Interleaving String Problem Documentation

Given strings s1, s2, and s3, determine whether s3 is formed by an interleaving of s1 and s2.

An interleaving of two strings s and t is a configuration where s and t are divided into n and m substrings respectively, such that:

- s = s1 + s2 + ... + sn
- t = t1 + t2 + ... + tm
- |n m| <= 1

The interleaving is s1 + t1 + s2 + t2 + s3 + t3 + ... or t1 + s1 + t2 + s2 + t3 + s3 + ...

Note: a + b denotes the concatenation of strings a and b.

Example 1

Input:

s1 = "aabcc"

s2 = "dbbca"

s3 = "aadbbcbcac"

Output: true

Explanation: One way to obtain s3 is:

- Split s1 into s1 = "aa" + "bc" + "c"
- Split s2 into s2 = "dbbc" + "a"

Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac". Since s3 can be obtained by interleaving s1 and s2, the output is true.

Example 2

Input:

s1 = "aabcc"

s2 = "dbbca"

s3 = "aadbbbaccc"

Output: false

Explanation: It is impossible to interleave s2 with any other string to obtain s3.

Example 3

Input:

s1 = ""

s2 = ""

s3 = ""

Output: true

Constraints

- 0 <= s1.length, s2.length <= 100
- 0 <= s3.length <= 200
- s1, s2, and s3 consist of lowercase English letters.

Follow-up

- Could you solve it using only O(s2.length) additional memory space?

Approach

We use dynamic programming (DP) to solve this problem. The idea is to create a DP table where dp[i][j] indicates whether s3[0:i+j] can be formed by interleaving s1[0:i] and s2[0:j].

Algorithm

1. **Check Length Constraint:** If the length of s1 plus the length of s2 does not equal the length of s3, return False.

2. **Initialize DP Table:** Create a DP table dp with dimensions (len(s1) + 1) x (len(s2) + 1) and initialize all values to False.

3. **Set Initial Condition:** Set dp[0][0] to True since an empty s3 can be formed by interleaving two empty strings.

4. **Initialize First Row:** Populate the first row of the DP table where dp[i][0] indicates whether s3[0:i] can be formed by interleaving s1[0:i] and an empty s2.

5. **Initialize First Column:** Populate the first column of the DP table where dp[0][j] indicates whether s3[0:j] can be formed by interleaving an empty s1 and s2[0:j].

6. **Fill DP Table: Use the recursive relation to fill in the DP table:**

- *dp[i][j] is True if either:*
  - dp[i-1][j] is True and s1[i-1] matches s3[i+j-1]
  - dp[i][j-1] is True and s2[j-1] matches s3[i+j-1]

7. **Return Result:** The value at dp[len(s1)][len(s2)] will indicate whether s3 can be formed by interleaving s1 and s2.

Complexity Analysis

- Time Complexity: O(len(s1) * len(s2))
- Space Complexity: O(len(s1) * len(s2))

The solution can be optimized to use O(len(s2)) additional memory space by using a rolling array technique.