PROBLEM A: EDIT DISTANCE

The edit distance between two words—sometimes also called the *Levenshtein* distance—is the *minimum* number of letter insertions, letter deletions, and letter substitutions required to transform one word into another.

For example, the edit distance between FOOD and MONEY is at most four:

FOOD → MOOD → MON_D → MONED → MONEY

Given two strings, find the edit distance between them.

INPUT:

Line 1: the first string, A

Line 2: the second string, B

OUTPUT:

Edit distance between the two strings

1) We are transforming string A to string B. Assume that string A[0] .. A[i-1] have been transformed to be identical to B[0] .. B[j-1], and the consideration now is on A[i] and B[j].

   The table below lists all possible scenarios at state (i, j) and edit operations that can be performed. What is the consequential state for each combination of condition and operation ?

condition	edit operation	next state to consider
A[i] == B[j]	None	
A[i] != B[j]	Insert B[j] in front of A[i]	
A[i] != B[j]	Delete A[i]	
A[i] != B[j]	Change A[i] to B[j]	

2) What is the beginning state?

3) If A runs out, but B has not yet, in other words, i == len(A), but j < len(B), what is the additional edit distance required to complete the transformation?

4) If B runs out, but A has not yet, what is the additional edit distance required to complete the transformation?

5) Use the concepts obtained from step 1 to 4 above in write a recursive brute-force solution for this problem. The zipped test case file is downloadable from Class Materials.

6) Given that a string can be up to 1000 letters long, improve the brute-force solution so that the program will finish in no more than 2.5 seconds (CPU processing time).

PROBLEM B: FROG

Attempt to solve the problem at the following link: [Educational DP Contest AtCoder A - Frog 1 - DMOJ: Modern Online Judge]

With Python 3, using memoization technique will pass the first batch of test cases, but will exceed the recursion limit in the second batch, after having passed two test cases, as the following.

Execution Results

✔ ✖

Batch #1 (0/0 points)
Case #1: **AC** [0.022s,8.99 MB]
Case #2: **AC** [0.021s,9.00 MB]
Case #3: **AC** [0.021s,8.99 MB]

Batch #2 (0/100 points)
Case #1: **AC** [0.021s, 9.00 MB]
Case #2: **AC** [0.021s, 9.00 MB]
Case #3: **IR** (RecursionError) [0.208s,111.06 MB]
Case #4: —
Case #5: —
Case #6: —
Case #7: —
Case #8: —

Resources: 0.315s, 111.06 MB
Final score: 0/100 (0.0/7 points)

This recursion limitation can be overcome by utilizing “dynamic programming” technique, which will be studied later.