Problem of the Week – *Longest Common Subsequence of Three Strings*

This problem was asked by YouTube.

Problem Description:

Scenario:
In many real-world applications like version control, spell checking, or DNA sequencing, comparing multiple strings and identifying common patterns is crucial.

In this challenge, you're given **three strings**, and you need to compute the **length of the longest common subsequence (LCS)** that is present in **all three strings**.

A **subsequence** is a sequence that appears in the same relative order, but not necessarily contiguous.
For example, in `"abcde"` and `"ace"`, `"ace"` is a subsequence.

Your goal is to implement an efficient algorithm that returns the **length** of the longest subsequence common to all three strings.

Input Format:

- Three lines of input, each containing one string:
  - `S1` (string 1)
  - `S2` (string 2)
  - `S3` (string 3)

Output Format:

- A single integer representing the length of the longest common subsequence among the three strings.

Constraints:

- `1 ≤ |S1|, |S2|, |S3| ≤ 100`
- Strings may contain **lowercase or uppercase** English letters

Example Input:

```
epidemiologist
refrigeration
supercalifragilisticexpialodocious
```

Example Output:

```
5
```

Explanation:

The longest common subsequence among the three input strings is:

`"eieio"`

So, the output is `5`.

Approach Hint:

- Use **Dynamic Programming with 3D DP table**: `dp[i][j][k]` stores the LCS length of first `i` characters of `S1`, first `j` of `S2`, and first `k` of `S3`.
- Transition:
  - If `S1[i-1] == S2[j-1] == S3[k-1]:`
    `dp[i][j][k] = 1 + dp[i-1][j-1][k-1]`
  - Else:
    `dp[i][j][k] = max(dp[i-1][j][k], dp[i][j-1][k], dp[i][j][k-1])`

Expected Time Complexity:

- `O(N^3)` where `N` = max length of input strings (acceptable for N ≤ 100)

Practice Links:

- 🔗 GeeksforGeeks – LCS of three strings
- 🔗 Leetcode (Related): LCS of two strings

Video Explanation:

- 🎥 YouTube – LCS of 3 strings using 3D DP

Sample Starter Code (Python):

```
def lcs_of_three(a, b, c):
    n1, n2, n3 = len(a), len(b), len(c)
    dp = [[[0]*(n3+1) for _ in range(n2+1)] for __ in range(n1+1)]

    for i in range(1, n1+1):
        for j in range(1, n2+1):
            for k in range(1, n3+1):
                if a[i-1] == b[j-1] == c[k-1]:
                    dp[i][j][k] = 1 + dp[i-1][j-1][k-1]
                else:
                    dp[i][j][k] = max(
                        dp[i-1][j][k],
                        dp[i][j-1][k],
                        dp[i][j][k-1]
                    )
    return dp[n1][n2][n3]

# Input
s1 = input().strip()
```

```
s2 = input().strip()
s3 = input().strip()
print(lcs_of_three(s1, s2, s3))
```