Provide a short explanation of your code/solution. Include references, if any.

Using a dynamic programming table with rows i (string Z) and columns j (string X), we can find the number of distinct occurrences of string Z in string X by calculating for the dp[m][n] where m is the length of string Z and n is the length of string X in the dynamic programming table. In filling up the table, we consider the base cases where null is found in every letter in string X hence we put a row of 1s, and the case where every letter in string Z cannot be found in null hence we put a column of 0s. For each cell, if the letters at the end of each string are the same, we delete the letter and we can either use the deleted letter or skip it. If we use it, we compare the i-1 letters (preceding letters of the last letter of string Z) with the j-1 letters (preceding letters of the last letter of string X) until we get the number of subsequences that match. If we don't want to use it, we compare the j-1 (preceding letters of the last letter in string X) with i (all letters in string Z). However, if they do not match, we only need the preceding letters of the last letter in string X (j-1) to compare with all letters in string Z (i). This means that if the last letters match, we add the top left and left values to identify the number in the current cell, and we take the value on its left if the letters don't match.

i. Characterize the structure of an optimal solution

- The optimal solution is stored in dp[m][n], where m = len(Z) and n = len(X)
- Base Cases
    - row of 1s (dp[0][j] where empty Z is a subsequence of any X);
    - column of 0s (dp[i][0] where Z is not a subsequence of any empty X)
- Recursive Transition Relation
    - new cell value = top left + left (Z[i] == X[j] then dp[i][j] = dp[i-1][j-1] + dp[i][j-1]) if the same and new cell value = left (dp[i][j] = dp[i][j-1]) if not the same

ii. Define the subproblems

Let dp[i][j] represent the number of times the first i characters of Z appear as a subsequence in the first characters of X.

- If Z[i] == X[j]: we can form Z[0:i] from X[0:j] using 2 cases:
    - Using the matching character (dp[i][j])
    - Skipping the character in X (dp[i][j-1])
    - dp[i][j] = dp[i-1][j-1] + dp[i][j-1]
- If Z[i] ≠ X[j-1]: we can form Z[0:i] from X[0:j] by ignoring the current character in X.
    - dp[i][j] = dp[i][j-1]

iii. Define the input and output

- Input: Strings X and Z
- Output: number of ways Z appears as a subsequence in X (dp[m][n])

iv. Define the recurrence relation using the dynamic programming (DP) approach

- dp[i][j] = dp[i-1][j-1] + dp[i][j-1] if Z[i-1] == X[j-1].
- dp[i][j] = dp[i][j-1] if Z[i-1] ≠ X[j-1].