The Levenshtein distance, or Minimum Edit Distance, is a famous measure used to compare the distance between two strings. The Levenshtein distance is calculated by the number of Insertions, Deletions and Substitutions required to convert one string to another.

Example 1. Let the insertion, deletion and substitution cost all be 1. The Levenshtein distance between PRíCE and PRíNCES is 2. It requires two insertions – the N and S.

Example 2. The Levenshtein distance between PRíCE and RICH is 2. It requires a deletion of P and a substitution of E for H.

Note that a substitution is equivalent to an insertion and a deletion. Depending on the relative costs of these three operations, it may be more or less efficient to use the substitution operation.

The calculation of Levenshtein distance is most efficiently calculated using Dynamic Programming. The algorithm is described in the Class 13 Lecture Notes, and was discussed in class.

In this assignment, you are asked to calculate the Levenshtein distance between pairs of words. Words and costs will be input to your program via command line parameters. The words will contain only capitalized alphanumeric characters. The costs will be integers.

Your program should output the following information, separated by commas, and ending with an end of line character.

1. The first word
2. The second word
3. The total Levenshtein distance between each pair of two words
4. The number of insertions used in the minimum edit
5. The number of deletions used in the minimum edit
6. The number of substitutions used in the minimum edit

The program must compile using the following command on a CS network computer.
The program must run using the following command on a CS network computer.

```
./levenshtein word_one word_two insert_cost delete_cost substitution_cost
```

or

```
java levenshtein word_one word_two insert_cost delete_cost substitution_cost
```

Example 2 would be called as follows

```
./levenshtein PRINCE RICH 1 1 1
```

and should produce **only** the following output (ending with an end of line character)

```
PRINCE,RICH,3,0,2,1
```

This assignment should not require any file io operations.

**Deliverables:**

- **levenshtein.cc**, or **levenshtein.java**, A program to calculate the levenshtein distance between two strings. You may expect all input strings to be capitalized.

- A plain text README including 1) pseudocode for the calculation of the levenshtein distance between two strings, 2) a description of how your program should be called, and the output it produces, and, 3) any relevant details of the execution of your code.

**Grading:**

- **15 points - Compilation** The program file must compile without error or warning into an executable as described above.

- **15 points - Execution** The executable must run without error or warning on valid input using the command line parameters described above.

- **15 points - README** Does your README documentation completely and accurately describe the levenshtein distance measure and the algorithm used to calculate it?
• **10 points - Within Code Documentation** Is the code documented for obvious understanding of the use, preconditions, and postconditions of each function?

• **10 points - Style** Are functions and variables given self-explanatory names? Are functions used to aid intelligibility of the code? Are functions used to reduce repeated blocks of code? Is indentation, spacing, use of parentheses, use of braces consistent and sensible?

• **20 points - Correctness** Is the dynamic programming algorithm implemented correctly?

• **15 points - Automated Testing** There will be fifteen automated tests. Does the executable produce correct output for all fifteen inputs?