

Minimum Deletions & Insertions to Transform a String into another

We'll cover the following



- Problem Statement
- Solution
- Bottom-up Dynamic Programming

Problem Statement

Given strings s_1 and s_2 , we need to transform s_1 into s_2 by deleting and inserting characters. Write a function to calculate the count of the minimum number of deletion and insertion operations.

Example 1:

```
Input: s1 = "abc"
       s2 = "fbc"
Output: 1 deletion and 1 insertion.
Explanation: We need to delete {'a'} and insert {'f'} to s1 to transform it into s2.
```

Example 2:

```
Input: s1 = "abdca"
       s2 = "cbda"
Output: 2 deletions and 1 insertion.
Explanation: We need to delete {'a', 'c'} and insert {'c'} to s1 to transform it into s2.
```

Example 3:

```
Input: s1 = "passport"
       s2 = "ppsspt"
Output: 3 deletions and 1 insertion
Explanation: We need to delete {'a', 'o', 'r'} and insert {'p'} to s1 to transform it into s2.
```

Solution

This problem can easily be converted to the Longest Common Subsequence

(<https://www.educative.io/collection/page/5668639101419520/5633779737559040/5657535201673216/>) (LCS). If we can find the LCS of the two input strings, we can easily find how many





characters we need to insert and delete from s1. Here is how we can do this:

1. Let's assume `len1` is the length of `s1` and `len2` is the length of `s2`.
2. Now let's assume `c1` is the length of LCS of the two strings `s1` and `s2`.
3. To transform `s1` into `s2`, we need to delete everything from `s1` which is not part of LCS, so minimum deletions we need to perform from `s1` => `len1 - c1`
4. Similarly, we need to insert everything in `s1` which is present in `s2` but not part of LCS, so minimum insertions we need to perform in `s1` => `len2 - c1`





Let's jump directly to the bottom-up dynamic programming solution:

Bottom-up Dynamic Programming

Here is the code for our bottom-up dynamic programming approach:

 Java	 JS	 Python3	 C++
----------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------

```
1 def find_MDI(s1, s2):
2     c1 = find_LCS_length(s1, s2)
3     print("Minimum deletions needed: " + str(len(s1) - c1))
4     print("Minimum insertions needed: " + str(len(s2) - c1))
5
6
7 def find_LCS_length(s1, s2):
8     n1, n2 = len(s1), len(s2)
9     dp = [[0 for _ in range(n2+1)] for _ in range(n1+1)]
10    maxLength = 0
11    for i in range(1, n1+1):
12        for j in range(1, n2+1):
13            if s1[i - 1] == s2[j - 1]:
14                dp[i][j] = 1 + dp[i - 1][j - 1]
15            else:
16                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
17
18        maxLength = max(maxLength, dp[i][j])
19
20    return maxLength
21
22
23 def main():
24     find_MDI("abc", "fbc")
25     find_MDI("abdca", "cbda")
26     find_MDI("passport", "ppsspt")
27
28
29 main()
30
```

The time and space complexity of the above algorithm is $O(m * n)$, where 'm' and 'n' are the lengths of the two input strings.



← Back

Next →

Longest Common Subsequence

Longest Increasing Subsequence

☒ Mark as Completed

🕒 Report an Issue

🔗 Ask a Question

(https://discuss.educative.io/tag/minimum-deletions--insertions-to-transform-a-string-into-another__pattern-5-longest-common-substring__grokking-dynamic-programming-patterns-for-coding-interviews)