

PRACTICAL NO. 5

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 and direction, final cost of LCS and the LCS.

CODE:-

```
import java.util.Scanner;

class C{
    int v;
    char d;

    C(){
        v = 0;
        d = 'h';
    }
}

public class Lcs {
    public static void findlcs(String a , String b){
        int m = a.length();
        int n = b.length();
        D[][]cost = new D[m+1][n+1];
        for(int i =0 ; i<=m ; i++){
            for(int j = 0 ; j<= n ;j++){
                cost[i][j] = new D();
            }
        }
        for(int i = 1; i<=m ; i++){
            for(int j = 1; j<=n ; j++){
                if(a.charAt(i-1) == b.charAt(j-1)){
                    cost[i][j].v = cost[i-1][j-1].v+1;
                    cost[i][j].d = 'd';
                }
                else {
                    if (cost[i - 1][j].v >= cost[i][j - 1].v) {
                        cost[i][j].v = cost[i - 1][j].v;
                    }
                }
            }
        }
    }
}
```

```

        cost[i][j].d = 'u';
    } else {
        cost[i][j].v = cost[i][j - 1].v;
        cost[i][j].d = 's';
    }
}

}

}

System.out.println("LONGEST COMMON SUBSEQUENCE IS:- ");
Printlcs(m , n , cost , a);
System.out.println();
System.out.println("LENGHT OF THE LCS IS:- "+cost[m][n].v);
}

public static void Printlcs(int i , int j , D[][]cost , String a){
    if(i == 0 || j==0){
        return;
    }
    else{
        if(cost[i][j].d == 'd'){
            Printlcs(i-1 , j-1 , cost , a);
            System.out.print(a.charAt(i-1));
        } else if (cost[i][j].d == 'u') {
            Printlcs(i-1 , j , cost , a);
        }
        else{
            Printlcs(i , j-1 , cost , a);
        }
    }
}

}

public static void main(String[] args) {
    Scanner S = new Scanner(System.in);
    System.out.println("ENTER STRING 1:- ");
    String a = S.next();
    System.out.println("ENTER STRING 2:- ");
    String b = S.next();

    findlcs(a,b);
}
}

```

OUTPUT:-

```

"C:\Program Files\Java\jdk-23\bin\java.exe
ENTER STRING 1:-
AGCCCTAAGGGCTACCTAGCTT
ENTER STRING 2:-
GACAGCCTACAAGCGTTAGCTTG
LONGEST COMMON SUBSEQUENCE IS:-
AGCCCAAGGTTAGCTT
LENGHT OF THE LCS IS:- 16

Process finished with exit code 0
|

```

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:-

```
import java.util.Scanner;

class D{
    int v;
    char d;

    D(){
        v = 0;
        d = 'h';
    }
}

public class Lrs {
    public static void findlrs(String a){
        int m = a.length();
        int n = a.length();
        D[][]cost = new D[m+1][n+1];
        for(int i =0 ; i<=m ; i++){
            for(int j = 0 ; j<= n ; j++){
                cost[i][j] = new D();
            }
        }
        for(int i = 1; i<=m ; i++){
            for(int j = 1; j<=n ; j++){
                if(a.charAt(i-1) == a.charAt(j-1)&&i!=j){
                    cost[i][j].v = cost[i-1][j-1].v+1;
                    cost[i][j].d = 'd';
                }
                else {
                    if (cost[i - 1][j].v >= cost[i][j - 1].v) {
                        cost[i][j].v = cost[i - 1][j].v;
                        cost[i][j].d = 'u';
                    } else {
                        cost[i][j].v = cost[i][j - 1].v;
                        cost[i][j].d = 's';
                    }
                }
            }
        }
        System.out.println("LONGEST REPEATING SUBSEQUENCE IS:- ");
        Printlrs(m , n , cost , a);
        System.out.println();
        System.out.println("LENGHT OF THE LRS IS:- "+cost[m][n].v);
    }
}
```

```
}  
public static void Printlrs(int i , int j , D[][]cost , String a){  
    if(i == 0 || j==0){  
        return;  
    }  
    else{  
        if(cost[i][j].d == 'd'){  
            Printlrs(i-1 , j-1 , cost , a);  
            System.out.print(a.charAt(i-1));  
        } else if (cost[i][j].d == 'u') {  
            Printlrs(i-1 , j , cost , a);  
        }  
        else{  
            Printlrs(i , j-1 , cost , a);  
        }  
    }  
}  
  
public static void main(String[] args) {  
    Scanner S = new Scanner(System.in);  
    System.out.println("ENTER STRING 1:- ");  
    String a = S.next();  
  
    findlrs(a);  
}  
}
```

OUTPUT:-

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-java  
ENTER STRING 1:-  
AABCBDC  
LONGEST REPEATING SUBSEQUENCE IS:-  
ABC  
LENGHT OF THE LRS IS:- 3  
  
Process finished with exit code 0  
|
```

NAME:- DIPANSH TIWARI

CLASS:- A4

ROLL NO. 35

PRACTICAL 05

LeetCode Assessment:

The screenshot displays the LeetCode submission interface for a problem. The top navigation bar includes links for Problem List, Accepted, Editorial, Solutions, and Submissions. The submission status is 'Accepted' with 47/47 testcases passed, submitted by 'XmVqgSKLL' on Sep 16, 2025 at 14:22. The runtime is 485 ms (Beats 48.41%) and memory is 42.82 MB (Beats 57.19%). A bar chart shows the runtime distribution across various time intervals. The code is written in Python3 and implements a dynamic programming solution for the Longest Common Subsequence problem. The test result shows the input text2 = "ace" and the output is 3, which matches the expected result.

Runtime: 485 ms | Beats 48.41%
Memory: 42.82 MB | Beats 57.19%

Code:

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

Testcase: text2 = "ace"
Output: 3
Expected: 3