Assignment 2

Spell Checker

Introduction

Please design a spelling checker to examine whether an input word is correctly spelled or not. If the word is misspelled, please list its nearest words which have the computed editing distances smaller than or equal to 3 from the given dictionary. (Remember to list the words in the ascending order of editing distance) Your dictionary file is here (dictionary.txt). The materials you need to upload is the same as those in the last programming assignment.

Process

First, the dictionary file was read in and the words in the dictionary were collected. Next the word is accepted and the dictionary is searched to see if it is in it. A set data structure of the words was created for faster searching. If the word is found in this set of words then a message saying it is correctly spelt is printed.

If the word is incorrectly spelt, the Edit Distance (or Levenstein Distance) is calculated for each word in the dictionary. The Edit distance was calculated based on the algorithm below

- The cost $D(0,0)$ of the node $(0,0)$ is zero.
- Each node (i,j) can be reached only through three allowable predecessors $\{(i-1,j),(i-1,j-1),(i,j-1)\}$.
  - **Diagonal transitions**:

$$d(i,j|i-1,j-1) = \begin{cases} 0 & \text{if } \mathbf{r}(i) = \mathbf{t}(j) \\ 1 & \text{if } \mathbf{r}(i) \neq \mathbf{t}(j) \end{cases}$$

  - **Horizontal and vertical transitions**:

$$d(i,j|i-1,j) = d(i,j|i,j-1) = 1$$

- Algorithm

```
D(0,0)=0
for i = 1 to I
    D(i,0)=D(i-1,0)+1
for j = 1 to J
    D(0,j)=D(0,j-1)+1
for i = 1 to I
    for j = 1 to J
        c1 = D(i-1,j-1)+d(i,j|i-1,j-1)
        c2 = D(i-1,j)+1
        c3 = D(i,j-1)+1
        D(i,j) = min(c1, c2, c3)
D(A,B) = D(I,J)
```

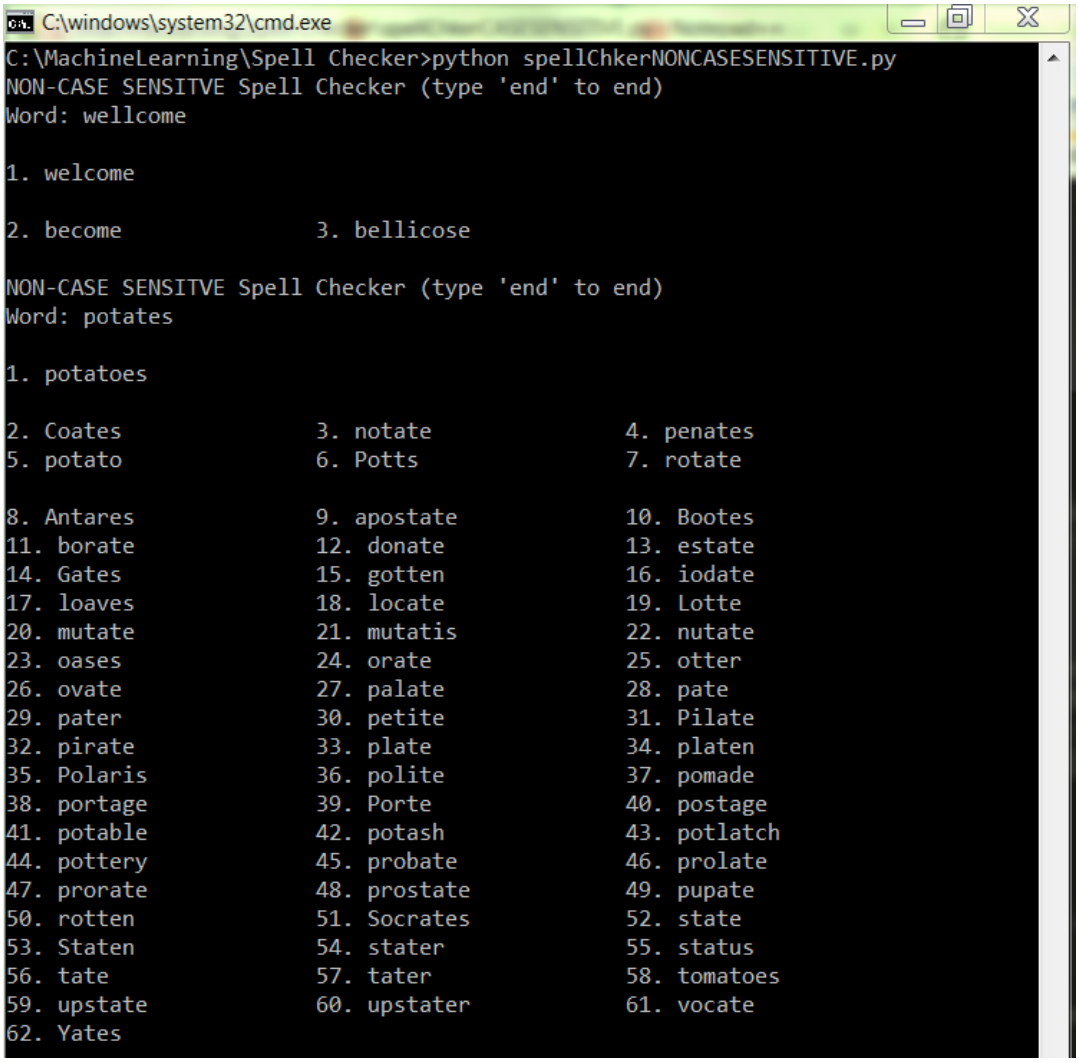
To help speed up this process, the editing distances of words that are too long or too short are passed over. This is because if a word is too long or too short, its editing distance can't possibly be less than the set editing distance limit. For example, given an input string "wosked" and an editing distance limit of 3, then it is clear that the string "woskedaaaa" and "wo" both would have an editing distance of 4 or higher due to insertions or deletions, so it does not make any sense finding the editing distances for these strings. They will not be included as suggestions anyway.

After the editing distances are calculated for words they are placed in a list based on their editing distance value. Words with editing distances of 1 are placed in the first list, words with editing distances of 2 are placed in the 2nd list, etc. This was done for easier printing after.

Finally the suggested words are printed. The words in the 1st list are printed first then those in the second list are printed and so on until all suggestions are printed.

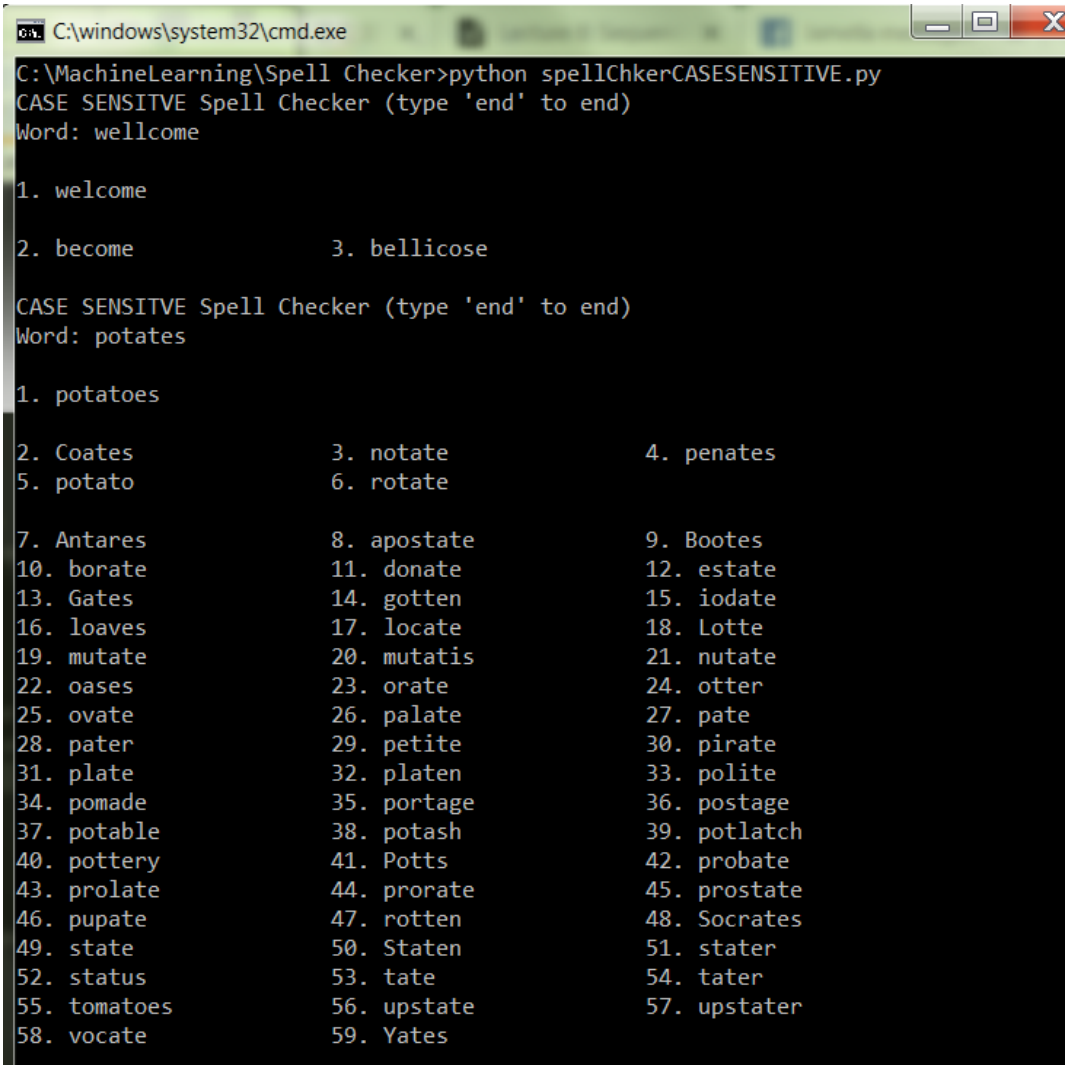
Results

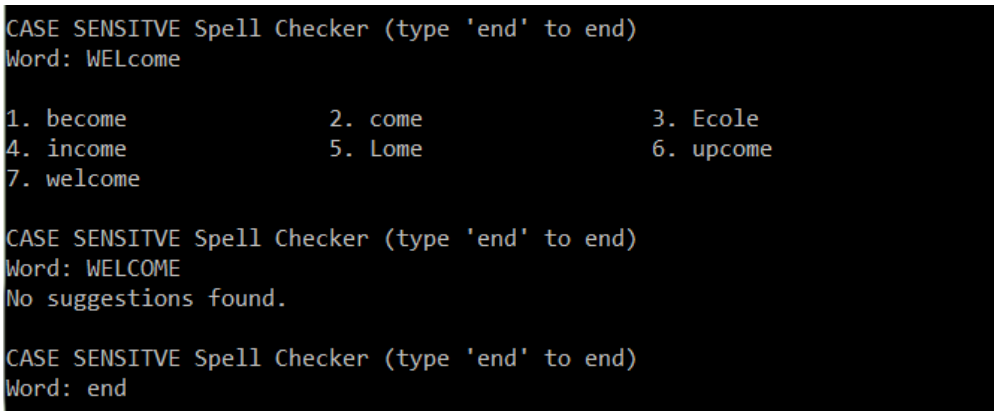
Non-Case Sensitive Spell Checker results




```
NON-CASE SENSITVE Spell Checker (type 'end' to end)
Word: WELcome
Word is correctly spelt


NON-CASE SENSITVE Spell Checker (type 'end' to end)
Word: WELLCOME

1. welcome

2. become               3. bellicose

NON-CASE SENSITVE Spell Checker (type 'end' to end)
Word: end
```

Case Sensitive Spell Checker results





Discussion

The only issue with this assignment is the runtime of the program when finding suggestions. I believe 1 way to lessen the runtime would be to initially sort the words by length, instead of checking for length each time. That way, I won't have to run through the entire dictionary each time looking for words of a certain length first.

Summary

The assignment was fairly simple. The concept of Levenstein Distance was also easily understood. This project was a fun little assignment to put that knowledge into practice.