**Edit Distance**

Given two strings **s1** and **s2**of lengths**m**and**n**respectively and below operations that can be performed on **s1**. Find the minimum number of edits (operations) to convert ‘**s1**‘into ‘**s2**‘.

* **Insert**: Insert any character before or after any index of **s1**
* **Remove:**Remove a character of **s1**
* **Replace:**Replace a character at any index of **s1** with some other character.

**Note:**All of the above operations are of equal cost.

**Examples:**

***Input:*** *s1 = “geek”, s2 = “gesek”****Output:*** *1****Explanation:*** *We can convert s1 into s2 by inserting a ‘s’ between two consecutive ‘e’ in s2.*

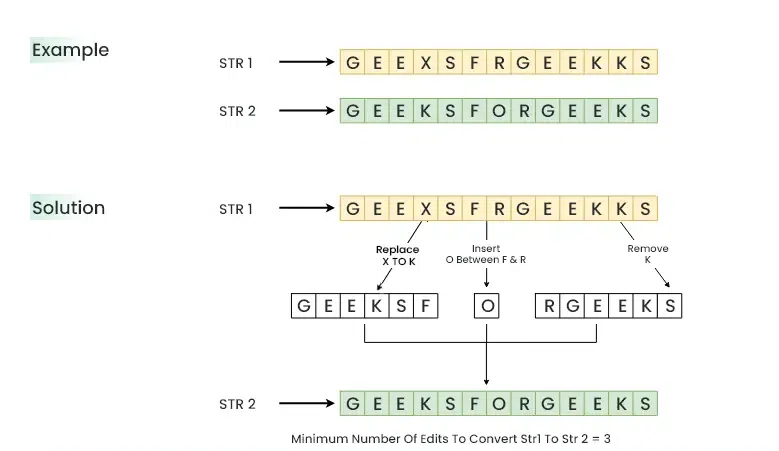
***Input:*** *s1 = “cat”, s2 = “cut”****Output:*** *1****Explanation:*** *We can convert s1 into s2 by replacing ‘a’ with ‘u’.*

***Input:*** *s1 = “sunday”, s2 = “saturday”****Output:*** *3****Explanation:*** *Last three and first characters are same.  We basically need to convert “un” to “atur”.  This can be done using below three operations. Replace ‘n’ with ‘r’, insert t, insert a*

**Illustration of Edit Distance:**

*Let’s suppose we have* ***s1=”GEEXSFRGEEKKS”*** *and* ***s2=”GEEKSFORGEEKS”***  *Now to convert* ***s1*** *into* ***s2*** *we would require* ***3*** *minimum operations:  
Operation 1: Replace ‘****X****‘ to ‘****K****‘   
Operation 2: Insert ‘****O****‘ between ‘****F****‘ and ‘****R****‘   
Operation 3: Remove second last character i.e. ‘****K****‘*

*Refer the below image for better understanding.*

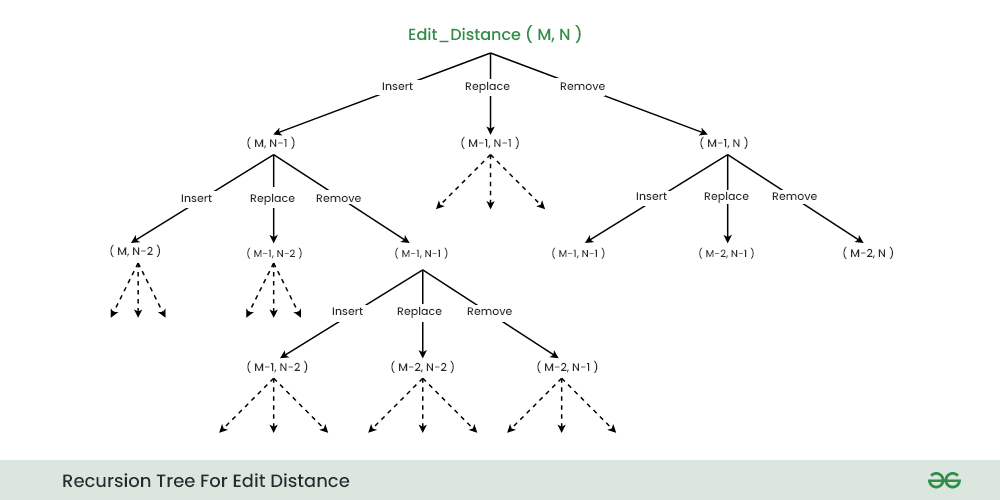
**

**Using**Recursion**:**

**Sub problems in Edit Distance:**

*The idea is to process all characters one by one starting from either from* ***left*** *or* ***right*** *sides of both strings.   
Let us process from the* ***right end*** *of the strings, there are two possibilities for every pair of characters being traversed, either they* ***match*** *or they* ***don’t match****. If last characters of both string matches, then we simply recursively calculate the* ***answer*** *for rest of part of the strings. When last characters do not match, we perform all three operations to match the last characters, i.e.* ***insert, replace, and remove.*** *We then recursively calculate the result for the remaining part of the string. Upon completion of these operations, we will select the minimum* ***answer*** *and add 1 to it.*

*Below is the* ***recursive tree*** *for this problem considering the case when the last characters never match.*

**

*When the last characters of strings matches. Make a recursive call* ***editDist(m-1, n-1)*** *to calculate the answer for remaining part of the strings.*

*When the last characters of strings don’t match. Make three recursive calls as show below:*

* *Insert s2[n-1] at last of s1 : editDist(m, n-1)*
* *Replace s1[m-1] with s2[n-1] : editDist(m-1, n-1)*
* *Remove s1[m-1] : editDist(m-1, n)*

**Recurrence Relations for Edit Distance:**

* ***Case 1****: When the last character of both the strings are same.* ***editDist(s1, s2, m, n) = editDist(s1, s2, m-1, n-1)***
* ***Case 2****: When the last characters are different*
  + ***editDist(s1, s2, m, n) = 1 + Minimum{ editDist(s1, s2 ,m,n-1), editDist(s1, s2 ,m-1, n), editDist(s1, s2 , m-1, n-1)}***

**Base Case for Edit Distance:**

* ***Case 1****: When* ***s1*** *becomes empty i.e. m****=0***
  + ***return n****, as it require n insertions to convert an empty string to* ***s2*** *of size* ***n***
* ***Case 2****: When s2 becomes empty i.e. n****=0***
  + ***return m****, as it require m removals to convert s1 of size* ***m*** *to an empty string.*