**Edit Distance Problem**

*Find the edit distance between two strings.*

**Input:** Strings *s* and *t*.

**Output:** The edit distance between *s* and *t*.

In 1966, Vladimir Levenshtein introduced the notion of the *edit distance* between two strings as the minimum number of *edit operations* needed to transform one string into another. Here, an edit operation is the insertion, deletion, or substitution of a single symbol. For example, the edit distance between GACT and ATG is 3 (**G**ACT → A**C**T → AT → AT**G**).

**IInput Format.** The first line of the input contains a string *s*, and the second line of the input contains a string *t*.

**Output Format.** The edit distance between *s* and *t*.

**Constraints.** |*s*| ≤ 1,000; |*t*| ≤ 1,000

**SAMPLE DATASET:**

Input:

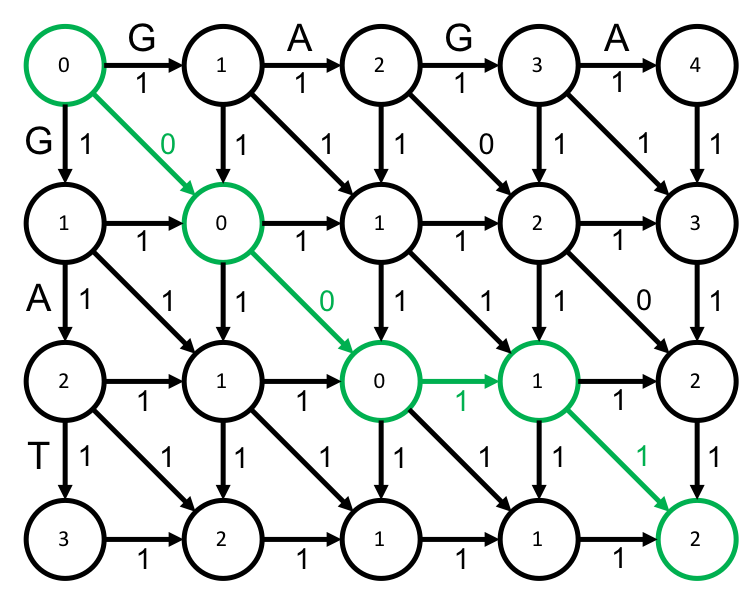
GAGA

GAT

Output:

2

The edit distance between the strings GAGA and GAT is 2 (GA**G**A → GA**A** → GAT). The Figure below represents the shortest transformation of one string to another (corresponding to the edit distance) as a green path in a graph.



**TEST DATASET 1:**

Input:

AC

AC

Output:

0

This test makes sure that your code doesn’t reward exact matches by adding a positive value to the edit distance. If two strings are exactly the same then their edit distance should be 0. It is easy to confuse edit distance with alignment, which could lead you to assign positive values to character matches in the dynamic programming matrix. When computing edit distance we only want to add to the edit distance when there is an indel or a mismatch. If your code outputs some multiple of 2 for this dataset it is likely that there is some mistake regarding the nature of edit distance computation. Alternatively your code could be finding *maximum* edit distance instead of *minimum* edit distance. This is an especially easy mistake to make when coming from alignment problems.

**TEST DATASET 2:**

Input:

AT

G

Output:

2

This test makes sure that your code correctly adds to the edit distance between the two strings when there are deletions or substitutions. Any sort of edit operation will add 1 to the edit distance between the two strings. If you are conflating alignment scores and edit distance it may be possible for you to come up with a negative result for this dataset. Don’t forget that all singular edit operations contribute exactly 1 to the edit distance between strings; don’t add different values to the edit distance for insertions, deletions, and substitutions.

**TEST DATASET 3:**

Input:

CAGACCGAGTTAG

CGG

Output:

10

This test makes sure that your code correctly handles inputs in which the strings to be compared drastically differ in length. If your output doesn’t match the correct output make sure that your implementation makes no assumptions about the length of the strings to be compared. Make sure that your dynamic programming matrix has dimensions or .

**TEST DATASET 4:**

Input:

CGT

CAGACGGTGACG

Output:

9

This test makes sure that your code correctly handles inputs in which the strings to be compared drastically differ in length. This test is similar to Test Dataset 3 except in this dataset string *s* is shorter than string *t*.