..

BUTWORK

Write down the answers and then, explain each of them in a video.

---

1. Given two strings $a_1a_2...a_n$ and $b_1b_2...b_m$, find their minimum edit distance. The edit distance is the number of mismatches in an alignment. For example, the minimum edit distance between the two strings ”SUNNY” and ”SNOWY” in the following alignment is 3:

Likewise, the minimum edit distance between ”INTENTION” and ”EXECUTION” is 5.

(a) Represent the minimum edit distance between $a_1a_2 \ldots a_k$ and $b_1b_2 \ldots b_l$ by a function name.

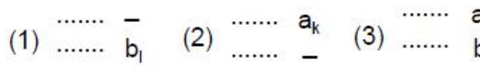
“minDistance”

(b) Every function has variable(s). Now, it is time to decide on them, by which you can narrow down the problem into sub-problems. Although they may seem straightforward, they can be tricky. The function’s variables usually correspond to the indices and/or capacities that make it easier to construct smaller sub-problems. For example, the matrix indices in the Matrix chain Multiplication, the index of the last item we consider and the weight capacity in the Knapsack, or the lengths of the two sequences in Longest Common Subsequence etc. As an example, however, there are no variables related to the three possible weights (i.e. 1, 4, 5) but only the weight capacity in the unbounded Knapsack because the weights are constants given to the problem. Write down the variables of your function in between parenthesis and define what your function represents as complete and concise as possible in only one sentence, as we did in class (e.g. Given a string X_i of length i and another string Y_j of length j, c(i, j) represented the length of their longest common subsequence in the longest Common Subsequence problem.)

minDistance(str1, str2, x, y):

This function calculates the minimum edit distance between str1 from 0 to x and str2 from 0 to y; initially x is equal to the length of str1 and y is equal to the length of str2.

(c) We can start writing the right-hand side of the formulation in a recursive manner. Now spend a considerable amount of time to imagine how would an optimal alignment would look like for two arbitrary strings. Yes, it is actually two strings that are placed one above the other and expanded at some locations with a ’ ’. Consider the last choice you would do to obtain this imaginative optimal solution. Yes, you are right! You must choose what to put as two last characters. Here, there can be three different scenarios, where the 3rd scenario has two subcases:



In the first scenario, assume that the edit distance is minimized if only you don’t put any of the first string’s letters, but just put a ’ ’ for the first string and put the second string’s last letter. Given this assumption, write down the sub-problem you must solve (i.e. the same function with slightly different variables). Then, if the last alignment was as in the first scenario, the right-hand side would be some function of the function you have written for the corresponding sub-problem. Write down the right-hand side for the first scenario.

Sub-problem: minDistance(str1, str2, x, y-1)

Right-hand side for the first scenario: minDistance(str1, str2, x, y-1) + 1

(d) In the second scenario, assume that the edit distance is minimized if only you don’t put any of the second string’s letters, but just put a ’ ’ for the second string and put the first string’s last letter. Given this assumption, write down the sub-problem you must solve (i.e. the same function with slightly different variables). Then, if the last alignment was as in the second scenario, the right-hand side would be some function of the function you have written for the corresponding sub-problem. Write down the right-hand side for the second scenario.

Sub-problem: minDistance(str1, str2, x-1, y)

Right-hand side for the second scenario: minDistance(str1, str2, x-1, y) + 1

(e) In the third scenario, assume that the edit distance is minimized if only you put the last letters from both strings. Given this assumption, write down the sub-problem you must solve (i.e. the same function with slightly different variables). Then, if the last alignment was as in the third scenario, the right-hand side would be some function of the function you have written for the corresponding sub-problem. This function has two legs: the one where the last letters are not the same, and the one where the last letters are not the same. Write down the right-hand side for the third scenario.

If the last characters are not the same:

minDistance(str1, str2, x-1, y-1)

If the last characters are the same:

minDistance(str1, str2, x-1, y-1) + 0

Right-hand side for the third scenario: minDistance(str1, str2, x-1, y-1) + "variable"

(Here variable will be 0 or 1 depending on whether the last characters are same)

(f) Actually, we do not do any choice in dynamic programming. What we must do is first to consider all possibilities like above and choose the minimum of those right-hand sides. Now you are ready to minimize them. Write down the whole formulation which collects all the above right-hand side functions into one as a minimization function.

Math.min(minDistance(str1, str2, x, y-1) + 1, minDistance(str1, str2, x-1, y) + 1, minDistance(str1, str2, x-1, y-1) + "variable"))

2. In this question, consider the Matrix Chain Multiplication problem that we discussed in class. First, understand what is being asked to minimize in the problem.

   (a) Give the optimization objective function that you will develop a name.

   "minMultiplication"

   (b) Write down the variables of your function in between parenthesis and define what your function represents as complete and concise as possible in only one sentence, as we did in class.

   For the Matrix Chain Multiplication problem, minMultiplication(x,y) denotes the minimum number of multiplications required to compute the chain multiplication between matrix A_x and matrix A_y.

   (c) We can start writing the right-hand side of the formulation in a recursive manner. Now spend a considerable amount of time to imagine how would an optimal splitting would look like for a set of matrices. Consider the last choice you would do to obtain this imaginative optimal solution. Yes, you are right! You must choose where to split. As you can guess, each index k is a different scenario.

   In the first scenario, assume that the number of operations to perform is minimized if only you split from the first index (i.e. k = 1). Then, if the last spit was at k = 1, the right-hand side would be some function of the function you have written for the corresponding sub-problem. Write down the right-hand side for the first scenario.

   Right-hand side for the first scenario

   $$\textbf{minMultplication (x,y) = minMultplication (x,1) + minMultplication (1 + 1, y) + } p_{x-1} \cdot p_1 \cdot p_y$$

   (d) Write down the right-hand side for k = 2. As you can guess, you need to do this for all valid k's.

   Right-hand side for the second scenario

   $$\textbf{minMultplication (x,y) = minMultplication (x,2) + minMultplication (2 + 1, y) + } p_{x-1} \cdot p_2 \cdot p_y$$

   (e) Now you are ready to minimize the set of all functions produced by different scenarios. Write down the whole formulation which collects all the above right-hand side functions into one as a minimization function.

$$\textbf{minMultplication (x,y) = } \min_{1 \le k \le j-1} \{ \textbf{ minMultplication (x,k) + minMultplication (k + 1, y) + } p_{x-1} \cdot p_k \cdot p_y \}$$