DAA Assignment 6

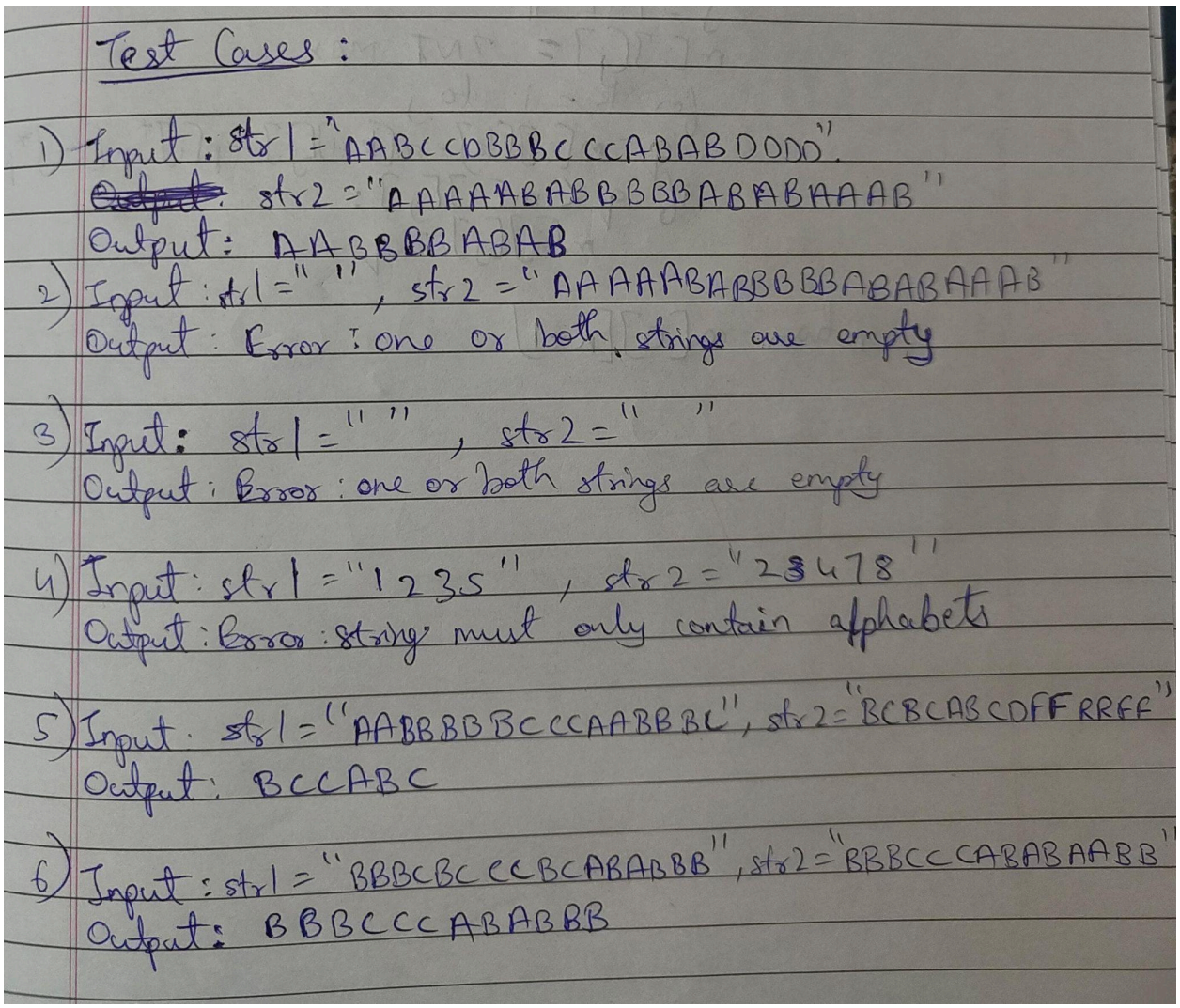
Ishaan Shaikh
231070063
Batch D

Q. Consider grades received by 20 students, like AA, AB, BB, ..., FF of each student.Computer the Longest common sequence of grades among students.

Algorithm:

```
LCS (str1, str2):
//Input: Two strings str1 and str2
//Output: Longest common subsequence of str1 and str2

    m ← length(str1) , n ← length(str2)

    matrix [m+1][n+1]

    for i ←1 to m+1 :
        for j ←1 to n+1 :
            if str1 [i-1] == str2 [j-1]:
                matrix [i][j] ←matrix [i-1][j-1] +1
            else
                matrix [i][j] ←max(matrix[i-1] [j],
                                   matrix[i] [j-1])

    index ←dp[m][n]
     lcs ← ["", "", .... index]
    i←m , j←n

    while i > 0 and j > 0 :
        if str1[i-1] == str2[j-1]:
            lcs[index-1] ← str1 [i-1]
            i-- , j-- , index--
        else if matrix[i-1][j] > matrix[i][j-1]:
            i--
        else
            j--

    return lcs.join("")
```

Test cases:



Time Complexity:

Time Complexity:

LCS (str1, str2)

Input: Two strings

Input size: m and n are lengths are strings

Basic operation: Filling the matrix and tracing back to find LCS using for loop

Let $C_{worst}(m,n)$ be worst case time complexity for the given algorithm

1) For filling matrix,

$$C_{worst}(m,n) = \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} 1$$

$$= \sum_{i=1}^{m+1} n+1-1+1$$

$$= \sum_{i=1}^{m+1} n+1$$

$$= (n+1)[n+1-1+1]$$

$$= (n+1)(m+1)$$

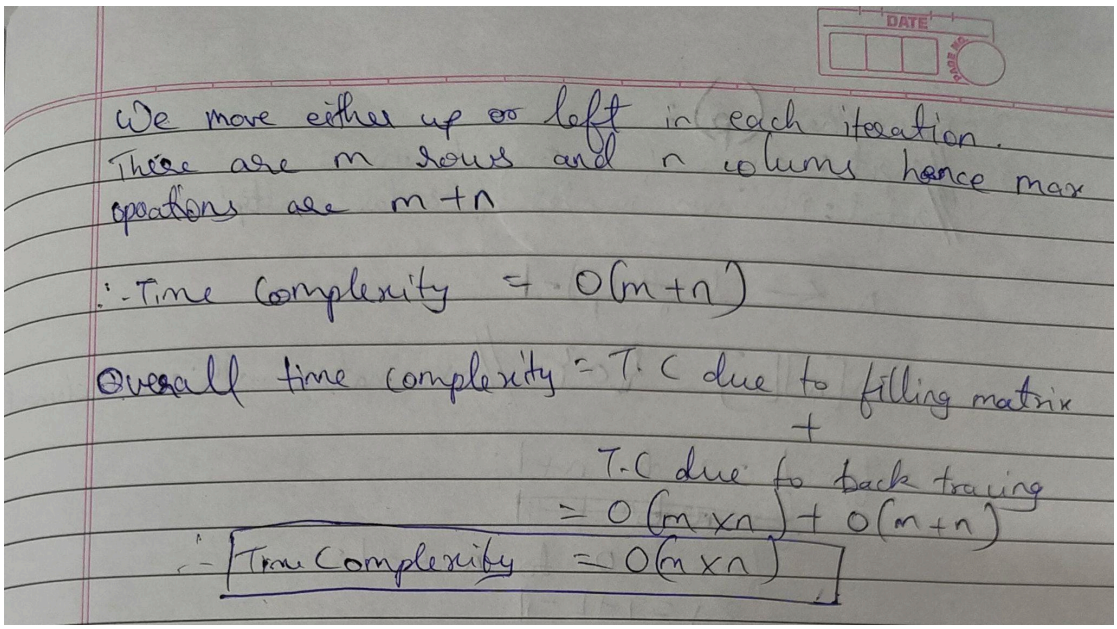
$$C_{worst}(m,n) \approx m \times n$$

∴ Time complexity = $O(m \times n)$

2) for tracing back

The backtracking process runs from (m, n) to (0, 0) taking a path by either moving diagonally or moving left/up.

When characters do not match, we move up/left. In this case, the maximum operations can take place.



Program:

```
class StringValidator:
    """Validates strings to ensure they meet criteria for LCS
computation."""

    @staticmethod
    def validate_non_empty(str1: str, str2: str) -> bool:
        """Checks if both strings are non-empty.

        Args:
            str1 (str): The first input string.
            str2 (str): The second input string.

        Returns:
            bool: True if both strings are non-empty, False otherwise.
        """
        return bool(str1) and bool(str2)

    @staticmethod
    def validate_alphabetic(str1: str, str2: str) -> bool:
        """Checks if both strings contain only alphabetic characters.

        Args:
            str1 (str): The first input string.
            str2 (str): The second input string.
```

```
        Returns:
            bool: True if both strings contain only alphabetic
characters.
        """
        return str1.isalpha() and str2.isalpha()

    @classmethod
    def validate(cls, str1: str, str2: str) -> str:
        """Runs all validation checks on the input strings.

        Args:
            str1 (str): The first input string.
            str2 (str): The second input string.

        Returns:
            str: An error message if validation fails, or None if
validation
                 passes.
        """
        if not cls.validate_non_empty(str1, str2):
            return "Error: One or both strings are empty."
        if not cls.validate_alphabetic(str1, str2):
            return "Error: Strings must contain only alphabets."
        return None


class LCSFinder:
    """Computes the Longest Common Subsequence (LCS) of two validated
strings."""

    def __init__(self, str1: str, str2: str):
        """Initializes the LCSFinder with two strings.

        Args:
            str1 (str): The first input string.
            str2 (str): The second input string.
        """
        self.str1 = str1
        self.str2 = str2
        self.m, self.n = len(str1), len(str2)
        self.dp_table = self._initialize_dp_table()
```

```
    def _initialize_dp_table(self) -> list:
        """Initializes a 2D DP table with dimensions (m+1) x (n+1).

        Returns:
            list: A 2D list initialized to zero.
        """
        return [[0] * (self.n + 1) for _ in range(self.m + 1)]

    def calculate_lcs_table(self) -> None:
        """Fills the DP table based on LCS dynamic programming rules."""
        for i in range(1, self.m + 1):
            for j in range(1, self.n + 1):
                if self.str1[i - 1] == self.str2[j - 1]:
                    self.dp_table[i][j] = self.dp_table[i - 1][j - 1] + 1
                else:
                    self.dp_table[i][j] = max(self.dp_table[i - 1][j],
                                              self.dp_table[i][j - 1])

    def construct_lcs(self) -> str:
        """Backtracks the DP table to construct the LCS string.

        Returns:
            str: The longest common subsequence.
        """
        i, j = self.m, self.n
        lcs_length = self.dp_table[self.m][self.n]
        lcs = [""] * lcs_length

        while i > 0 and j > 0:
            if self.str1[i - 1] == self.str2[j - 1]:
                lcs[lcs_length - 1] = self.str1[i - 1]
                i -= 1
                j -= 1
                lcs_length -= 1
            elif self.dp_table[i - 1][j] > self.dp_table[i][j - 1]:
                i -= 1
            else:
                j -= 1

        return "".join(lcs)
```

```
    def get_lcs(self) -> str:
        """Calculates and returns the longest common subsequence.

        Returns:
            str: The longest common subsequence.
        """
        self.calculate_lcs_table()
        return self.construct_lcs()


def longest_common_subsequence(str1: str, str2: str) -> str:
    """High-level function to validate input and compute the LCS.

    Args:
        str1 (str): The first input string.
        str2 (str): The second input string.

    Returns:
        str: The longest common subsequence, or an error message if
validation
             fails.
    """
    validation_error = StringValidator.validate(str1, str2)
    if validation_error:
        return validation_error

    lcs_finder = LCSFinder(str1, str2)
    return lcs_finder.get_lcs()


def run_lcs_tests() -> None:
    """Runs predefined test cases for the longest_common_subsequence
function."""
    test_cases = [
        # Test case 1: Common subsequence in non-trivial strings
        ("AABCCDBBBCCCABABDDDD", "AAAAABABBBBBABABAAAB"),

        # Test case 2: One string is empty
        ("", "AAAAABABBBBBABABAAAB"),
```

```
        # Test case 3: Both strings are empty
        ("", ""),

        # Test case 4: Strings with non-alphabetic characters
        ("1235", "23478"),

        # Test case 5: Typical case with mixed subsequences
        ("AABBBBBCCCAABBBC", "BCBCABCDFFRRFF"),

        # Test case 6: Known LCS in two similar strings
        ("BBBCBCCCBCABABBB", "BBBCCCABABAABB")
    ]

    for i, (str1, str2) in enumerate(test_cases):
        result = longest_common_subsequence(str1, str2)
        print(f"Test case {i + 1} result: {result}\n")


# Run the test cases
run_lcs_tests()
```

Output:

```
Test case 1 result: AABBBBABAB

Test case 2 result: Error: One or both strings are empty.

Test case 3 result: Error: One or both strings are empty.

Test case 4 result: Error: Strings must contain only alphabets.

Test case 5 result: BCCABC

Test case 6 result: BBBCCCABABBB
```

Conclusion: In conclusion, we have studied the 5 SOLID principles of software development and implemented it in the program of longest common subsequence. We have implemented the program of LCS using dynamic programming to optimize the time complexity of the algortihm