Example: The Levenshtein distance

Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'.

A. Insert
B. Remove
C. Replace

All of the above operations are of equal cost.

Examples:

Input: str1 = "geek", str2 = "gesek"

Output: 1

We can convert str1 into str2 by inserting a 's'.

Input: str1 = "cat", str2 = "cut"

Output: 1

We can convert str1 into str2 by replacing 'a' with 'u'.

Solution:

$$dp[i][j] = \begin{cases} MaxLength(str1(i),str2(j)) & i=0 \text{ OR } j=0 \\ dp[i-1][j-1] & str1[i] = str2[j] \\ \min \begin{cases} dp[i][j-1] + 1 & \text{// Insert} \\ dp[i-1][j] + 1 & str1[i] \neq str2[j] \quad \text{// Remove} \\ dp[i-1][j-1] + 1 & \text{// Replace} \end{cases} \end{cases}$$

Example 1

abcd abckd

Str1 = < k i t t e n > str2 = < s i t t i n g >

i

```
EditDistance(str1, str2):

   n = length(str1)
   m = length(str2)
   Create dp[0..n][0..m]
   for j from 0 to m:                  // لو 1: قاعدة str1 إدخال كل حروف = الحل فاضية، str2
      dp[0][j] = j
   for i from 0 to n:                  //    // لو 2: قاعدة str2 حذف كل حروف = الحل فاضية، str1
      dp[i][0] = i
   // تعبئة الجدول
   for i from 1 to n:
      for j from 1 to m:
         if str1[i] == str2[j]:
            dp[i][j] = dp[i-1][j-1]      // No operation
         else
            insertCost  = dp[i][j-1] + 1   // Insert
            removeCost  = dp[i-1][j] + 1   // Remove
            replaceCost = dp[i-1][j-1] + 1 // Replace

            dp[i][j] = min(insertCost, removeCost, replaceCost)
   return dp[n][m]
```

Example: Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.

Example: Input: [10,9,2,5,3,7,101,18] Output: 4 Explanation: The longest increasing subsequence is [2,3,7,101],

therefore the length is 4. Note:

There may be more than one LIS combination, it is only necessary for you to return the length.

Solution:

Initial Value LIS[i] = 1

$$LIS[i] = \begin{cases} \max(LIS[j] + 1, LIS[i]) & , arr[j] < arr[i] \\ j+1 & \\ & \\ j+1 & , arr[j] \geq arr[i] \\ i+1, j = 0 & , j + 1 = i \end{cases}$$

```
LIS_Length(arr):
    n = length(arr)
    Create array LIS[0..n-1]
     // 1 كل عنصر يبدأ بـ
     for i from 0 to n-1:
       LIS[i] = 1

     // مقارنة كل عنصر مع ما قبله
         for i from 1 to n-1:
            for j from 0 to i-1
               if arr[j] < arr[i]:      // الشرط للسلسلة التصاعدية
                  LIS[i] = max( LIS[i], LIS[j] + 1 )

  return max value in LIS[]
```