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. **If you submit an answer substantially longer than 2 pages, the TAs are allowed to stop reading at the end of page 2.**

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 have a different box for each problem, please give yourself time to submit.

1. Longer 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:

- 2 points for deleting a character.
- 2 points for inserting a character.
- 3 points for substituting a character

(If each of those numbers had been a 1, we'd have the edit distance we saw in class).

(a) Update the recurrence to calculate the new edit distance (start from the one Lecture 13 slide 9).

(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 problem from Homework 4

Robbie realized after the Thai food, his TA's want pastries during the weekly meetings, and to feed all of his TA's, 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. Robbie 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.

(a) Write a recurrence (or multiple recurrences) that you will use to solve this problem.

(b) Clearly state **in English** what your recurrence(s) calculate. Be sure to mention any parameters you use.

(c) Give a brief explanation of your recurrence; we're not looking for a formal proof, but a brief explanation of what each of the cases represent, and why that set of cases is exhaustive.

(d) Describe a memoization structure.

(e) Describe a filling-order for your memoization structure. If you wrote (iterative) pseudocode to fill the structure, what would its running time be?

(f) What is your final answer going to be? (where in the memoization structure do we look?)

3. Plan Your Bathroom Breaks

You're planning your first post-pandemic roadtrip! Unfortunately, given the time since your last vacation, your bladder isn't exactly in road-trip-ready shape. As a result, you will need to visit a rest stop at most every 500 miles (every time you visit the rest stop, you also get another beverage, so you again need to stop within the next 500 miles, regardless of when you last stopped).

You'll have a sorted array `R[]` that contains the mile locations of all rest stops (all rest stops are on a straight line and the mile location is calculated from your starting point, you can assume elements are distinct). `R[n-1]` is your destination. You will also be given an array `T[]` of the same length. `T[]` contains the amount of time it takes to use that rest stop (which can vary for different stops); it takes 0 time to "skip" a rest stop. Your goal is to minimize the time added to your trip by the rest stops you take.

Given arrays `R[]`, `T[]`, return the minimum amount of time you can add to the trip, while satisfying your bladder-based-requirements.

(a) Write a recurrence (or multiple recurrences) that you will use to solve this problem.

(b) Clearly state **in English** what your recurrence(s) calculate. Be sure to mention any parameters you use.

(c) Give a brief explanation of your recurrence; we're not looking for a formal proof, but a brief explanation of what each of the cases represent, and why that set of cases is exhaustive.

(d) Describe a memoization structure.

(e) Describe a filling-order for your memoization structure. If you wrote (iterative) pseudocode to fill the structure, what would its running time be?

(f) What is your final answer going to be? (where in the memoization structure do we look?)

4. `#AmBigUITies`

On social media sites, it is common to make a hashtag that is a multi-word phrase (even though tags will not allow for spaces), and use CamelCase to indicate how the string should be parsed. For some strings, capitalization is key as the same set of characters can be split into valid English words in more than one way. For example, the common disclaimer `#NotAnAd` ("not an ad") can be misunderstood as `#NoTanAd` ("no tan ad"). Most sites treat hashtags as case-insensitive (e.g., `#notanad`, `#NOTANAD`,`#NotAnAd`, and `#NoTanAd` are all "the same" hashtag), which can lead to confusion for those hashtags that have multiple interpretations.

Your task, given a string s, return `true` if s, written as an all-lower-case hashtag, is **ambiguous** because there is more than one way to break it into substrings where each substring is a word, and `false` otherwise. Assume you have access to a function `DictionaryLookUp(w)` which will return `true` if `w` is a valid English word and `false` otherwise.

For example

- On input `yearlyespousedad` you should return `true` (there are many splits, for example, yearly/espoused/ad, yearly/espouse/dad, year/lye/spouse/dad – there are others)
- On input `superbowl` you should return `true` (Super Bowl or Superb Owl)
- On input `pizzapie` you should return `false` as there is exactly one split (pizza pie)
- On input `eiohsdlkdn`, you will return `false` (there is no valid way to break the string into English words).

In you running time analysis, assume that `DictionaryLookUp(w)` takes t time, regardless of the length of w. Give your running time in terms of t and n, the length of the input string.

(a) Write a recurrence (or multiple recurrences) that you will use to solve this problem.

(b) Clearly state **in English** what your recurrence(s) calculate. Be sure to mention any parameters you use.

(c) Give a brief explanation of your recurrence; we're not looking for a formal proof, but a brief explanation of what each of the cases represent, and why that set of cases is exhaustive.

(d) Describe a memoization structure.

(e) Describe a filling-order for your memoization structure. If you wrote (iterative) pseudocode to fill the structure, what would its running time be?

(f) What is your final answer going to be? (where in the memoization structure do we look?)