**Assignment 3**

**Interleaving two strings using Dynamic Programming**

**Team members: NAME - EMAIL - UID**

* Chris Davis Jaldi – jaldi.2@wright.edu – U01099335
* Aditya Mallakula – mallakula.2@wright.edu – U01093160
* Vanaja Uppala – uppala.19@wright.edu – U01080568

**Recurrence relation Used:**

* Interleaving Validity
* Interleaving Count

**Informal argument for correctness:**

This approach ensures correctness by systematically building up solutions for smaller substrings:

* Order preservation:
  + At each stage, only the characters of *s1* and *s2* are considered in order. This adheres to the rule that *s1* and *s2* must preserve their internal order when forming *s3*.
* Tracking possible interleavings:
  + The algorithm ensures that all valid interleaving paths are considered by keeping track of both *dp[i][j]* (feasibility) and *count[i][j]* (number of ways).

**DP Algorithm Pseudocode:**

1. **Read Input:**

* Load the strings *s1*, *s2*, and *s3* from the input file.

1. **Check Length Constraint:**
   * If the total length of *s1* and *s2* doesn't match the length of *s3*, it's impossible for *s3* to be an interleaving of *s1* and *s2*. Stop here.
2. **Initialize DP Tables:**
   * Create a 2D DP table *‘dp’* to store whether *s3* can be formed up to a certain point using parts of *s1* and *s2*.
   * Create another 2D table *‘count’* to store the number of ways to form *s3*.
   * Set *‘dp[0][0] = True’* and *‘count[0][0] = 1’*, as empty strings interleave to form an empty string.
3. **Handle Base Cases:**
   * For the first row (using only *s2*): If the characters in *s2* match *s3* sequentially, mark it as valid in *‘dp’* and update the count.
   * For the first column (using only *s1*): If the characters in *s1* match *s3* sequentially, mark it as valid in *‘dp’* and update the count.
4. **Fill DP Table for All Combinations:**
   * For every character in *s1* and *s2*:
     + Check if taking the current character from *s1* forms a valid interleaving.
       - If yes: Mark *‘dp[i][j]’* as *True* and add the count from *‘dp[i-1][j]’*.
     + Check if taking the current character from *s2* forms a valid interleaving.
       - If yes: Mark *‘dp[i][j]’* as *True* and add the count from *‘dp[i][j-1]’*.
5. **Check Final Result:**
   * The value at *‘dp[len(s1)][len(s2)]’* tells whether *s3* can be formed by interleaving *s1* and *s2*.
   * The value at *‘count[len(s1)][len(s2)]’* gives the total number of ways this interleaving can happen.
6. **Backtrack to Find Substrings:**
   * If *s3* is interleavable, reconstruct the sequence of substrings from *s1* and *s2* by backtracking through the *‘dp’* table:
     + Start from the bottom-right of the table and move upwards or leftwards depending on which string contributed to *s3* at each step.
     + Record the substrings as you move.
7. **Return Results:**
   * Return whether *s3* is an interleaving, the total number of interleavings, and the substrings from *s1* and *s2*.

**Complexity Analysis:**

* Overall Time Complexity: *O(n X m)*
  + - Where *n = len(s1)* and *m = len(s2)*