Assignment 1

Introduction

In this assignment you will write an automatic spelling corrector. It will be used to check and correct spelling errors in pieces of text read from text files. The typical use case scenario of your finished program will look like this:

1) A user provides the name of a file that contains text with possible spelling errors.
2) The software opens and processes the file:
    a. The whole text input is split into words
    b. Each word is checked against a dictionary of known words in a pre-defined language
    c. If a word exists in the dictionary (so the word is spelled correctly), it is copied without changes to an output file
    d. If a word doesn't exist in the dictionary, it is replaced by the most likely substitute word (a word with corrected spelling) and written to the output file.

For example, imagine that we have a spelling dictionary containing the following words:

Word	Frequency
black	200
car	100
cat	150
the	500
waz	1

Notice that each word has a frequency count associated with it. In our exercise a dictionary is constructed from a big body of text containing a few million words. The number of occurrences of each distinct word in this text is counted and this count is put in the dictionary next to a word. As a consequence, the more often a particular word appears in the source text, the bigger its frequency count. (If our text is long enough to be representative of a language, the frequency count is a good approximation of the popularity of a word in this language).

We say that the **probability** of encountering words with high frequency counts is bigger than those with low frequency counts. For example, there are two similar words in our dictionary: *car* and *cat*; but since *cat* has a higher frequency count it is more likely to appear then *car*.

There is also a word with a count of only 1 in the dictionary: *waz*. The frequency count of only 1 strongly suggests that it appears there because the dictionary was constructed from a text source which itself contained misspelled words. This happens commonly and to counteract it, one may define a cut-off frequency (e.g. 5), below which the words don't make it into a dictionary.

Now imagine that given our 5-word dictionary we are checking the spelling of a file containing just one sentence:

The cae waz completely blak.

The analysis will proceed as follows:

1) Word *the* exists in the dictionary so it's left unchanged,
2) Word *cae* doesn't exist in the dictionary, but there are two suitable candidates: *car* and *cat* - they both differ only by one letter. Because *cat* is **more probable** (it has a higher frequency count, so it is more common in the language), it's chosen as a replacement,
3) Word *waz* exists in the dictionary (however misspelled) so it's left unchanged,
4) Word *completely* doesn't exist in the dictionary and there is no suitable replacement. What is done with such words depends on the **policy**, the most common one (used e.g. by MS Word) is to leave them unchanged and suggest adding them to a user-dictionary,
5) *blak* doesn't exists in the dictionary, so it is replaced by *black.*

Finally, our input sentence becomes:

The cat waz completely black.

It still contains a spelling error but that's just shortcoming of our dictionary.

Some computer linguistics

To better understand how spelling correction works we need to define a few terms.

Corpus

Your spelling dictionary will be constructed by the program from a big body of text representative to the language. We call such a collation of text a corpus. In this exercise you will work with a corpus composed of English Wikipedia articles and web news. The corpus is available at Leipzig Corpora Collection (http://wortschatz.uni-leipzig.de/en/). The corpus is split into numbered sentences, so when you read it to build your spelling dictionary, you should read it line by line (each line is one sentence) and remove the number prefix. The corpus files are also available on Blackboard.

Spelling dictionary

The spelling dictionary is constructed by your program by processing the text corpus and counting the frequencies of the words occurring in it. A data structure like a `HashMap<String, Integer>` is a good candidate for storing your dictionary. Also think about removing very infrequently appearing words from it. After construction, the dictionary will represent a frequency distribution of the words in language.

Language model of spelling errors

To be able to detect spelling errors and correct them, we first need to define what a spelling error is. A word can be changed by applying a *simple edit* to it. A *simple edit* can be:

a) a *deletion* – a letter is deleted from a word, e.g. *table* becomes *tble* or *tabl.* There are as many possible deletions as letters in the word (5 for table)
b) an *insertion* – a random letter is inserted anywhere in the word, e.g. *table* becomes *ytable* or *tabble*. There are n+1 possible insertion points for an n-letter word. Also, there are as many possibilities for each insertion as letters in an alphabet (26 in English). In total there are 26(n+1) different insertions possible for an n-letter word.
c) a *transposition* – two adjacent letters are swapped, e.g. *table* becomes *atble* or *tabel*. There are n-1 such possibilities for an n-letter word.
d) a *replacement* – a letter is replaced by any random letter, e.g. *table* becomes *tavle* or *tabke*. There are 26n possible replacements for an n-letter word in English.

We say that the *edit distance* between a word and a result of applying a *simple edit* to it is **one**. For an n-letter word there are 54n + 25 words with an *edit distance* of **one** from it.

For the purpose of this exercise we will assume that a word is misspelled if its *edit distance* to a known (correct) word is at maximum two. It means, that we anticipate that a user can make two spelling errors in one word.

For example, the following words are misspelled variants of the word table:

Word	Edits	Edit distance
ttable	Insertion at 0-th letter (t)	1
tale	Deletion of letter 2 (b)	1
tab**n**el	Insertion at letter 3 (n) Transposition between letters 3 & 4 (le $\rightarrow$ el)	2
ta**i**l	Replacement at letter 2 (b $\rightarrow$ i) Deletion at index 4 (e)	2
t**o**able	Insertion at letter 1 (o)	1
toable	Insertion at letter 0 (t) Replacement at letter 0 (t $\rightarrow$ o)	2

Notice that sometimes a misspelled word forms another correct word (tale, tail). In such cases, unless we know the context and have a context-sensitive spelling corrector, we have no other option but to accept such misspelling as correct.

Also notice that sometimes a misspelled word can be interpreted as having 1 or 2 simple edits (*toable*), if that's the case we take the 1-edit version as more probable.

Correcting spelling errors with a dictionary

Imagine that your program has to correct the following sentence:

There arre two koputers on the tabnel.

To fix the errors we will use the reversibility property of *simple edits*. Notice that:

- Each deletion has a corresponding insertion
- Replacements are symmetric (i.e. if the letter *a* is replaced by the letter *z*, we can go back to the original by applying the replacement in reverse)
- Transpositions are also symmetric.

This means that given a word that contains a spelling error, we can recover the original word by applying all the possible *simple edits* to it.

For *arre* we can generate:

- Four deletion: *rre, are, are, arr*
- 26·5 insertions, here the first few: *aarre, aarre, arare, arrae, arrea, barre, abrre, arbre, arrbe, arreb*
- Three transpositions: *rare, arre, arer*
- 26·4 replacements, here the first few: *arre, aare, arae, arra, brre, abre, arbe, arrb*

In total we end up with 54·4+25 = 241 different candidates, and that's just for the *edit distance* of one! To each of those candidates we need to once more apply simple edits to come up with all the possible words that are within the *edit distance* of two from *arre*. This will give 64885 words (many of them duplicated).

Among all the possible substitutes for *arre* there are: *are, barre, rare.* Because *are* is the most common word of those, we'll choose it as the correct spelling for *arre*.

For the word *koputers* among the candidates within the edit distance of two the word *computers* will appear, and it will be chosen as the substitute. The word *tabnel* is already covered in the table above.

Task

1. As already stated, your task is to write a spelling corrector. Specifically your program has to:
   - Create a spelling dictionary from the provided English corpus files by reading them and counting the frequencies of the word appearing in the sentences in those files.
   - Be able to accept any text file as the input for checking the spelling. The file name can be entered by the user during the program execution or provided as the command line parameter (it's your choice which way you choose).
   - Process the input file, using the spelling errors language model and the spelling dictionary constructed from the corpus.
   - Write the corrected text to another file (the output file name can be provided by the user or generated by your program, it's up to you)
   - While correcting the spelling of the input file, the program must preserve all the punctuation marks, newline and tab characters, etc.

2. **You also have to create and submit a class diagram of your program. It is advisable that you first create the class diagram and show it before you do any substantial programming.**

When you design your program think about how you want to split the responsibilities between the classes and how you want to connect them. You will likely need separate classes for: user interaction, reading/processing an input file, spelling dictionary and possibly another one for dictionary creation, etc.

Also think about providing as much flexibility as possible. For example if you write a class that reads an input file and perhaps provides some methods for iterating over the words in this file, write an interface that groups those methods and make your class implement it. This way, if you later decide to add support for other input types (e.g. keyboard or a network connection) you won't have to change your program design but just another class that also implements this interface and can be used instead of your original class.

Tips and hints

1. You can create UML diagrams using the https://www.draw.io/ website or StarUML available at http://staruml.io/ . Submit your class diagram as a PDF document or an image file.
2. When enumerating all the possible word edits for a misspelled word, start with the edits with distance one and store them all in a list. Then check whether any of the candidate words exists in your spelling dictionary. Only if you don't have a suitable candidate use the just generated words to enumerate all the possible candidates with the edit distance of two. Do not attempt to store all those words anywhere: for a 7-letter word there are 173k words with the edit distance of two from it. It's better to check them against your spelling dictionary directly on the fly, as you generate them, and only store the words that appear in the dictionary
3. If you use regex during the creation of the spelling dictionary don't create a new `Matcher` object for each line/sentence you process. It's better to reset the `Matcher` with a new input by using its `reset` method – this way you'll avoid unnecessary object allocation and destruction.

If you are bored

A few things you can add if you've finished the main part:

- Think about caching your spelling dictionary. It's time consuming to create it. Once created, it might be a good idea to save it to a text file that you can load later.
- Add support for another language (e.g. your mother tongue) – you'll have to download the corpus files yourself from http://wortschatz.uni-leipzig.de/en/
- Work on accepting command line parameters and adding options like language selection etc.
- If you support more than one language, add automatic language detection – it's actually pretty easy, just check in which dictionary the most frequent words of the input text appear.
- Handle unknown words in some creative ways. If you encounter a word in the input text that cannot be matched with a word in your spelling dictionary ask the user to either provide the correct spelling or add the word to a user dictionary.

This assignment is inspired by Peter Norvig's excellent article: *How to Write A Spelling Corrector* (https://norvig.com/spell-correct.html).

Peter Norvig is one of the most influential computer scientists and a director of research at Google.