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 our 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. Modified 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 inserting a character will be less of a penalty than deleting a character:

- 3 points for deleting a character from the first string x.
- 2 points for inserting a character into the first string x.
- 3 points for inserting two characters into x in consecutive locations.
- 4 points for substituting a character.

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

(b) Build the table to evaluate the recurrence and state the distance from the string "pair" to the string "spared".

2. Dynamic Pastries [25 points]

The CSE 421TAs want pastries during the weekly meetings, and to feed all of of the TAs, Nathan wants 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 boxes of five, etc.) but you can always buy a chocolate cake that comes alone in a box. Nathan 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.

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. Insider Trading [25 points]

Nevermind who it is, but Nathan has a source. This source has provided Nathan with the exact values of front row tickets to the upcoming Taylor Swift concert on the resale market for the next n days. In particular, Nathan knows that Taylor Swift tickets will sell for exactly p_i dollars on day i. Nathan doesn't actually care to attend the concert, but he plans to use this knowledge to make a profit. To avoid suspicion, Nathan can only own up to one ticket at a time, and he can only buy or sell a ticket once per day.

On each day Nathan can either buy a ticket (if he doesn't have one) or sell his ticket (if he does). Each time he sells a ticket there is a fixed fee of c dollars for the sale that he needs to pay. (When he buys one, he doesn't pay a fee.)

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 Nathan his maximum total profit from buying and selling tickets. (You don't actually need to produce the sequence of purchases and sales, just produce the maximum profit number in dollars.)

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. Playing Post Office [25 points]

We idealize interstate highway I-5 as a straight highway from Washington all the way to California. There are n towns/cities (we will just call them all towns) alongside this highway. Think about the highway as a real axis, and the position of a town is some non-negative real number x_i representing its distance from the northern Washington border. Assume that there are no two towns are the same position, i.e., $x_i \neq x_j$ for $i \neq j$. The distance between two towns x_i and x_j is simply $|x_i - x_j|$. Each town i has a population of p_i inhabitants.

Originally USPS had post offices in each town but they have been reducing the number of post offices and want to keep only k post offices in some, but not necessarily all of the towns along I-5. A town and the post office in it have the same position. We want to choose the k positions of these post offices so that the *total sum over all people* of the distance that each person needs to travel to their *nearest* post office is minimized.

Design an algorithm that runs in time polynomial in n and outputs the minimum possible total sum of distances to the optimal location for post offices.

For example, given the following location of 5 towns with $x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 6,\ x_5 = 7$ with populations $p_1 = 50,\ p_2 = 20,\ p_3 = 40,\ p_4 = 35, p_5 = 30$ and $k = 2$, then the optimal location for post offices are in towns 2 and 4, and your algorithm should output 120 as the *sum* of distances that people need to travel to their nearest post offices.

(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 k.

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.)