

Practical – 5

Name: Mohammad Anas Ajaz

Batch: B4

Roll Number: 61

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

Problem Statement:

(i) 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 and direction, final cost of LCS and the LCS.

Length of LCS=16

CODE:

```
def LCS(x, y):  
    m, n = len(x), len(y)  
  
    # Initialize Cost Matrix (c) and direction matrices (b)  
    c = [[0] * (n + 1) for _ in range(m + 1)]  
    b = [[""] * (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]:
                c[i][j] = c[i - 1][j - 1] + 1
                b[i][j] = "D"
            elif c[i - 1][j] >= c[i][j - 1]:
                c[i][j] = c[i - 1][j]
                b[i][j] = "U"
            else:
                c[i][j] = c[i][j - 1]
                b[i][j] = "L"
    return c, b

def get_LCS(b, x, i, j):
    """Backtrack to find the LCS string"""
    if i == 0 or j == 0:
        return ""
    if b[i][j] == "D":
        return get_LCS(b, x, i - 1, j - 1) + x[i - 1]
    elif b[i][j] == "U":
        return get_LCS(b, x, i - 1, j)
    else:
        return get_LCS(b, x, i, j - 1)

# Input sequences
X = "AGCCCTAAGGGCTACCTAGCTT"
Y = "GACAGCCTACAAGCGTTAGCTTG"

c, b = LCS(X, Y)
lcs_string = get_LCS(b, X, len(X), len(Y))

print("Final Cost of LCS:", c[len(X)][len(Y)])
print("LCS String:", lcs_string)

# Optional: print cost matrix
print("\nCost Matrix:")
for row in c:
    print(row)

# Printing Direction Matrix
print("\nDirection Matrix:")
for row in b:
    print(row)

```

Code Screenshot:

```
Task-1.py > LCS
1 def LCS(x, y):
2     m, n = len(x), len(y)
3
4     # Initialize Cost Matrix (c) and direction matrices (b)
5     c = [[0] * (n + 1) for _ in range(m + 1)]
6     b = [[""] * (n + 1) for _ in range(m + 1)]
7
8     for i in range(1, m + 1):
9         for j in range(1, n + 1):
10             if x[i - 1] == y[j - 1]:
11                 c[i][j] = c[i - 1][j - 1] + 1
12                 b[i][j] = "D"
13             elif c[i - 1][j] >= c[i][j - 1]:
14                 c[i][j] = c[i - 1][j]
15                 b[i][j] = "U"
16             else:
17                 c[i][j] = c[i][j - 1]
18                 b[i][j] = "L"
19     return c, b
```

```
20
21
22 def get_LCS(b, x, i, j):
23     """Backtrack to find the LCS string"""
24     if i == 0 or j == 0:
25         return ""
26     if b[i][j] == "D":
27         return get_LCS(b, x, i - 1, j - 1) + x[i - 1]
28     elif b[i][j] == "U":
29         return get_LCS(b, x, i - 1, j)
30     else:
31         return get_LCS(b, x, i, j - 1)
32
33
```

```
34 # Input sequences
35 x = "AGCCCTAAGGGCTACCTAGCTT"
36 y = "GACAGCCTACAAGCGTTAGCTTG"
37
38 c, b = LCS(x, y)
39 lcs_string = get_LCS(b, x, len(x), len(y))
40
41 print("Final Cost of LCS:", c[len(x)][len(y)])
42 print("LCS String:", lcs_string)
43
44 # Optional: print cost matrix
45 print("\nCost Matrix:")
46 for row in c:
47     print(row)
48
49 # Printing Direction Matrix
50 print("\nDirection Matrix:")
51 for row in b:
52     print(row)
53
```

Code Output:

```
[Running] python -u "c:\Users\Krish\OneDrive\Desktop\RBU\RBU-Sem-3\LABS\DESIGN ALGORITHM ANALYSIS\Practical-5\Task-1.p
Final Cost of LCS: 16
LCS String: AGCCCAAGGTAGCTT
```

Cost Matrix:

[0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0	, 0]
[0	, 0	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1	, 1]
[0	, 1	, 1	, 1	, 1	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2	, 2]
[0	, 1	, 1	, 2	, 2	, 2	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3	, 3]
[0	, 1	, 1	, 2	, 2	, 2	, 3	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4	, 4]
[0	, 1	, 1	, 2	, 2	, 2	, 3	, 4	, 4	, 4	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5	, 5]
[0	, 1	, 1	, 2	, 2	, 2	, 3	, 4	, 5	, 5	, 5	, 5	, 5	, 5	, 6	, 6	, 6	, 6	, 6	, 6	, 6	, 6	, 6]
[0	, 1	, 2	, 2	, 2	, 3	, 3	, 3	, 4	, 5	, 6	, 6	, 6	, 6	, 6	, 6	, 7	, 7	, 7	, 7	, 7	, 7	, 7]
[0	, 1	, 2	, 2	, 3	, 3	, 3	, 4	, 5	, 6	, 6	, 7	, 7	, 7	, 7	, 7	, 7	, 7	, 7	, 7	, 7	, 7	, 7]
[0	, 1	, 2	, 2	, 3	, 4	, 4	, 4	, 5	, 6	, 6	, 7	, 7	, 8	, 8	, 8	, 8	, 8	, 8	, 8	, 8	, 8	, 8]
[0	, 1	, 2	, 2	, 3	, 4	, 4	, 4	, 5	, 6	, 6	, 7	, 7	, 8	, 9	, 9	, 9	, 9	, 9	, 9	, 9	, 9	, 9]
[0	, 1	, 2	, 2	, 3	, 4	, 4	, 4	, 5	, 6	, 6	, 7	, 7	, 8	, 8	, 9	, 9	, 9	, 10	, 10	, 10	, 10	, 10]
[0	, 1	, 2	, 2	, 3	, 4	, 5	, 5	, 6	, 7	, 7	, 7	, 8	, 9	, 9	, 9	, 9	, 9	, 10	, 11	, 11	, 11	, 11]
[0	, 1	, 2	, 3	, 3	, 4	, 5	, 5	, 6	, 6	, 7	, 7	, 7	, 8	, 9	, 9	, 10	, 10	, 10	, 10	, 11	, 12	, 12]
[0	, 1	, 2	, 3	, 4	, 4	, 5	, 5	, 6	, 7	, 7	, 8	, 8	, 8	, 9	, 9	, 10	, 10	, 11	, 11	, 11	, 12	, 12]
[0	, 1	, 2	, 3	, 4	, 4	, 5	, 6	, 6	, 7	, 8	, 8	, 8	, 8	, 9	, 9	, 10	, 10	, 11	, 11	, 12	, 12	, 12]
[0	, 1	, 2	, 3	, 4	, 4	, 5	, 6	, 6	, 7	, 8	, 8	, 8	, 8	, 9	, 9	, 10	, 10	, 11	, 11	, 12	, 12	, 12]
[0	, 1	, 2	, 3	, 4	, 4	, 5	, 6	, 7	, 7	, 8	, 8	, 8	, 8	, 9	, 9	, 10	, 11	, 11	, 11	, 12	, 13	, 13]
[0	, 1	, 2	, 3	, 4	, 4	, 5	, 6	, 7	, 8	, 8	, 9	, 9	, 9	, 9	, 9	, 10	, 11	, 12	, 12	, 12	, 13	, 13]
[0	, 1	, 2	, 3	, 4	, 5	, 5	, 6	, 7	, 8	, 8	, 9	, 9	, 10	, 10	, 10	, 10	, 11	, 12	, 13	, 13	, 13	, 14]
[0	, 1	, 2	, 3	, 4	, 5	, 6	, 6	, 7	, 8	, 9	, 9	, 9	, 10	, 11	, 11	, 11	, 11	, 12	, 13	, 14	, 14	, 14]
[0	, 1	, 2	, 3	, 4	, 5	, 6	, 6	, 7	, 8	, 9	, 9	, 9	, 10	, 11	, 11	, 12	, 12	, 12	, 13	, 14	, 15	, 15]
[0	, 1	, 2	, 3	, 4	, 5	, 6	, 6	, 7	, 8	, 9	, 9	, 9	, 10	, 11	, 11	, 12	, 13	, 13	, 13	, 14	, 15	, 16]

```
[Done] exited with code=0 in 0.084 seconds
```

TASK-2: Find the longest repeating subsequence (LRS). Consider it as a variation of the longest common subsequence (LCS) problem. Let the given string be S. You need to find the LRS within S. To use the LCS framework, you effectively compare S with itself. So, consider string₁ = S and string₂ = S.

Example:

AABCBDC

LRS= ABC or ABD

CODE:

```
def LRS(s):
    n = len(s)
    # Step 1: Initialize DP table
    c = [[0] * (n + 1) for _ in range(n + 1)]

    # Step 2: Fill DP table
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if s[i - 1] == s[j - 1] and i != j:
                c[i][j] = 1 + c[i - 1][j - 1] # diagonal
            else:
                c[i][j] = max(c[i - 1][j], c[i][j - 1]) # top or left

    # Step 3: Backtracking to reconstruct LRS
    i, j = n, n
    lrs_seq = []
    while i > 0 and j > 0:
        if s[i - 1] == s[j - 1] and i != j:
            lrs_seq.append(s[i - 1])
            i -= 1
            j -= 1
        elif c[i - 1][j] >= c[i][j - 1]:
            i -= 1
        else:
            j -= 1

    return c, c[n][n], "".join(reversed(lrs_seq))

# Example Run
S = "AABEBDCDD"
dp, length, lrs_str = LRS(S)

print("LRS Length:", length)
print("LRS:", lrs_str)
```

```

print("\nDP Matrix:")
for row in dp:
    print(row)

```

Code Screenshot:

```

Task-2.py > ...
1  def LRS(s):
2      n = len(s)
3      # Step 1: Initialize DP table
4      c = [[0] * (n + 1) for _ in range(n + 1)]
5
6      # Step 2: Fill DP table
7      for i in range(1, n + 1):
8          for j in range(1, n + 1):
9              if s[i - 1] == s[j - 1] and i != j:
10                 c[i][j] = 1 + c[i - 1][j - 1] # diagonal
11             else:
12                 c[i][j] = max(c[i - 1][j], c[i][j - 1]) # top or left
13
14     # Step 3: Backtracking to reconstruct LRS
15     i, j = n, n
16     lrs_seq = []
17     while i > 0 and j > 0:
18         if s[i - 1] == s[j - 1] and i != j:
19             lrs_seq.append(s[i - 1])
20             i -= 1
21             j -= 1
22         elif c[i - 1][j] >= c[i][j - 1]:
23             i -= 1
24         else:
25             j -= 1
26
27     return c, c[n][n], "".join(reversed(lrs_seq))
28
29
30 # Example Run
31 S = "AABEBCDD"
32 dp, length, lrs_str = LRS(S)
33
34 print("LRS Length:", length)
35 print("LRS:", lrs_str)
36
37 print("\nDP Matrix:")
38 for row in dp:
39     print(row)
40

```

Code Output:

```
[Running] python -u "c:\Users\Krish\OneDrive\Desktop\RBU\RBU-Sem-3\LAB
LRS Length: 3
LRS: ABD
```

DP Matrix:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 2, 2, 2, 2]
[0, 1, 1, 1, 1, 2, 2, 2, 2]
[0, 1, 1, 2, 2, 2, 2, 2, 2]
[0, 1, 1, 2, 2, 2, 2, 2, 2]
[0, 1, 1, 2, 2, 2, 2, 2, 3]
[0, 1, 1, 2, 2, 2, 2, 3, 3]
```

```
[Done] exited with code=0 in 0.099 seconds
```

Leetcode Question:

linke : <https://leetcode.com/problems/longest-common-subsequence/submissions/1803160931/>

The screenshot displays the LeetCode interface for the 'Longest Common Subsequence' problem. The submission status is 'Accepted' with 47/47 testcases passed. The user's submission, 'anas_sk', was submitted on Oct 16, 2025, at 13:17. The performance metrics show a runtime of 527 ms (Beats 74.61%) and a memory usage of 33.86 MB (Beats 33.41%). A bar chart illustrates the distribution of runtime times, with the user's submission highlighted at 527 ms. The code editor on the right shows a Python solution using dynamic programming.

```
class Solution(object):
    def longestCommonSubsequence(self, text1, text2):
        m, n = len(text1), len(text2)
        dp = [[0] * (n + 1) for _ in range(m + 1)]

        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if text1[i - 1] == text2[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + 1
                else:
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

        return dp[m][n]
```

Problem List

Accepted

Editorial

Solutions

Submissions

1143. Longest Common Subsequence

Solved

MediumTopicsCompaniesHint

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"

PythonAuto

```
1 class Solution(object):
2     def longestCommonSubsequence(self, text1, text2):
3         m, n = len(text1), len(text2)
4         dp = [[0] * (n + 1) for _ in range(m + 1)]
5
6         for i in range(1, m + 1):
7             for j in range(1, n + 1):
8                 if text1[i - 1] == text2[j - 1]:
9                     dp[i][j] = dp[i - 1][j - 1] + 1
10                else:
11                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
12
13         return dp[m][n]
14
```

SavedLn 13, Col 10

TestcaseTest Result

14.7K242128 Online

