LING165 Lab #5: Spelling correction

1. Task

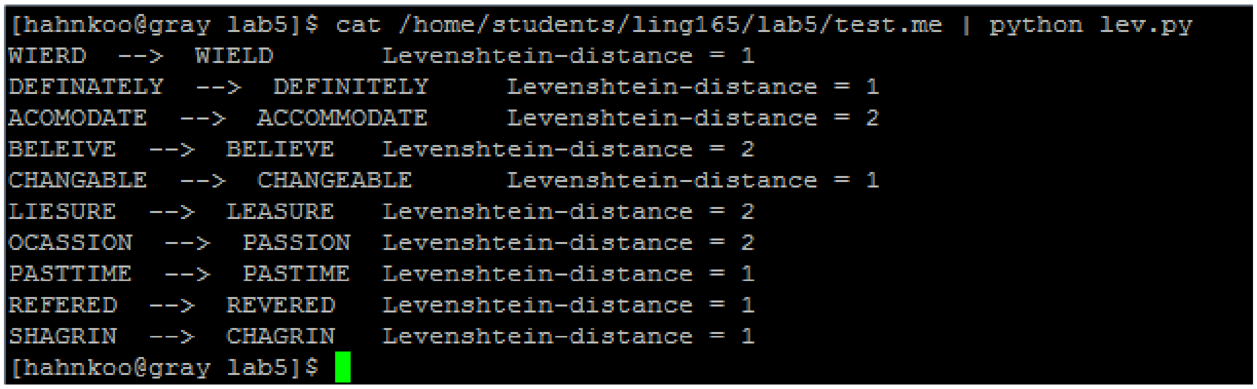
Write a spelling correction program that does the following:

- Take lines of words (one word per line) as input.
- For each word, do the following:
    - Look up the Brown Dictionary to see if it's there.
    - If it is there, print out the input word.
    - If not, print out the most similar word in the dictionary.

For Brown Dictionary, use `/home/students/ling165/lab5/brown.words` . Keep in mind that words in the dictionary are all in upper-case.

In identifying the most similar word in the dictionary, use the Levenshtein distance (see section 2).

For benchmark results, I ran my program on `/home/students/ling165/lab5/test.me` and got the following:



```
[hahnkoo@gray lab5]$ cat /home/students/ling165/lab5/test.me | python lev.py
WIERD  -->  WIELD       Levenshtein-distance = 1
DEFINATELY  -->  DEFINITELY     Levenshtein-distance = 1
ACOMODATE  -->  ACCOMMODATE     Levenshtein-distance = 2
BELEIVE  -->  BELIEVE   Levenshtein-distance = 2
CHANGABLE  -->  CHANGEABLE      Levenshtein-distance = 1
LIESURE  -->  LEASURE   Levenshtein-distance = 2
OCASSION  -->  PASSION  Levenshtein-distance = 2
PASTTIME  -->  PASTIME  Levenshtein-distance = 1
REFERED  -->  REVERED   Levenshtein-distance = 1
SHAGRIN  -->  CHAGRIN   Levenshtein-distance = 1
[hahnkoo@gray lab5]$
```

My program prints out how each word in `test.me` should be corrected along with the Levenshtein distance between the word in `test.me` and the suggested alternative. It's possible that your program may suggest a different alternative from mine if the Levenshtein distance is the same. For example, for REFERED, your program may suggest REFEREE, another word that has the Levenshtein distance of 1 from REFERED. It's just that my program saw REVERED before REFEREE for some reason. So I suggest you make your program not only print out suggested words but also their Levenshtein distances for comparison.

2. Levenshtein distance

The Levenshtein distance, also called minimum edit distance, between two strings x and y is the minimum number of edit operations you need to perform to get y from x. By edit operations, I mean any of the following:

- substitution: replacing a letter in x by another letter (e.g. la**z**er --> la**s**er)
- insertion: inserting a letter after a letter in x (e.g. fe**b**uary --> fe**br**uary)
- deletion: deleting a letter in x (e.g. s**h**ee --> see)

For example, you would need two such operations to get britney from brintey: substitute n in brintey with t and substitute t in brintey with n. So the Levenshtein distance between the two would be 2.

Calculating the Levenshtein distance between two strings is no different from what you did in class for string alignment using dynamic programming: (1) assume the distance between two letters is 1 if they are different and 0 if they are the same, and (2) align the given pair of strings as you add up the letter distances recursively. The value in the bottom right cell of the distance matrix is the Levenshtein distance.

How are the two methods the same? Inserting a letter after a letter in x is the same as aligning the inserted letter in y with a null symbol in x (e.g. feb**_**uary vs. feb**r**uary). Deleting a letter in x is the same as aligning the deleted letter in x with a null symbol in y (e.g. s**h**ee vs. s**_**ee). With that, you just count how many pairs of letters are different between the two strings. Number of pairs with the null symbol _ on either side should equal the number of insertions and deletions combined. Number of pairs with different letters should equal the number of substitutions.

See 3.10 – 3.11 of the Jurafsky and Martin textbook if you would like more information.