Two DP solutions.

3. We are given two strings of length n and m respectively. We want to find the largest matching substring, i.e. if one text is $a_1 a_2 \cdots a_n$ and the other is $b_1 b_2 \cdots b_m$, then we want to find the maximum number t such that there are indices $1 \leq i \leq n$ and $1 \leq j \leq m$, such that, $a_i a_{i+1} a_{i+2} \ldots a_{i+t-1} = b_j b_{j+1} b_{j+2} \ldots b_{j+t-1}$ is satisfied. Give an algorithm for this problem using $O(nm)$ steps.

   Solution: Since the largest matching subsequence must end at some index between $1 \leq i \leq n$ and $1 \leq j \leq m$, we will have subproblems indicating length of longest subsequence **ending** in i, j instead of just longest subsequence.

   - SUBPROBLEMS: $A[i, j]$ = the length of the longest matching substring ending in a_i and b_j.
   - ORDER: In the order of increasing i from 0 to n (top to bottom), and in each row, j increases from 0 to m (left to right).
   - START: $A[i, 0] = A[0, j] = 0$ (there is no matching substring if one of the strings is empty.
   - CONTINUE: if $a_i = b_j$, then $A[i, j] = A[i-1, j-1] + 1$, otherwise 0.
   - END: We take the maximum of all $A[i, j]$'s to find the length of maximum matching substring.
   - CORRECTNESS: The maximum matching substring ending in a_i and b_j is of zero length if $a_i \neq b_j$. If they are equal, then any matching subsequence will give us a matching subsequence ending in a_{i-1} and b_{j-1} by removing the last common letter. This is what we have used in the recurrence.
   - RUNNING TIME: Filling out any $A[i, j]$ requires checking if $a_i = b_j$ and then a constant number of subtractions, additions, and one look up in the A matrix. This is all constant amount of steps, and since there are $(n+1)^2$ entries in the matrix, the running time is $O(n^2)$.
   - OPTIMAL OBJECT: To find the actual matching subsequence, we find the maximum number in the A matrix. Suppose this happens at (k, l) and the value is t. Then $a_{k-t+1} a_{k-t+2} \ldots a_k$ gives us the longest matching subsequence.

6. Each element of a table of size $n \times n$ is a positive integer. We want to go from the bottom left corner of the table to the top right corner by taking one step up or one step to the right in the table. We further want that the elements on our path should be in increasing order. Give an algorithm with running time $O(n^2)$ that determines,
   (a) how many paths satisfy the rules.
   (b) what is the largest value of a path satisfying the rules if the value of a path is the product of the numbers in it.

   Solution only for part (a): We will say the the values stored in the table can be accessed by the matrix $T[1:n, 1:n]$.

   - SUBPROBLEMS: Let $B[i, j]$ indicate the number of paths from $(1, 1)$ to the cell (i, j).
   - ORDER: We will solve in the order of increasing i from 1 to n, and in a row we will increase j from 1 to n (left to right).
   - START: We set $B[1, 1] = 1$. For $j > 1$, if $T[1, j-1] < T[1, j]$, then we set $B[1, j] = B[1, j-1]$ otherwise 0. And similarly for $i > 1$, if $T[i-1, 1] < T[i, 1]$, then we set $B[i, 1] = B[i-1, 1]$ otherwise 0.
   - CONTINUE: let *left* be a boolean variable that is 1 if $T[i, j-1] < T[i, j]$ and 0 otherwise. Similarly, let *down* be a boolean variable that is 1 if $T[i-1, j] < T[i, j]$ and 0 otherwise. Then $B[i, j] = left \cdot B[i, j-1] + down \cdot B[i-1, j]$.
   - END: $B[n, n]$ gives us the number of paths to (n, n).

- CORRECTNESS: Any path to (i, j) has to come from below or from the left and the total number of paths is the sum of the number of paths to the left square and the square below, if the values in the squares is less than the current square.
- RUNNING TIME: We take constant number of checks of the T and B matrices, comparisons, additions and subtractions for filling each $B[i, j]$. Since the B matrix has size $n \times n$, the running time is $O(n^2)$.
- OPTIMAL OBJECT: Since we just have to find the number of paths, this is in $B[n, n]$, and there is no other object to find.