DESIGN AND ANALYSIS OF ALGORITHM

PRACTICAL-6

NAME: VEDANT BHUTADA

ROLL: 69

BATCH: A4

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

TASK-1

ITERATIVE

```
def print_matrix_with_direction(matrix, X, Y):
    m = len(X)
    n = len(Y)
    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0 or j == 0:
                print(f"{matrix[i][j]:>2}h", end=" ")
            else:
                if X[i - 1] == Y[j - 1]:
                    direction = "d"  # diagonal
                elif matrix[i - 1][j] >= matrix[i][j - 1]:
                    direction = "u"  # up
                else:
                    direction = "s"  # side
                print(f"{matrix[i][j]:>2}{direction}", end=" ")
        print()

def longest_common_subsequence(X, Y):
    m = len(X)
    n = len(Y)
    # Create a matrix to store the length of LCS for each subproblem
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    # Retrieve the LCS length from the matrix
    lcs_length = dp[m][n]

    # Construct the LCS string
    lcs = [""] * lcs_length
    i, j = m, n
    while i > 0 and j > 0:
        if X[i - 1] == Y[j - 1]:
            lcs[lcs_length - 1] = X[i - 1]
            i -= 1
            j -= 1
            lcs_length -= 1
        elif dp[i - 1][j] >= dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    return lcs, len(lcs), dp

X = "AGCCCTAAGGGCTACCTAGCTT"
Y = "GACAGCCTACAAGCGTTAGCTTG"
lcs, lcs_length, dp = longest_common_subsequence(X, Y)
```

```
print("LCS:", "".join(lcs))
print("LCS Length:", lcs_length)
print("Cost Matrix:")
print_matrix_with_direction(dp, X, Y)
print("Final Cost of LCS:", dp[len(X)][len(Y)])
```

```
LCS: AGCCCAAGGTTAGCTT
LCS Length: 16
Cost Matrix:
 0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h
 0h  0u  1d  1s  1d  1s  1s  1s  1s  1d  1s  1d  1d  1s  1s  1s  1s  1s  1d  1s  1s  1s  1s  1s
 0h  1d  1u  1u  1u  2d  2s  2s  2s  2s  2s  2s  2s  2d  2s  2d  2s  2s  2s  2d  2s  2s  2s  2d
 0h  1u  1u  2d  2s  2u  3d  3d  3s  3s  3d  3s  3s  3s  3d  3s  3s  3s  3s  3s  3d  3s  3s  3s
 0h  1u  1u  2d  2u  2u  3d  4d  4s  4s  4d  4s  4s  4s  4d  4s  4s  4s  4s  4s  4d  4s  4s  4s
 0h  1u  1u  2d  2u  2u  3d  4d  4u  4u  5d  5s  5s  5s  5d  5s  5s  5s  5s  5s  5d  5s  5s  5s
 0h  1u  1u  2u  2u  2u  3u  4u  5d  5s  5u  5u  5u  5u  5u  6d  6d  6s  6s  6s  6d  6d  6s
 0h  1u  2d  2u  3d  3s  3u  4u  5u  6d  6s  6d  6d  6s  6s  6s  6u  6u  7d  7s  7s  7s  7s  7s
 0h  1u  2d  2u  3d  3u  3u  4u  5u  6d  6u  7d  7d  7s  7s  7s  7s  7s  7d  7u  7u  7u  7u  7u
 0h  1d  2u  2u  3u  4d  4s  4u  5u  6u  6u  7u  7u  8d  8s  8d  8s  8s  8s  8d  8s  8s  8s  8d
 0h  1d  2u  2u  3u  4d  4u  4u  5u  6u  6u  7u  7u  8d  8u  9d  9s  9s  9s  9d  9s  9s  9s  9d
 0h  1d  2u  2u  3u  4d  4u  4u  5u  6u  6u  7u  7u  8d  8u  9d  9u  9u  9u 10d 10s 10s 10s 10d
 0h  1u  2u  3d  3u  4u  5d  5d  5u  6u  7d  7u  7u  8u  9d  9u  9u  9u  9u 10u 11d 11s 11s 11s
 0h  1u  2u  3u  3u  4u  5u  5u  6d  6u  7u  7u  7u  8u  9u  9u 10d 10d 10s 10u 11u 12d 12d 12s
 0h  1u  2d  3u  4d  4u  5u  5u  6u  7d  7u  8d  8d  8u  9u  9u 10u 10u 11d 11s 11u 12u 12u 12u
 0h  1u  2u  3d  4u  4u  5d  6d  6u  7u  8d  8u  8u  8u  9d  9u 10u 10u 11u 11u 12d 12u 12u 12u
 0h  1u  2u  3d  4u  4u  5d  6d  6u  7u  8d  8u  8u  8u  9d  9u 10u 10u 11u 11u 12d 12u 12u 12u
 0h  1u  2u  3u  4u  4u  5u  6u  7d  7u  8u  8u  8u  9u  9u 10d 11d 11u 11u 12u 13d 13d 13s
 0h  1u  2d  3u  4d  4u  5u  6u  7u  8d  8u  9d  9d  9s  9u  9u 10u 11u 12d 12s 12u 13u 13u 13u
 0h  1d  2u  3u  4u  5d  5u  6u  7u  8u  8u  9u  9u 10d 10s 10d 10u 11u 12u 13d 13s 13u 13u 14d
 0h  1u  2u  3d  4u  5u  6d  6d  7u  8u  9d  9u  9u 10u 11d 11s 11s 11u 12u 13u 14d 14s 14s 14u
 0h  1u  2u  3u  4u  5u  6u  6u  7d  8u  9u  9u  9u 10u 11u 11u 12d 12d 12u 13u 14u 15d 15d 15s
 0h  1u  2u  3u  4u  5u  6u  6u  7d  8u  9u  9u  9u 10u 11u 11u 12d 13d 13s 13u 14u 15d 16d 16s
Final Cost of LCS: 16
```

RECURSIVE

```
def print_matrix_with_direction(matrix, X, Y):
    m = len(X)
    n = len(Y)
    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0 or j == 0:
                print(f"{matrix[i][j]:>2}h", end=" ")
            else:
                if X[i - 1] == Y[j - 1]:
                    direction = "d"  # diagonal
                elif matrix[i - 1][j] >= matrix[i][j - 1]:
                    direction = "u"  # up
                else:
                    direction = "s"  # side
                print(f"{matrix[i][j]:>2}{direction}", end=" ")
        print()

def longest_common_subsequence(X, Y):
    m = len(X)
    n = len(Y)

    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    lcs_length = dp[m][n]

    def construct_lcs(i, j):
        if i == 0 or j == 0:
            return []
        if X[i - 1] == Y[j - 1]:
            lcs = construct_lcs(i - 1, j - 1)
            lcs.append(X[i - 1])
            return lcs
        if dp[i - 1][j] >= dp[i][j - 1]:
            return construct_lcs(i - 1, j)
        else:
            return construct_lcs(i, j - 1)

    lcs = construct_lcs(m, n)
```

```
    return lcs, lcs_length, dp

X = "AGCCCTAAGGGCTACCTAGCTT"
Y = "GACAGCCTACAAGCGTTAGCTTG"
lcs, lcs_length, dp = longest_common_subsequence(X, Y)

print("LCS:", "".join(lcs))
print("LCS Length:", lcs_length)
print("Cost Matrix:")
print_matrix_with_direction(dp, X, Y)
print("Final Cost of LCS:", dp[len(X)][len(Y)])
```

```
LCS: AGCCCAAGGTTAGCTT
LCS Length: 16
Cost Matrix:
 0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h  0h
 0h  0u  1d  1s  1d  1s  1s  1s  1s  1d  1s  1d  1d  1s  1s  1s  1s  1s  1d  1s  1s  1s  1s  1s
 0h  1d  1u  1u  1u  2d  2s  2s  2s  2s  2s  2s  2s  2d  2s  2d  2s  2s  2s  2d  2s  2s  2s  2d
 0h  1u  1u  2d  2s  2u  3d  3d  3s  3s  3d  3s  3s  3s  3d  3s  3s  3s  3s  3s  3d  3s  3s  3s
 0h  1u  1u  2d  2u  2u  3d  4d  4s  4s  4d  4s  4s  4s  4d  4s  4s  4s  4s  4s  4d  4s  4s  4s
 0h  1u  1u  2d  2u  2u  3d  4d  4u  4u  5d  5s  5s  5s  5d  5s  5s  5s  5s  5s  5d  5s  5s  5s
 0h  1u  1u  2u  2u  2u  3u  4u  5d  5s  5u  5u  5u  5u  5u  6d  6d  6s  6s  6s  6d  6d  6s
 0h  1u  2d  2u  3d  3s  3u  4u  5u  6d  6s  6d  6d  6s  6s  6s  6u  6u  7d  7s  7s  7s  7s  7s
 0h  1u  2d  2u  3d  3u  3u  4u  5u  6d  6u  7d  7d  7s  7s  7s  7s  7s  7d  7u  7u  7u  7u  7u
 0h  1d  2u  2u  3u  4d  4s  4u  5u  6u  6u  7u  7u  8d  8s  8d  8s  8s  8s  8d  8s  8s  8s  8d
 0h  1d  2u  2u  3u  4d  4u  4u  5u  6u  6u  7u  7u  8d  8u  9d  9s  9s  9s  9d  9s  9s  9s  9d
 0h  1d  2u  2u  3u  4d  4u  4u  5u  6u  6u  7u  7u  8d  8u  9d  9u  9u  9u 10d 10s 10s 10s 10d
 0h  1u  2u  3d  3u  4u  5d  5d  5u  6u  7d  7u  7u  8u  9d  9u  9u  9u  9u 10u 11d 11s 11s 11s
 0h  1u  2u  3u  3u  4u  5u  5u  6d  6u  7u  7u  7u  8u  9u  9u 10d 10d 10s 10u 11u 12d 12d 12s
 0h  1u  2d  3u  4d  4u  5u  5u  6u  7d  7u  8d  8d  8u  9u  9u 10u 10u 11d 11s 11u 12u 12u 12u
 0h  1u  2u  3d  4u  4u  5d  6d  6u  7u  8d  8u  8u  8u  9d  9u 10u 10u 11u 11u 12d 12u 12u 12u
 0h  1u  2u  3d  4u  4u  5d  6d  6u  7u  8d  8u  8u  8u  9d  9u 10u 10u 11u 11u 12d 12u 12u 12u
 0h  1u  2u  3u  4u  4u  5u  6u  7d  7u  8u  8u  8u  8u  9u  9u 10d 11d 11u 11u 12u 13d 13d 13s
 0h  1u  2d  3u  4d  4u  5u  6u  7u  8d  8u  9d  9d  9s  9u  9u 10u 11u 12d 12s 12u 13u 13u 13u
 0h  1d  2u  3u  4u  5d  5u  6u  7u  8u  8u  9u  9u 10d 10s 10d 10u 11u 12u 13d 13s 13u 13u 14d
 0h  1u  2u  3d  4u  5u  6d  6d  7u  8u  9d  9u  9u 10u 11d 11s 11s 11u 12u 13u 14d 14s 14s 14u
 0h  1u  2u  3u  4u  5u  6u  6u  7d  8u  9u  9u  9u 10u 11u 11u 12d 12d 12u 13u 14u 15d 15d 15s
 0h  1u  2u  3u  4u  5u  6u  6u  7d  8u  9u  9u  9u 10u 11u 11u 12d 13d 13s 13u 14u 15d 16d 16s
Final Cost of LCS: 16
```

TASK-2

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

def find_palindrome_subsequences(sequence):
    def backtrack(start, path):
        if len(path) >= 2 and is_palindrome(path):
            palindromic_subsequences.add(path)
        if start == len(sequence):
            return

        for i in range(start, len(sequence)):
            path += sequence[i]
            backtrack(i + 1, path)
            path = path[:-1]

    palindromic_subsequences = set()
    backtrack(0, "")

    return list(palindromic_subsequences)

sequence = "ACGTGTCAAATCG"
palindrome_subsequences = find_palindrome_subsequences(sequence)
print("Palindromic Subsequences:")
for subsequence in palindrome_subsequences:
    print(subsequence)
print("Total Number of Palindromic Subsequences:", len(palindrome_subsequences))
```

```
TTT
CGC
GCG
CTATC
ATGTA
GCAAACG
TAAAT
GAG
AA
ACTCA
```

GTTG
TCT
GTCTG
TT
ACGTGCA
GTATG
GTTTG
ACGCA
AGA
CTGTC
CTTTC
CAAC
GG
GCAACG
GCCG
ATA
GCTCG
CGGC
TAT
GTG
CTC
GAAG
CGTGC
AGGA
GAAAG
AGTGA
GTAATG
ACTTCA
AAAA
GTAAATG
ATTA
CTTC
CC
CAAAC
GCACG
GGG
AAA
CTAATC
ACA
CAC
TGT
TAAT
ACCA
CCC
Total Number of Palindromic Subsequences: 59