Homework 2

Due: Oct. 8, 2020 at 2:30 PM EDT (scratch work submission)
Oct. 15, 2020 at 2:30 PM EDT (final submission)

Problems 4 and 5 on this homework cover material that will be discussed in lecture by Tuesday, October 6th.

Please submit evidence that you've thought about **Problems 1 through 3** by the first due date (scratch work submission) in assignment **HW2: Scratch Work** on Gradescope. This can be an outline of your proofs, any thoughts on how you plan to approach the problems, etc. You are also free to submit completed problems instead of outlines, although you will not receive feedback on your work until after the final homework due date.

This homework must be typed in LaTeX and handed in via Gradescope.

Please ensure that your solutions are complete, concise, and communicated clearly. Use full sentences and plan your presentation before you write. Except in the rare cases where it is indicated otherwise, consider every problem as asking you to prove your result.

Note: The Dynamic Programming problems are longer than the other problems on this homework. Please don't leave them until the last minute!

Selection

Problem 1

Recall the deterministic selection algorithm, which was covered in class and whose pseudocode appears as Algorithm 9.7, DeterministicSelect(S, k), in the book, as shown in Figure **??**. Assume that if baby sequence (group) S_g has 2 or 4 elements, the algorithm picks the first or second element of S_g as its baby median, x_g, respectively.

Input
    Sequence S of n comparable elements, and an integer $k \in [1, n]$
Output
    The kth smallest element of S
procedure DeterministicSelect(S, k)
    if $n = 1$ **then**
        return the (first) element of S
    end if
    Divide S into $g = \lceil n/5 \rceil$ groups, $S_1, ..., S_g$, such that each of groups $S_1, ..., S_{g-1}$ has 5 elements and group S_g has at most 5 elements.
    for $i \leftarrow 1$ to g **do**
        Find the baby median, x_i, in S_i (using any method)
    end for
    $x \leftarrow$ DeterministicSelect$(\{x_1, ..., x_g\}, \lceil g/2 \rceil)$
    Remove all the elements from S and put them into three sequences:

- L, storing the elements in S less than x,
- E, storing the elements in S equal to x,
- G, storing the elements in S greater than x.

    if $k \leq |L|$ **then**
        return DeterministicSelect(L, k)
    else if $k \leq |L| + |E|$ **then**
        return x ▷ each element in E is equal to x
    else
        return DeterministicSelect$(G, k - |L| - |E|)$
    end if
end procedure

Let $S \leftarrow [34, 32, 21, 9, 43, 35, 42, 15, 1, 4, 17, 20, 8, 42, 27]$ be the input sequence for the algorithm. Show the step-by-step execution of DeterministicSelect$(S, 8)$. In particular, you should write the following information for each recursive call of the algorithm and indent appropriately the recursive calls.

- the input sequence, S
- the input rank, k
- the pivot, x (if the recursive call reaches this step)
- sets L, E and G (if the recursive call reaches this step)
- the output value returned

To give you a template, we have provided below the initial part of the solution for you. Note how in the LaTeX source for the solution, the nesting of recursive calls corresponds to the nesting of itemize environments.

- DeterministicSelect$(S, 8)$

  Baby sequences: $[[34, 32, 21, 9, 43], [35, 42, 15, 1, 4], [17, 20, 8, 42, 27]]$

  Baby medians: $[32, 15, 20]$

  - DeterministicSelect$([32, 15, 20], 2)$

    Baby sequences: $[[32, 15, 20]]$

    Baby medians: $[20]$

    * DeterministicSelect$([20], 1)$
      return 20

    $x \leftarrow 20$

    $L \leftarrow [15]$

    $E \leftarrow [20]$

    $G \leftarrow [32]$

    return 20

  $x \leftarrow 20$

  $L \leftarrow ...$

  $E \leftarrow ...$

  $G \leftarrow ...$

Greedy Algorithms

Problem 2

James is very ambitious about cooking and he want to be the best cook he can be. To develop his cooking skills he travels to Cookfest, an international cooking festival where best chefs around the world gather to share their knowledge with each other and have a good time.

1. Various workshops are scheduled for the first day of Cookfest. Unfortunately some of the workshops may be at overlapping times. James wants to attend as many workshops as possible (without being rude and arriving late or leaving early). If the i^{th} workshop goes from time s_i to time t_i, evaluate each of the three scheduling strategies below, and either *prove* it is correct or find a *counterexample* to show that it is not optimal.

   (a) Greedily pick workshops in increasing order of start time. (Sort workshops in increasing order of start time s_i i.e. earliest to latest, and repeatedly run the following process: pick the first workshop in the list, then delete all overlapping workshops).

   (b) Greedily pick workshops in decreasing order of start time. (Sort workshops in decreasing order of start time s_i i.e. latest to earliest and repeatedly run the following process: pick the first workshop in the list, and delete all overlapping workshop).

   (c) Greedily pick workshops in increasing order of length $(t_i - s_i)$.

2. After a long day of attending as many workshops as possible, James wants to try out what he has learned so he decides to try out a new recipe which he came up with. To cook his special meal, he needs to buy certain ingredients which are available at shopping stands at Cookfest. The ith shop at the festival will open at time s_i and will close at time t_i. James wants to minimize the the time spent outside of his kitchen so he can focus on cooking as much as possible while also buying every ingredient he needs. Therefore James wants to make as few trips to Cookfest as possible.

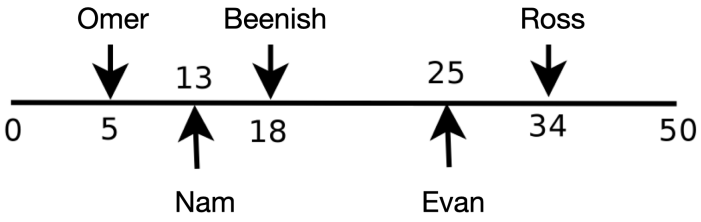
   Design a greedy algorithm for James to figure out what times he should make a trip to the festival, and prove its correctness.

   (To simplify the problem, you can assume it takes no time to go to the festival, buy ingredients and come back. You can also assume James buys one ingredient at every shop and it does not matter at what time he buys an ingredient as long as he buys it at the end of the day.)

Problem 3

The power lines along a country road have been modified to carry broadband internet. Wi-Fi towers are being built along the road to provide the community with internet. To find the minimum number of towers required so that each house is sufficiently close to at least one tower, we model the problem as follows:

a) The entire course staff has taken up residence on Algorithm Street. The diagram below shows where on the street they all live.



Omer lives 5 miles down the road, Nam lives 13 miles down the road, and so on. Wi-Fi towers have an effective radius of 5 miles. Give the locations for a minimal number of wi-fi towers such that each staff member has internet.

b) We're given a line segment ℓ, a set of non-negative numbers N that represents the locations of customers on ℓ, and a distance d. We wish to find a minimum number of points to put a wi-fi tower on ℓ such that each location is at most d from some tower. Give an efficient greedy algorithm which returns a minimum size set of points. Prove its correctness and justify its runtime.

Now we generalize our model to account for houses which are not by the side of the road.

c) We're given a line segment ℓ, a set of pairs N representing locations of customers, and a distance d. For each pair $(x, y) \in N$, $x \in [0, \infty)$ is the distance along ℓ and $y \in [-d, d]$ is the distance above or below ℓ. We wish to find a minimum number of points to put a wi-fi tower on ℓ such that each location in N is at most d from some tower. Modify your algorithm from part (b) to solve this variation of the problem. You do not need to prove its correctness, but please explain how your proof from part (b) would (or would not) need to change based on your modifications.

Can we generalize further?

d) Does the correctness of your algorithm depend on the fact that ℓ is a line segment and not some curve? If so, give an example which illustrates the problem with your algorithm when ℓ is a line. If not, explain how your algorithm could handle a curve. You shouldn't be writing another algorithm, or modifying your existing algorithm, just explain your reasoning.

Dynamic Programming

Problem 4

The **edit distance** between string X and string Y of length n and m, respectively, is defined as the number of edits that it takes to change X into Y.

For all parts of this problem, we strongly suggest you to solve each part by hand, to give you insight, as opposed to coding up the algorithm and letting the computer solve the problems for you.

1.
   - Write the recurrence relation $T(i, j)$, which is defined as "the minimum cost of editing the first i characters of X into the first j characters of Y". Be sure to include the base case(s) in your recurrence relation.
   - Justify the correctness of your solution, meaning prove that your solution will hold for all $i, j \geq 0$.

2. You hate to see it. You are attending a post-COVID Giant Dinner Party ™ at Christina Paxson's house, where she, intending to showcase to donors the resilience that Brown students have in the face of adversity, declared that she would only allow entrance if you can correctly calculate the edit distance of the names of the dishes that are offered at the event.

   - Given $X =$ **TONKATSU** and $Y =$ **PORKCHOP**, fill in the 2D array T with values calculated using the recurrence relation and base cases you have figured out above.

		P	O	R	K	C	H	O	P
	0	1	2	3	4	5	6	7	8
	0									
T	1									
O	2									
N	3									
K	4									
A	5									
T	6									
S	7									
U	8									

   Give a final result for the edit distance between X and Y.

   - Briefly describe the process that you took to fill in the table above, and justify how your method is consistent with the recurrence relation that you came up with in the previous part.

Donors, of course, also have to pass this test to enter the event. Like you, your good friend Marty G. wants to do it the smart way - come up with a good dynamic algorithm to solve the problem first, and then apply the algorithm to get the final result. Marty's algorithm, however, runs into an error in the initialization phase. Being sick of exclusivity, you want to offer Marty an example of two words that his algorithm will fail on so Marty can also figure out the problem and enjoy the dinner with you.

Suppose that there is an error in the initialization, and, mistakenly, $T(0, 2) = 17$ (instead of 2). Find two strings of length 5 such that the algorithm will compute the *wrong* edit distance in entry $T(5, 5)$. Justify your choice.

Note: These two strings need not be actual words with meaning.

Hint: What is the meaning of the wrong entry $T(0, 2)$? When, broadly, will the recurrence rely on this entry? How can you find two words that lead to this occurring?

Problem 5

Upon you successfully proving that $P = NP$, your company's product now has the super power to find out everyone's relationship with every bok choy that has ever been consumed. Your company now needs to display all of the information you've uncovered to the customers, and want to find the way to arrange the text in the most aesthetically pleasing way possible.

The first engineer your company hired ordered all of the words in this way: as you go along, greedily put as many words as possible on the current line until you hit the margin, then go to the next line.

The Chief Aesthetics Officer (CAO) decided this would be not aesthetic enough and fired the engineer in question. Being completely in awe of your Computer Science expertise, your CAO has now hired you to also devise a more elegant word-arranging scheme.

You decide that you'd like to somehow arrange the text such that you minimize the **penalty** on each line of text. **Penalty** is defined to be the square of the difference (in number of characters) between the last letter in a line of text and the actual line cutoff length. Thus, your algorithm is to minimize the total **penalty** over all lines of text *except the last line*. Further, the last line cannot go over the margin. Note that on any other line, your algorithm may arrange words such that they go over the line cutoff length, if this minimizes the total **penalty**.

For example, if you want to write: "Your closest bok choy relation is Vanessa The Great, who was eaten by Barack Obama in 2005." with margins that allow 16 characters between them, the greedy solution would do this:

```
|Your closest bok|    : 16 characters, 0 characters remaining (0^2 = 0 penalty)
|choy relation is|    : 16 characters, 0 characters remaining (0^2 = 0 penalty)
|Vanessa The     |    : 11 characters, 5 character remaining (5^2 = 25 penalty)
|Great, who was  |    : 14 characters, 2 characters remaining (2^2 = 4 penalty)
|eaten by Barack |    : 15 characters, 1 character remaining (1^2 = 1 penalty)
|Obama in 2005.  |    : 14 characters, 2 character remaining (0 penalty — last line)
```

This has a total penalty of 0+0+25+4+1+0 = 30. However, the optimal solution has only 5 penalty:

```
|Your closest bok|    : 16 characters, 0 characters remaining (0^2 = 0 penalty)
|choy relation is|    : 16 characters, 0 characters remaining (0^2 = 0 penalty)
|Vanessa The Great,   : 18 characters, 2 characters over (2^2 = 4 penalty)
|who was eaten by|    : 16 characters, 0 characters remaining (0^2 = 0 penalty)
|Barack Obama in |    : 15 characters, 1 character remaining (1^2 = 1 penalty)
|2005.           |    : 5 characters, 11 characters remaining (0 penalty — last line)
```

(a) Design a dynamic programming algorithm to achieve the minimum penalty, when input a sequence of words. Your algorithm should both find the minimum possible penalty as well as the line splitting scheme that yields this penalty. As a first step, or for partial credit, consider using the same penalty for the last line as for all the others.

(b) Prove the correctness of your algorithm.

(c) What is the runtime of your algorithm? Justify your answer.

Hint: The point of a dynamic programming recurrence is to optimally make a choice about the "last action", making use of a table that represents information about how to optimally solve subproblems. How can you set things up (the "subproblem structure") so that you will be able to cleanly make optimal choices based on optimal solutions to smaller versions of the same problem?