**Minimum Deletions & Insertions to Transform a String into another**

**WE'LL COVER THE FOLLOWING**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#problem-statement)
    - [Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#solution)
    - [Bottom-up Dynamic Programming](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#bottom-up-dynamic-programming)

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#problem-statement)

Given strings s1 and s2, we need to transform s1 into s2 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**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#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**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O#bottom-up-dynamic-programming)

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

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27OkE8D08O)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

class MDI {

  public void findMDI(String s1, String s2) {

    int c1 = findLCSLength(s1, s2);

    System.out.println("Minimum deletions needed: " + (s1.length() - c1));

    System.out.println("Minimum insertions needed: " + (s2.length() - c1));

  }

  private int findLCSLength(String s1, String s2) {

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

    int maxLength = 0;

    for(int i=1; i <= s1.length(); i++) {

      for(int j=1; j <= s2.length(); j++) {

        if(s1.charAt(i-1) == s2.charAt(j-1))

          dp[i][j] = 1 + dp[i-1][j-1];

        else

          dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]);

        maxLength = Math.max(maxLength, dp[i][j]);

      }

    }

    return maxLength;

  }

  public static void main(String[] args) {

    MDI mdi = new MDI();

    mdi.findMDI("abc", "fbc");

    mdi.findMDI("abdca", "cbda");

    mdi.findMDI("passport", "ppsspt");

  }

}





RUN

SAVERESET

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

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8Pq4ZnBN0N)

Longest Common Subsequence

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/YQ0y0QOJQ69)

Longest Increasing Subsequence

Stuck? Get help on

[**DISCUSS**](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-5-longest-common-substring-minimum-deletions-and-insertions-to-transform-a-string-into-another)

Send feedback

8 Recommendations

Cannot edit in read-only editor (occurred 6 times)