ADA 2022 Tutorial 4

This tutorial is more warmup on DPs. What we would like you to do for each of the problems is:-

1. Define the subproblems clearly
2. Write a recursion using the above definition and argue properly about why the recursion is correct (this is the optimal substructure property)
3. Implement an iterative algorithm using tables and argue runtime.

1 Longest Common Subsequence

Given two strings $X = x_1, x_2, x_3, \cdots x_m$ and $Y = y_1, y_2, \cdots y_n$, find a common subsequence (not necessarily contiguous) of X, Y that is of the longest possible length.

Solution. The first thing to note here is that there might not be a unique optimal solution to this problem. Can you give an example ?

Again, the first step would be to think of an optimal solution to this problem and try to imagine what happens at the last characters of the two given strings X, Y .

Let $Z^\star = z_1, z_2, \cdots z_k$ be the longest common subsequence. Now, there could be one of the following two cases :

- Case I : $x_m = y_n$, that is the last characters of the two strings match. We claim that, in this case, $z_k = x_m = y_n$ that is , there is one optimal subsequence which ends with these occurences of this character. Why ? Well assume that this is not the case.

  Now, if none of x_m and y_n are the last character of $Z^\star$, then we can increase the length of the subsequence by appending this common character. This contradicts the fact that $Z^\star$ is an optimal subsequence

  Now suppose (without loss of generality) that x_m is matched with a different occurrence of y_n - call it y'. Note that since y_n is the *last* character in Y, y' appears before y_n. Hence, instead of matching x_m to y', we can simple match it to y_n without changing the feasibility or optimality of the solution.

  Ok, now that we know the last common character is necessarily part of $Z^\star$, what is the subproblem that we need to solve ? Well clearly it is the subproblem on the substrings $X[1, 2 \cdots x_{m-1}], Y[1, 2, \cdots y_{n-1}]$. The formal claim is

  Claim 1 *In case I, an optimal solution to the subproblem $X[1, 2 \cdots x_{m-1}], Y[1, 2, \cdots y_{n-1}]$ is indeed the subsequence $Z^\star \setminus z_k$*

- Case II : Suppose $x_m \neq y_n$. In this case, the last two characters cannot be matched with each other to form z_k. Hence, the claim is

  Claim 2 *In case II, an optimal solution to the subproblem $X[1, 2 \cdots x_{m-1}], Y[1, 2, \cdots y_{n-1}]$ is same as $Z^\star \setminus z_k$*

Subproblem definition. For $i = 0, 1, 2, \cdots m, j = 0, 1, 2, \cdots n$, let $Subseq(X_i, Y_j)$ denote the optimal common subsequence length for substrings $X[1, 2, \cdots i], Y[1, 2, \cdots j]$.

Recurrence. For all $i \geq 1, j \geq 1$,

$$Subseq(X_i, Y_j) = \max\{Subseq(X_{i-1}, Y_{j-1}) + 1, Subseq(X, Y_{j-1}), Subseq(X_{i-1}, Y)\}$$

$$Subseq(\emptyset, Y_j) = 0, \forall j \geq 0$$
$$Subseq(X_i, \emptyset) = 0, \forall i \geq 0$$

The rest is just converting this to a pseudocode using a 2D table. Can you figure out the reconstruction of actual solution from this table once it is populated ?

2 Dictionary

You are given a string of n characters s, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like *"itwasthebestoftimes..."*). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function $dict()$: for any string w, $dict(w)$ outputs true if w is a valid word false otherwise. Give a dynamic programming algorithm that determines whether the string s can be reconstituted as a sequence of valid words. The running time should be at most $\mathcal{O}(n^2)$, assuming each call to $dict()$ takes unit time.

Solution. This problem has a slightly different flavor since we need to figure out whether a valid sequence of words can be constructed out of the given gibberish.

Subproblem. Let $T[i]$ be 1 if string $s[i \ldots n]$ can be reconstituted as a sequence of valid words else 0. Hence, the subproblem here has a Boolean value. Now the reurrence is quite straightforward.

$$T[i] = \max_{j=i \ldots n} (dict(i \ldots j) = 1 \wedge T[j+1] = 1)$$

The size of the DP table is $1 \times (n+1)$. Filling each entry takes $O(n)$ int the worst case. Therefore time complexity is $O(n^2)$.

3 File Placement Problem

Suppose we want to replicate a file over a collection of n servers,labeled $S_1, S_2, \ldots, S_n$. To place a copy of the file at server S_i results in a placement cost of c_i, for an integer $c_i > 0$. Now, if

a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}, \cdots S_n$ are searched in order until a copy of the file is finally found, say at server S_j, where $j > i$. This results in an access cost of $j - i$. (Note that the lower-indexed servers $S_{i-1}, S_{i-2}, \ldots$ are not consulted in this search.) The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n. We would like to place copies of the files at the servers so as to minimize the sum of placement and access costs. We know that the accesses are going to happen at every server $S_1, S_2, \cdots S_n$. Formally, we say that a configuration is a choice, for each server S_i with $i = 1, 2, \cdots, n-1$, of whether to place a copy of the file at S_i or not. (Recall that a copy is always placed at S_n.) The total cost of a configuration is the sum of all placement costs for the selected servers with a copy of the file, plus the sum of all access costs associated with all n servers. Give a polynomial-time algorithm to find a configuration of minimum total cost.

Solution.

Subproblems. $T(i)$ = The minimum cost solution assuming we are placing a copy at S_i and we only have the servers $S_i, S_{i+1}, \cdots S_n$, for all $i = 1, 2, \cdots n$.

Recurrence.

$$T((n) = c_n)$$

$$T(i) = c_i + \min_{i+1 \leq k \leq n} \left(\frac{(k-i-1)(k-i)}{2} + T(k) \right), \forall i < n$$

Why is the recurrence above correct ? Suppose you have placed a copy at S_i (that is required by the definition of $T(i)$) - you pay c_i. Now, imagine that I tell you that the next copy is placed at server $S_k, i+1 \leq k \leq n$. Then what is the cost of this configuration ? Well, you need to pay access cost for serving $i+1, i+2, \cdots k-1$ with the copy at S_k which is

$$(k-i-1) + (k-i-2) + \cdots + 1$$

We do not need to pay for access at i since we have already placed a copy at S_i. So the other cost we pay is given by the optimal solution to the subproblem assuming S_k has a copy and only servers from k to n. The only catch is - we do not know the correct index k. So we try all and take the one which gives the minimum total cost.