CSE 6140 Assignment 2
due Oct 1st, 2017 at 6pm on T-Square

September 16, 2017

Please upload:

1) a PDF cover letter indicating with whom you worked (if applicable), the sources you used, and if you wish, your impressions about the assignment (what was fun, what was difficult, why...)

2) a PDF with your solutions to all the problems

Each file name should start by `<GTusername>_HW2`.

Please type your answers in LaTeX. You may handwrite them if you wish, but if we cannot read your handwriting, you will not receive points for your answer.

1 Scheduling with Weights

The deadline for homework 1 is upon us. The TA sits in front of the computer, with a mountain of emails from students with doubts. Say we have a set of n emails to answer, where each email j requires time t_j to answer, and has an importance w_j that represents how urgent it is (higher weight means more urgent). We want to find an order to schedule the replies that gives precedence to writing critical emails sooner. Formally, let C_j denote the completion time of email j. For example, if the first three emails to write are i, j, and k (in that order), we would have that $C_i = t_i$, $C_j = t_i + t_j$, and $C_k = t_i + t_j + t_k$. Our goal is to find an order to schedule the replies that minimizes $\sum_{i=1}^{n} w_i C_i$.

As it turns out, this problem has a similar solution to the *Minimizing Max Lateness* problem we solved in class. Specifically, if we sort the jobs based on the correct criteria, our schedule will be optimal. For each of the following criteria, either give a small example showing it does not always produce an optimal schedule, or give a proof of correctness based on the exchange argument (following the framework we used for minimizing max lateness, i.e. show that swapping two adjacent out-of-order emails in a schedule only improves the objective).

a) Greedy by smallest time t_i first.

b) Greedy by largest weight w_i first.

c) Greedy by largest weight-per-unit-time $\frac{w_i}{t_i}$ first (break ties by index of emails, i.e. if two emails have the same ratio then choose the one with smaller index).

2 Divide and Conquer

Let T be a table of n relative integers. We want to find the maximum sum of contiguous elements, namely, two indices i and j ($1 \leq i \leq j \leq n$) that maximize $\sum_{k=i}^{j} T[k]$.

a) If the values in the table are $T[1] = 2, T[2] = 18, T[3] = -22, T[4] = 20, T[5] = 8, T[6] = -6, T[7] = 10, T[8] = -24, T[9] = 13$, and $T[10] = 3$, can you return the two indices and the corresponding optimal sum?

b) Design an algorithm that return the maximum sum of contiguous elements with a divide-and-conquer algorithm.

c) Bonus: Design a linear-time algorithm that solves the problem through a single scan of the array.

3 Master Theorem

For each of the following recurrences, give the asymptotic solution if the recurrence can be solved with Master Theorem. Otherwise, briefly explain why Master Theorem is not applicable.

a) $T(n) = 49T(\frac{n}{7}) + n^2 + 3n$

b) $T(n) = \frac{1}{4}T(\frac{n}{9}) + n$

c) $T(n) = 4T(\frac{n}{2}) - n^2 - 2n$

d) $T(n) = 2T(\frac{n}{4}) + 1000n^{0.6}$

e) $T(n) = 3T(\frac{n}{2}) + \ln 2$

4 Dynamic Programming

Barry Cage is developing a new Search engine, Boogle, and beta testing has shown that many users hastily enter queries that do not have spaces between words, like “herecomesthesun” and “javatutorials”. So, Barry needs your help in writing an efficient algorithm to perform string segmentation. Now there are several possible segmentations of the string “herecomesthesun”, such as “here comes the sun” or “her eco me st hesun”. One possible solution would be to maximize the cumulative quality/plausibility of the individual segmented words.

Thanks to some open source code, he already has access to a function that computes the plausibility of words. Given a string of letters $x = x_1x_2...x_k$, *plausibility*(x) returns a number that can be either positive, or negative; larger numbers indicate more plausible English words (so *plausibility*(‘sun’) would be positive, while *plausibility*(‘hesun’) would be negative). The value of *plausibility*(x) is defined for any string of characters.

Given a long string of letters $y = y_1y_2...y_k$, a segmentation of y is a partition of its letters into contiguous blocks of letters; each block corresponding to a word in the segmentation. The total plausibility of a segmentation is determined by adding up the plausibility scores of each of its blocks. For example, the plausibility of the segmentation “here comes the sun” of the string “herecomesthesun” is equal to *plausibility*('here') + *plausibility*('comes')+ *plausibility*('the') + *plausibility*('sun').

Give and analyze a polynomial time algorithm that takes a string y and computes a segmentation of maximum total plausibility, *MaxPlausibility*(y). (One call to the function *plausibility*(x) can be considered as a single computational step.) Follow these steps:

a) prove that the problem has optimal substructure

b) write a recursive expression for *MaxPlausibility*, specifying the base cases and the goal

c) write a recursive algorithm with memoization, and analyze its time and space complexity

d) derive a valid order in which subproblems can be evaluated bottom-up, and write the iterative (bottom-up) version of your algorithm.