written without spaces, then we need to infer likely boundaries between consecutive words in the text. This is called word segmentation. For example, given `meetateight`, you can probably decide that the best segmentation is `meet␣at␣eight`. (and not `me␣et␣at␣eight`, or `meet␣ate␣ight`. This is all the more relevant in languages like Chinese and Japanese, which are written without spaces between the words.

   How could we automate this process? A simple approach that is at least reasonably effective 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 $x = x_1x_2 \dots x_n$, will return a number $quality(x)$. This number can be either positive or negative; larger numbers correspond to more plausible English words. (So $quality(\texttt{me})$ would be positive, while $quality(\texttt{ght})$ would be negative.)

   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 determined by adding up the qualities of each of its blocks. (So we'd get the right answer above provided that $quality(\texttt{meet}) + quality(\texttt{at}) + quality(\texttt{eight})$ was greater than the total quality of any other segmentation of the string.)

   Descrive and analyze an efficient algorithm that takes a string y and computes a segmentation of maximum total quality. (You can treat a single call to the black box computing $quality()$ as a single computational step.)

3. (10 points (bonus)) (Greedy matrix-chain multiplication?) As suggested in class, one approach to choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} \ldots A_j$ is by selecting k to minimize the quantity $d_{i-1} d_k d_j$. Does this always give an optimal solution. Prove it or give a counterexample (i.e., an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.)