

PRACTICAL NO. : 05

Name : Ranveer M. Bhortekar

Section/Batch : A4/B1

Roll No. : 06

Subject : DAA

Leet Code ID : [Bhorteckar Coder - LeetCode Profile](#)

AIM : Implement a dynamic algorithm for Longest Common Subsequence (LCS) 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

CODE IN TEXT FORMAT :

#TASK 1 : LCS

X="AGCCCTAAGGGCTACCTAGCTT"

Y="GACAGCCTACAAGCGTTAGCTTG"

```
def lcs(X,Y):
```

```
    m,n=len(X),len(Y)
```

```

c = [[0]*n for _ in range(m)]
b = [['']*n for _ in range(m)]

for i in range(m):
    for j in range(n):
        if X[i] == Y[j]:
            c[i][j] = c[i-1][j-1] + 1 if i > 0 and j > 0 else 1
            b[i][j] = '↖'
        elif i > 0 and (j == 0 or c[i-1][j] >= c[i][j-1]):
            c[i][j] = c[i-1][j]
            b[i][j] = '↑'
        elif j > 0:
            c[i][j] = c[i][j-1]
            b[i][j] = '←'
        else:
            c[i][j] = 0
            b[i][j] = ""

return b, c

# Print LCS()

def print_lcs(b, X, i, j):
    if i < 0 or j < 0:
        return ""
    if b[i][j] == '↖':
        return print_lcs(b, X, i-1, j-1) + X[i]
    elif b[i][j] == '↑':
        return print_lcs(b, X, i-1, j)
    else:
        return

```

```
return print_lcs(b, X, i, j-1)
```

```
b, c = lcs(X, Y)
```

```
lcs_string = print_lcs(b, X, len(X)-1, len(Y)-1)
```

```
print("Longest Common Subsequence:", lcs_string)
```

```
print("Length of LCS:", len(lcs_string))
```

CODE/OUTPUT SCREENSHOT :

The screenshot shows a Jupyter Notebook interface with two code cells.

Cell 1:

```
#TASK 1 : LCS
X="AGCCCTAACGGGCTACCTAGCTT"
Y="GACAGCCTACAAGCGTTAGCTTG"

def lcs(X,Y):
    m,n=len(X),len(Y)
    c=[[0]*n for _ in range(m)]
    b=[[None]*n for _ in range(m)]

    for i in range(m):
        for j in range(n):
            if X[i] == Y[j]:
                c[i][j] = c[i-1][j-1] + 1 if i > 0 and j > 0 else 1
                b[i][j] = '^'
            elif i > 0 and (j == 0 or c[i-1][j] >= c[i][j-1]):
                c[i][j] = c[i-1][j]
                b[i][j] = 't'
            elif j > 0:
                c[i][j] = c[i][j-1]
                b[i][j] = 'v'
            else:
                c[i][j] = 0
                b[i][j] = '**'

    return b, c
```

Cell 2:

```
def print_lcs(b, X, i, j):
    if i < 0 or j < 0:
        return ''
    if b[i][j] == '^':
        return print_lcs(b, X, i-1, j-1) + X[i]
    elif b[i][j] == 't':
        return print_lcs(b, X, i-1, j)
    else:
        return print_lcs(b, X, i, j-1)

b, c = lcs(X, Y)
lcs_string = print_lcs(b, X, len(X)-1, len(Y)-1)

print("Longest Common Subsequence:", lcs_string)
print("Length of LCS:", len(lcs_string))
```

The output pane shows the results of the execution:

```
Longest Common Subsequence: AGCCAAGGTTAGCTT
Length of LCS: 16
```

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 IN TEXT FORMAT :

TASK 2 : LRS

```
def LRS(a):
    n = len(a)
    c = [[0]*(n+1) for _ in range(n+1)]
    for i in range(1,n+1):
        for j in range(1,n+1):
            if a[i-1]==a[j-1] and i!=j:
                c[i][j]=1+c[i-1][j-1]
            else:
                c[i][j]=max(c[i-1][j],c[i][j-1])
    i, j = n, n
    lrs_string = ""
    while i>0 and j>0:
        if a[i-1]==a[j-1] and i!=j:
            lrs_string=a[i-1]+lrs_string
            i-=1
            j-=1
        elif c[i-1][j]>c[i][j-1]:
            i-=1
        else:
            j-=1
    return lrs_string
```

```
else:
```

```
j-=1
```

```
return c[n][n], lrs_string
```

```
s="AABCBDC"
```

```
length, LRS_string = LRS(s)
```

```
print("Length of Longest Repeated Subsequence:", length)
```

```
print("Longest Repeated Subsequence:", LRS_string)
```

CODE/OUTPUT SCREENSHOT :

The screenshot shows a Jupyter Notebook interface with the following details:

- File:** RANVEER B. A4_B1_06 DAA PRAC 05.ipynb
- Code Cell Content:**

```
# TASK 2 : LRS
def LRS(a):
    n = len(a)
    c = [[0]*(n+1) for _ in range(n+1)]
    for i in range(1,n+1):
        for j in range(1,n+1):
            if a[i-1]==a[j-1] and i!=j:
                c[i][j]=1+c[i-1][j-1]
            else:
                c[i][j]=max(c[i-1][j],c[i][j-1])
    i, j = n, n
    lrs_string = ""
    while i>0 and j>0:
        if a[i-1]==a[j-1] and i!=j:
            lrs_string=a[i-1]+lrs_string
            i-=1
            j-=1
        elif c[i-1][j]>c[i][j-1]:
            i-=1
        else:
            j-=1
    return c[n][n], lrs_string

s="AABCBDC"
length, LRS_string = LRS(s)
print("Length of Longest Repeated Subsequence:", length)
print("Longest Repeated Subsequence:", LRS_string)
```
- Output Cell Content:**

```
Length of LCS: 16
Length of Longest Repeated Subsequence: 3
Longest Repeated Subsequence: ABC
```

LEETCODE SUBMISSION : [Longest Common Subsequence - LeetCode](#)

Problem Statement : ID – 1143.Longest Common Subsequence

PROBLEM STATEMENT SCREENSHOT :

1143. Longest Common Subsequence

Medium Topics Companies Hint

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.

CODE IN TEXT FORMAT :

```
int longestCommonSubsequence(char *text1, char *text2) {

    int n = strlen(text1), m = strlen(text2);

    int dp[n+1][m+1];

    for(int i=0;i<=n;i++) {
        for(int j=0;j<=m;j++) {
            if(i==0 || j==0)
                dp[i][j] = 0;
            else 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[n][m];
}
```

```

dp[i][j] = 1 + dp[i-1][j-1];

else

    dp[i][j] = dp[i-1][j] > dp[i][j-1] ? dp[i-1][j] : dp[i][j-1];

}

return dp[n][m];
}

```

CODE/SUBMISSION SCREENSHOT :

The screenshot shows a LeetCode submission page for the "Longest Common Subsequence" problem. The code is a C implementation of a dynamic programming solution. It uses a 2D array `dp` where `dp[i][j]` represents the length of the longest common subsequence of the first `i` characters of `text1` and the first `j` characters of `text2`. The code initializes `dp[0][0] = 0` and iterates through both strings, updating the DP table based on whether characters at the current positions match or not. The runtime is 23 ms (68.50% beats) and memory usage is 12.15 MB (79.71% beats). The code editor shows the submitted code, and the test case section shows inputs "abcde" and "ace" with expected output "ace".

```

int longestCommonSubsequence(char *text1, char *text2) {
    int n = strlen(text1), m = strlen(text2);
    int dp[n+1][m+1];

    for(int i=0;i<n;i++) {
        for(int j=0;j<m;j++) {
            if(i==0 || j==0)
                dp[i][j] = 0;
            else if(text1[i-1] == text2[j-1])
                dp[i][j] = 1 + dp[i-1][j-1];
            else
                dp[i][j] = dp[i-1][j] > dp[i][j-1] ? dp[i-1][j] : dp[i][j-1];
        }
    }
    return dp[n][m];
}

```