1143. Longest Common Subsequence

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- `1 <= text1.length, text2.length <= 1000`
- `text1` and `text2` consist of only lowercase English characters.

```
def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        n = len(text1)
        m = len(text2)
        dp = [[0]*(m+1) for i in range(n+1)]
        for i in range(1,n+1):
            for j in range(1,m+1):
```

```
                if text1[i-1]==text2[j-1]:
                    dp[i][j] = 1+dp[i-1][j-1]
                else:
                    dp[i][j] = max(dp[i-1][j],dp[i][j-1])
        return dp[len(text1)][len(text2)]
```

Recursive approach:

```
def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        n = len(text1)
        m = len(text2)
        return self.helper(text1,text2,0,0,n,m)

    def helper(self,s1,s2,i,j,n,m):
        if i==n or j==m:
            return 0
        if s1[i]==s2[j]:
            return 1+self.helper(s1,s2,i+1,j+1,n,m)
        else:
            return
max(self.helper(s1,s2,i,j+1,n,m),self.helper(s1,s2,i+1,j,n,m))
```

Recursive Memoized:

```
def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        n = len(text1)
        m = len(text2)
        dp = [[-1]*(m+1) for i in range(n+1)]
        return self.helper(text1,text2,0,0,n,m,dp)

    def helper(self,s1,s2,i,j,n,m,dp):
        if i==n or j==m:
            return 0
        if dp[i][j]!=-1:
            return dp[i][j]
        if s1[i]==s2[j]:
            dp[i][j]=1+self.helper(s1,s2,i+1,j+1,n,m,dp)
        else:
            dp[i]
[j]=max(self.helper(s1,s2,i,j+1,n,m,dp),self.helper(s1,s2,i+1,j,n,m,dp))
        return dp[i][j]
```