**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:- ");

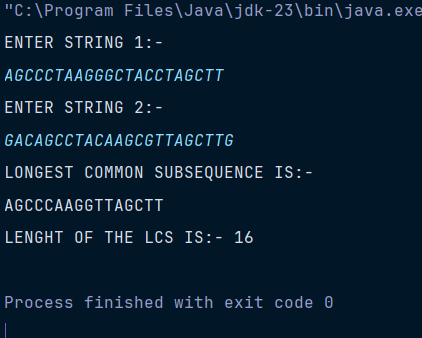
String b = S.next();

*findlcs*(a,b);

}

}

OUTPUT:-



**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:- ");

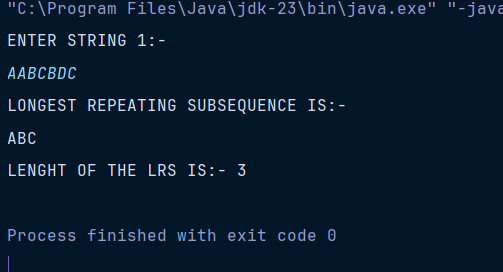
String a = S.next();

*findlrs*(a);

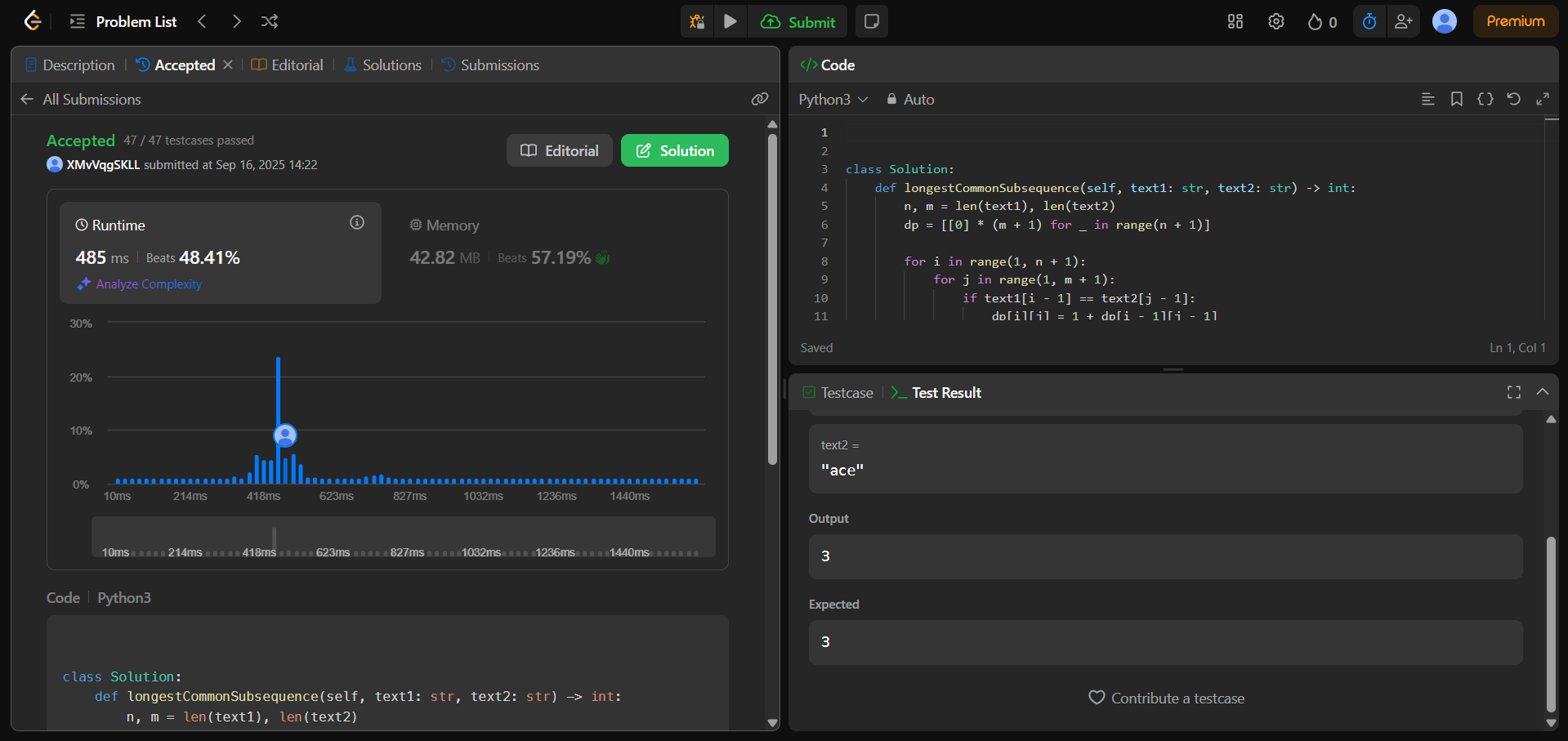
}

}

OUTPUT:-



**LeetCode Assessment:**

****