DP-S016 (E043)

Solved Challenges **0/1**

Back To Challenges List

Minimum Edit Distance Two Strings

ID:11156 **Solved By 476 Users**

The program must accept two string values **S1** and **S2** as the input. The program must print the minimum cost required to convert the string S1 to S2 as the output. The cost of insertion, deletion and substitution of any character in the string S1 will be **1**.

Boundary Condition(s):
1 <= Length of S1, S2 <= 1000

Input Format:
The first line contains S1.
The second line contains S2.

Output Format:
The first line contains the minimum cost required to convert the string S1 to S2.

Example Input/Output 1:
Input:
```
hello
hail
```

Output:
```
3
```

Explanation:
Here S1 = **hello** and S2 = **hail**
The minimum cost required to convert the string **hello** to **hail** is **3**.
The 3 operations are given below.
1 - **a** is substituted in place of **e**. Now the string S1 becomes **hallo**.
2 - **i** is substituted in place of **l** (first occurring l). Now the string S1 becomes **hailo**.
3 - **o** is deleted. Now the string S1 becomes **hail**.

Example Input/Output 2:
Input:
intrinsic
intrusive

Output:
4

Max Execution Time Limit: 500 millisecs

Ambiance

Java (12.0)

Reset

```java
import java.util.*;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s1 = sc.next();
        String s2 = sc.next();

        int matrix[][] = new int[s2.length()+1][s1.length()+1];

        for(int col = 0;col<=s1.length();col++)
            matrix[0][col]=col;
        for(int row = 0;row<=s2.length();row++)
            matrix[row][0]=row;

        for(int row=1;row<=s2.length();row++)
        {
            for(int col=1;col<=s1.length();col++)
            {
                if(s1.charAt(col-1) == s2.charAt(row-1))
                {
                    matrix[row][col] = matrix[row-1][col-1];
                }
                else
                {
                    int left = matrix[row][col-1];
                    int top = matrix[row-1][col];
                    int topleft = matrix[row-1][col-1];
                    matrix[row][col] = Math.min(topleft, Math.min(left,top)) + 1;
                }
            }
        }

        System.out.println(matrix[s2.length()][s1.length()]);

    }
}
```

Code did not pass the execution

TestCase ID: 64347

Input:

```
hello
hail
```

Expected Output:

```
3
```

Your Program Output:

```
0
```

Save Run

☐ Run with a custom test case (Input/Output)