

Practical No. 5

Name: Rana Chavan

Section: A4 – B2

Roll no. 26

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

Task 1:

CODE:

```
public class LCS_Task1 {
    public static void main(String[] args) {
        String X = "AGCCCTAAGGGCTACCTAGCTT";
        String Y = "GACAGCCTACAAGCGTTAGCTTG";

        int m = X.length();
        int n = Y.length();
        int[][] dp = new int[m + 1][n + 1];

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (X.charAt(i - 1) == Y.charAt(j - 1))
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                else
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
            }
        }

        System.out.println("LCS COST MATRIX:");
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                System.out.print(dp[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("Length of LCS = " + dp[m][n]);

        int i = m, j = n;
        String lcs = "";
        while (i > 0 && j > 0) {
            if (X.charAt(i - 1) == Y.charAt(j - 1)) {
                lcs = X.charAt(i - 1) + lcs;
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1])
                i--;
            else
                j--;
        }

        System.out.println("LCS = " + lcs);
    }
}
```

OUTPUT:

Task 2:

CODE:

```
public class LRS_Task2 {
    public static void main(String[] args) {
        String S = "AABCBDC";
        int n = S.length();
        int[][] dp = new int[n + 1][n + 1];

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (S.charAt(i - 1) == S.charAt(j - 1) && i != j)
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                else
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
}
```

```

        System.out.println("LRS MATRIX:");
        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= n; j++) {
                System.out.print(dp[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("Length of LRS = " + dp[n][n]);

        int i = n, j = n;
        String lrs = "";
        while (i > 0 && j > 0) {
            if (S.charAt(i - 1) == S.charAt(j - 1) && i != j) {
                lrs = S.charAt(i - 1) + lrs;
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1])
                i--;
            else
                j--;
        }

        System.out.println("LRS = " + lrs);
    }
}

```

OUTPUT:

The screenshot shows the IntelliJ IDEA run window for a project named 'DAA lab'. The current run configuration is 'Run - DAA lab' with 'LRS_Task2' selected. The output pane displays the following text:

```

Run - DAA lab
Run  LRS_Task2 ×
C:\Users\ranac\.jdks\openjdk-24.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
↑   LRS MATRIX:
↓   0 0 0 0 0 0
→   0 0 1 1 1 1
←   0 1 1 1 1 1
☰   0 1 1 1 2 2
📄   0 1 1 1 2 2
📝   0 1 1 2 2 2
📄   0 1 1 2 2 2
📝   0 1 1 2 3 3
Length of LRS = 3
LRS = ABC

Process finished with exit code 0

```

LeetCode Assesment:

The screenshot shows a LeetCode submission page for a Java solution. The submission was accepted with 47/47 testcases passed and a time taken of 53 ms. The code implements a dynamic programming approach to find the longest common subsequence between two strings, `text1` and `text2`. The runtime chart shows a distribution of execution times, with the vast majority of calls taking between 3ms and 23ms, with a notable peak around 18ms. The memory usage is 51.12 MB, which is 31.99% faster than other submissions. The code editor displays the Java code, and the test results show a successful run for the test case "ace".

```
1 class Solution {
2     public int longestCommonSubsequence(String text1, String text2) {
3         char [] A = text1.toCharArray();
4         char [] B = text2.toCharArray();
5         int m = A.length;
6         int n = B.length;
7         int[][] dp = new int[m+1][n+1];
8
9         for (int i = 1; i <= m; i++) {
10            for (int j = 1; j <= n; j++) {
11                if(A[i-1] == B[j-1]){
12                    dp[i][j] = 1 + dp[i-1][j-1];
13                }
14                else{
15                    dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]);
16                }
17            }
18        }
19        return dp[m][n];
20    }
21 }
```

Profile Link: <https://leetcode.com/u/ranachavan/>

Submission detail: <https://leetcode.com/submissions/detail/1803509703/>