Homework 5: Dynamic Programming

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less.

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to Gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Asymmetric Edit Distance [10 points]

For this problem, we're going to redefine edit distance as the smallest sum of penalties for the operations below to convert one string into another but our definition won't be symmetric since deleting a character will be less of a penalty than inserting one:

- 2 points for deleting a character from the first string x.
- 3 points for inserting a character in the first string x.
- 4 points for substituting a character.

(In class we had the same penalty δ for unaligned characters in either string).

(a) Update the recurrence to calculate the new edit distance

(b) Build the table to evaluate the recurrence and state the distance between the strings "space" and "paced"

2. Dynamic Pastries [25 points]

Recall the following set-up for Q1 on Homework 4:

Paul's TAs want pastries during the weekly meetings, and to feed all of his TAs, he needs to purchase **exactly** n total pastries. At the bakery from which he buys his pastries, pastries come in boxes that fit differing numbers of pastries (e.g., donuts come in a box that fits a dozen, croissants only come in a box that fits five, etc.) but you can always buy a chocolate cake that fits in a single box. Paul is interested in finding the minimal number of **boxes** he will have to carry, while getting exactly n total pastries.

`Minimum Pastry Boxes`
Input: An array $P[]$ containing the pastry ordering quantities. You may assume P contains only positive `ints`, that $P[0] = 1$, and that $P[i] < P[j]$ for all $i < j$. And a positive integer n.
Output: The minimal number of boxes required to ensure exactly n pastries in the order.

On Homework 4, you proved that a greedy algorithm did **not** produce the minimum number of pastry boxes.

Design a dynamic programming algorithm that *does* produce the minimum number of pastry boxes given that there are p possible kinds of pastries to choose from, using the following steps:

(a) Define, in English, the quantities that you will use for your recursive solution.

(b) Given a recurrence relation for the quantities you have defined.

(c) Argue for the correctness of your recurrence relation.

(d) Describe the parameters for the subproblems in your recursion and how you will store their solutions.

(e) In what order can you evaluate them iteratively?

(f) Write the pseudocode for your iterative algorithm

(g) Analyze the running time of your algorithm in terms of n and p.

3. It's Tough to Make Predictions, Especially About the Future [25 points]

That piece of Yogi Berra wisdom doesn't bother you because you happen to know someone who can make such perfect predictions in a particular narrow domain: Lord of the Rings memorabilia. In particular they know the future prices of a special one-of-a-kind piece of Lord of the Rings memorabilia: the one ring to rule them all. In particular, suppose that you know for each of the next n days, what the price p_i in dollars of the one ring will be on day i. You aren't interested in the power of the ring (that doesn't seem to be healthy for anyone) but you are interested in making money from buying and selling the ring.

Suppose that on each day you can either buy the ring (if you don't have it) or sell the ring (if you do). Each time you sell the ring there is a fixed commission of c dollars for the sale that you need to pay. (When you buy it, you don't pay commission.)

We'll assume that the reason that you've been let in on this secret future knowledge is that you have at least $\max_{i=1}^{n} p_i$ dollars so you can make the purchases without needing to borrow money.

Design an efficient dynamic programming algorithm (running in $O(n)$ time for full credit but any polynomial-time algorithm can get almost all the credit) that, armed with the knowledge of the future prices, can tell you what your maximum total profit from buying and selling the one ring (possibly multiple times) can get you. (You don't actually need to produce the sequence of purchases and sales, just produce the maximum profit number in dollars.)

As usual complete the following components for your solution:

(a) Define, in English, the quantities that you will use for your recursive solution.

(b) Given a recurrence relation for the quantities you have defined.

(c) Argue for the correctness of your recurrence relation.

(d) Describe the parameters for the subproblems in your recursion and how you will store their solutions.

(e) In what order can you evaluate them iteratively?

(f) Write the pseudocode for your iterative algorithm

(g) Analyze the running time of your algorithm.

4. Formatting Paragraphs [25 points]

In typesetting systems, like the LaTeX system in which this problem set is formatted, the input for paragraphs is given in blocks of plain text consisting of words and punctuation separated by blanks and new lines. Formatted paragraphs are produced by laying out this text using fixed maximum width L per line by figuring out where to break each line of text and continue the paragraph on the next line. An innovation of the TeX system on which LaTeX is based was to phrase paragraph formatting as an optimization problem. which produced superior paragraph formatting and has been adopted by many other word processing systems.

We consider a simplified version of this optimization problem in which words are considered indivisible units of fixed width (no hyphenation) and punctuation is considered part of the word it is next to.

In our version, the input consists of a sequence of n positive real numbers $w_1, \ldots, w_n$ representing the lengths of that the words would occupy in a given font in order, together with a positive real number b representing the minimum width of a blank in that font, and another real number representing the allowed width L of each line. (You can assume that each $w_i \leq L$.)

A legal formatting for this text consists of a sequence of integers $0 = e_0 < e_1 < \ldots < e_k = n$ for some k, where the i-th line of the paragraph consists of the words numbered $e_{i-1} + 1, \ldots, e_i$ separated by blanks. (That is, e_i is the number of the word that ends the i-th line in the paragraph and the number of blanks in a line is 1 less than the number of words in it.) It is required that no line be longer than L, so if a line consists of words t through u then

- $w_t + b + w_{t+1} + b + ... + b + w_u \leq L$.
- The *slack* of this line is the difference between the left and right sides of this inequality.

An optimum formatting for this input is one that minimizes the sum of the squares of the slacks of all but the last line since (as is usual in paragraph formatting) the last line is simply left-justified and we don't care how long it is compared to the other lines.

Give an efficient dynamic programming solution that finds an optimal formatting from the inputs $w_1, \ldots, w_n, b, L$, as usual according the steps for dynamic programming:

(a) Define, in English, the quantities that you will use for your recursive solution.

(b) Given a recurrence relation for the quantities you have defined.

(c) Argue for the correctness of your recurrence relation.

(d) Describe the parameters for the subproblems in your recursion and how you will store their solutions.

(e) In what order can you evaluate them iteratively?

(f) Write the pseudocode for your iterative algorithm

(g) Analyze the running time of your algorithm.

5. A Genealogy Project [Extra Credit]

You are not required to submit attempts at extra credit problems (they do not count toward the dropped/counted problems at the end of the quarter). They will have much *smaller effects on grades than the main problems, so we do not recommend attempting them until you've done all the other problems on the assignment. At the end of the quarter, after determining grade breaks, we will add in extra credit points.*

In this problem you are given as input a rooted binary tree $T = (V, E)$ with each leaf w labelled by a symbol s_w from some fixed alphabet A, as well as a two-dimensional non-negative array of costs indexed by pairs of elements of A

such that $c[s,t]$ is the cost of changing symbol s to symbol t and this satisfies $c[s,s] = 0$ and $c[s,t] = c[t,s]$ for all s,t in A.

Design an efficient algorithm to label each internal node v of the binary tree T by a symbol s_v from A so as to minimize the sum over all (u,v) in E of $c[s_u, s_v]$.

(This kind of problem arises in computational biology when one wishes to assess a potential evolutionary tree and reconstruct properties of potential ancestral species given this tree. In this case each "symbol" might be a very short sequence - or even just a letter - of DNA or protein that might at a given position within a longer sequence, and $c[s,t]$ is the cost of the evolutionary change in going from s to t.)