DAA Lab - Practical - 6

Name : Rishabh Jain

Roll No : A-54

Aim :Implement Longest Common Subsequence (LCS) algorithm to find the length and LCS for DNA sequences.

Problem Statement:

DNA sequences can be viewed as strings of A, C, G, and T characters, which represent nucleotides. Finding the similarities between two DNA sequences are an important computation performed in bioinformatics. [Note that a subsequence might not include consecutive elements of the original sequence.]

TASK-1: Find the similarity between the given X and Y sequence.

X=AGCCCTAAGGGCTACCTAGCTT Y= GACAGCCTACAAGCGTTAGCTTG Output: Cost matrix with all costs, final cost of LCS.

TASK-2: A subsequence of a given sequence is palindrome if it reads the same when read from left to right or right to left. Design an algorithm that take a sequence X[1...n].

Find all the possible palindrome sub-sequences for the given DNA sequence A C G T G T C A A A A T C G

Part A (Practical 6 DAA)

```
class LCS:
    def __init__(self):
        self.val = 0
        self.dir = "H"

def longest_common_string(s1, s2):
    n = len(s1)
    m = len(s2)

    dp = [[LCS() for _ in range(m + 1)] for _ in range(n + 1)]

    for i in range(n + 1):
        for j in range(m + 1):
            if i == 0 or j == 0:
                dp[i][j].val = 0
                dp[i][j].dir = "H"
            elif s1[i - 1] != s2[j - 1]:
                if dp[i - 1][j].val >= dp[i][j - 1].val:
                    dp[i][j].val = dp[i - 1][j].val
                    dp[i][j].dir = "U"
                else:
                    dp[i][j].val = dp[i][j - 1].val
                    dp[i][j].dir = "S"
            elif s1[i - 1] == s2[j - 1]:
                dp[i][j].val = dp[i - 1][j - 1].val + 1
                dp[i][j].dir = "D"

    for i in range(n + 1):
        for j in range(m + 1):
            print(f"{dp[i][j].val}/{dp[i][j].dir}", end=" ")
        print()

    return dp[n][m].val

if __name__ == "__main__":
    s1 = "AGCCCTAAGGGCTACCTAGCTT"
    s2 = "GACAGCCTACAAGCGTTAGCTTG"

    # s1 = "POLYNOMIAL"
    # s2 = "EXPONENTIAL"

    len = longest_common_string(s1, s2)
    print("Length:", len)
```

```
0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H 0/H
0/H 0/U 1/D 1/S 1/D 1/S 1/S 1/S 1/S 1/D 1/S 1/D 1/D 1/S 1/S 1/S 1/S 1/S 1/D 1/S 1/S 1/S 1/S 1/S 1/S
0/H 1/D 1/U 1/U 1/U 2/D 2/S 2/S 2/S 2/S 2/S 2/S 2/D 2/S 2/D 2/S 2/S 2/S 2/D 2/S 2/S 2/S 2/D
0/H 1/U 1/U 2/D 2/S 2/U 3/D 3/D 3/S 3/S 3/D 3/S 3/S 3/S 3/D 3/S 3/S 3/S 3/S 3/S 3/D 3/S 3/S 3/S
0/H 1/U 1/U 2/D 2/U 2/U 3/D 4/D 4/S 4/S 4/D 4/S 4/S 4/S 4/D 4/S 4/S 4/S 4/S 4/S 4/D 4/S 4/S 4/S
0/H 1/U 1/U 2/D 2/U 2/U 3/D 4/D 4/U 4/U 5/D 5/S 5/S 5/S 5/D 5/S 5/S 5/S 5/S 5/S 5/D 5/S 5/S 5/S
0/H 1/U 1/U 2/U 2/U 2/U 3/U 4/U 5/D 5/S 5/U 5/U 5/U 5/U 5/U 5/U 6/D 6/D 6/S 6/S 6/S 6/D 6/D 6/S
0/H 1/U 2/D 2/U 3/D 3/S 3/U 4/U 5/U 6/D 6/S 6/D 6/D 6/S 6/S 6/S 6/U 6/U 7/D 7/S 7/S 7/S 7/S 7/S
0/H 1/U 2/D 2/U 3/D 3/U 3/U 4/U 5/U 6/D 6/U 7/D 7/D 7/S 7/S 7/S 7/S 7/D 7/U 7/U 7/U 7/U 7/U
0/H 1/D 2/U 2/U 3/U 4/D 4/S 4/U 5/U 6/U 6/U 7/U 7/U 8/D 8/S 8/D 8/S 8/S 8/S 8/D 8/S 8/S 8/S 8/D
0/H 1/D 2/U 2/U 3/U 4/D 4/U 4/U 5/U 6/U 6/U 7/U 7/U 8/D 8/U 9/D 9/S 9/S 9/S 9/D 9/S 9/S 9/S 9/D
0/H 1/D 2/U 2/U 3/U 4/D 4/U 4/U 5/U 6/U 6/U 7/U 7/U 8/D 8/U 9/D 9/U 9/U 9/U 10/D 10/S 10/S 10/S 10/D
0/H 1/U 2/U 3/D 3/U 4/U 5/D 5/D 5/U 6/U 7/D 7/U 7/U 8/U 9/D 9/U 9/U 9/U 9/U 10/U 11/D 11/S 11/S 11/S
0/H 1/U 2/U 3/U 3/U 4/U 5/U 5/U 6/D 6/U 7/U 7/U 7/U 8/U 9/U 9/U 10/D 10/D 10/S 10/U 11/U 12/D 12/D 12/S
0/H 1/U 2/D 3/U 4/D 4/U 5/U 5/U 6/U 7/D 7/U 8/D 8/D 8/U 9/U 9/U 10/U 10/U 11/D 11/S 11/U 12/U 12/U 12/U
0/H 1/U 2/U 3/D 4/U 4/U 5/D 6/D 6/U 7/U 8/D 8/U 8/U 8/U 9/D 9/U 10/U 10/U 11/U 11/U 12/D 12/U 12/U 12/U
0/H 1/U 2/U 3/D 4/U 4/U 5/D 6/D 6/U 7/U 8/D 8/U 8/U 8/U 9/D 9/U 10/U 10/U 11/U 11/U 12/D 12/U 12/U 12/U
0/H 1/U 2/U 3/U 4/U 4/U 5/U 6/U 7/D 7/U 8/U 8/U 8/U 8/U 9/U 9/U 10/D 11/D 11/U 11/U 12/U 13/D 13/D 13/S
0/H 1/U 2/D 3/U 4/D 4/U 5/U 6/U 7/U 8/D 8/U 9/D 9/D 9/S 9/U 9/U 10/U 11/U 12/D 12/S 12/U 13/U 13/U 13/U
0/H 1/D 2/U 3/U 4/U 5/D 5/U 6/U 7/U 8/U 8/U 9/U 9/U 10/D 10/S 10/D 10/U 11/U 12/U 13/D 13/S 13/U 13/U 14/D
0/H 1/U 2/U 3/D 4/U 5/U 6/D 6/D 7/U 8/U 9/D 9/U 9/U 10/U 11/D 11/S 11/S 11/U 12/U 13/U 14/D 14/S 14/S 14/U
0/H 1/U 2/U 3/U 4/U 5/U 6/U 6/U 7/D 8/U 9/U 9/U 9/U 10/U 11/U 11/U 12/D 12/D 12/U 13/U 14/U 15/D 15/D 15/S
0/H 1/U 2/U 3/U 4/U 5/U 6/U 6/U 7/D 8/U 9/U 9/U 9/U 10/U 11/U 11/U 12/D 13/D 13/S 13/U 14/U 15/D 16/D 16/S
Length: 16
```

Part B (Practical 6 DAA)

```
def is_palindrome(subseq):
    return subseq == subseq[::-1]

def generate_palindrome_subsequences(seq, start, current, palindrome_subsequences):
    if start == len(seq):
        if is_palindrome(current) and len(current) > 1:
            palindrome_subsequences.append(current)
        return

    generate_palindrome_subsequences(seq, start + 1, current + seq[start], palindrome_subsequences)

    generate_palindrome_subsequences(seq, start + 1, current, palindrome_subsequences)

sequence = "ACGTGTCAAATCG"
palindrome_subsequences = []
generate_palindrome_subsequences(sequence, 0, "", palindrome_subsequences)

unique_list = []
[unique_list.append(x) for x in palindrome_subsequences if x not in unique_list]

print(len(unique_list))
print(unique_list)
```

```
59
['ACGTGCA', 'ACGGCA', 'ACGCA', 'ACTGTCA', 'ACTTCA', 'ACTCA', 'ACCA', 'ACA', 'AGTGA', 'AGGA', 'AGA', 'ATGTA', 'ATTA', 'ATA', 'AAAA',
```