English spelling checker

Problem definition

In this homework, you need to implement an English spelling checker (and corrector). In general, there are two types of spelling errors:

- Non-word errors (from Birkbeck Spelling Error Corpus)
  - I felt very strang → I felt very strange
  - in the weanter when it was snowing → in the winter when it was snowing
- Real-word errors
  - at brake time → at break time
  - when the brack was finished → when the break was finished

For simplicity, you only need to deal with non-word errors without context information in this homework. More specifically, given an English word (which might be mis-spelled), you need to check if it is listed in a dictionary. If no, you need to give suggestions as what are the most likely words that can be used to replace the original mis-spelled word.

For this homework, use the CMU pronuncing dictionary. Please read its introduction in the previous link, and download the lastest version of CMUDict-0.7b to check out its format. (Remember that we are only using the dictionary to check the spelling of a word. The pronunciation part is not used at all.)

For the suggestions to a mis-spelled word, you can first create a confusable set and then delete those not in the dictionary. For a given mis-spelled word of length nn, the confusable set can be generated by the following operations (see Peter Norvig's spelling corrector):

a. Insert: 26(n+1)26(n+1)
   For instance: face --> {Xface|X=a~z} ∪ ∪ {fXace|X=a~z} ∪ ∪ {faXce|X=a~z} ∪ ∪ {facXe|X=a~z} ∪ ∪ {faceX|X=a~z}
b. Delete: nn
   For instance: face --> {ace, fce, fae, fac}
c. Substitute: 26n26n
   For instance: face --> {Xace|X=a~z} ∪ ∪ {fXce|X=a~z} ∪ ∪ {faXe|X=a~z} ∪ ∪ {facX|X=a~z}

d. Transpose: $n-1$
   For instance: face --> {afce, fcae, faec}

This leads to a total of $54n+25$, with some duplicates. We can denote the above confusable set as $ED_1(string)$, which is set of words that have edit distance of 1 to the given stringstring. For instance, the number of elements $ED_1('something')$ is 494 (with duplicate removed). Similarly, we can have $ED_2(string)=ED_1(ED_1(string))$, which is a set of words that have edit distance of 2 to the given stringstring. This set is easy to write, but takes much longer to compute. For instance, the size of $ED_2('something')$ is 114,324 (with duplicates removed). Then we need to remove words not in the dictionary. If this step is denoted as cleanclean, then our approach is like this:

a. $x_1=ED_1('something')$ ==> $size(x_1)=494$
b. $x_2=ED_1(x_1)$ ==> $size(x_2)=114,324$
c. $out=clean(x_1 \cup x_2)$ ==> $size(out)=5$, that is, out='seething','smoothing','something','somethings','soothing'

(Note that the edit distance used here has unconventional operators such as transposition. Other operators can be defined if necessary.)

The dictionary file is on linux1~linux13: /tmp2/dsa2016_hw5/cmudict-0.7b

An typical example of input file is as follows:

```
severly xxx...
symetrical      xxx...
affets  xxx...
pinoneered      xxx...
xxyyzz  xxx...
happy   xxx...
$severely       xxx...
ever-day        xxx...
ma'amam
.
.
.
```

That is, each row starts with a correct or mis-spelled English word, possibly with some other irrelevant information (as denoted by "xxx..." in the previous example) separated by a tab or a space. In other words, you should read the word at the beginning of each line till the first space or tab, or till the end of line, whichever comes first. Please check out the input files of the following test cases to have a better understanding of the format.

The output corresponding to the previous example input is shown next:

```
severly ==> beverley beverly caverly cheverly cleverly ...
symetrical ==> asymmetrical metrical symmetrical
affets ==> affect affects assets buffets effects ...
pinoneered ==> pioneered
xxyyzz ==> NONE
happy ==> OK
$severely ==> severely
ever-day ==> eveready everyday
ma'amam ==> ma'am macadam manama
.
.
.
```

More about the format:

a. The first word is the original one (either correct or mis-spelled).
b. Everything after "==>" is the list of suggested words in the dictionary, in ascending alphabetic order, separated by a space.
c. If the given word is correct (already in the dictionary), then the suggested word is "OK" alone, such as "happy ==> OK".
d. If the given word is mis-spelled and we cannot find any suggested words, then the suggested word is "NONE" alone, such as "xxyyzz ==> NONE".
e. If the given word has uppercase letters or non-alphabets (such as digits or dashes), just take it as it is to perform the operations.

Requirements & suggestions

- After reading the CMU dictionary, you need to transform all the words into lowercase internally.

- To support efficient access, you can store the dictionary in a hash table. You can also use any other data structures as long as they are as efficient as hash tables.
- TA has tested an STL-based approach to have a running time of 2 minutes for n=600 words. To make it more tolerant for different word lengths, we shall have a time limit of 5 minutes for n=1000.
- The grading criteria:
    a. If the running time is within the time limit, your score is proportional to the number of correct output entries.
    b. If the running time is above the time limit, your score is 50% of the above.
- You can safely assume that all the input words are well hehaved without exceptions. (No big-5 spaces, etc.) To avoid further confusion, we shall assume all the input words are composed of single-byte printable ASCII characters.
- As usual...
    a. Your program should take the input from the standard input and send the output to the standard output.
    b. You need to write the program from scratch. You cannot use any open-source or readily available solvers.