CS2230: Assignment 8

Project: Build a Spellchecker

Assigned November 15, 2016, Due December 9, 2016

Problem 1: 20+30=50 points

Problem 2: 30 points (extra credit)

There are two problems in this project. Problem 1 is a regular assignment (50 points), and Problem 2 is for **extra credit** (30 points). Problem 2 should be taken up only after Problem 1 has been solved.

Problem 1. Build a basic spellchecker that will check for spelling errors in a given file supplied by the user, and output a file with all the misspelled words highlighted by special characters.

Part 1. Build a dictionary

The spellchecker will check spellings by comparing the words from the given input file with the words in a dictionary. We will supply you with a few English novels to build the dictionary. A word is a sequence of at least two letters followed by a non-letter. Do not distinguish between capital and small letters, so 'dog' and 'Dog' are the same word. Your `BuildDict` program will insert distinct words extracted from the given documents into a file `myDict.dat`. In addition to the words in `myDict.dat` also add valid single letter words such as A and I to the data structure.

The program `BuildDict` should start by asking the user for a list of text files to read from. The user should respond by typing the filenames as

```
[filename1] [space] [filename2] [space] [filename3]...
```

The program `BuildDict` will read all of them, and build the dictionary of all distinct words, and write them into the dictionary called `myDict.dat`. You can start with the following three novels to build your dictionary using the attached text files:

1. Leo Tolstoy's "War and Peace": `war.txt`
2. Jonathan Swift's "Gulliver's Travels": `gulliver.txt`
3. R.L. Stevenson's "Treasure Island": `treasure.txt`

Once the dictionary is built, you should take a printout of 40 words from it in alphabetic order in a nice readable format.

Part 2. Build the spellchecker

The dictionary `myDict.dat` will be used by your spellchecker program `SpellCheck`. This program should start by reading the list of words from the file myDict.dat and store them in a `WordList` object. Use hashing with separate chaining (see Assignment 7) to implement the `WordList` class that will support at least the following five functions.

```
public boolean search(String myWord)
```

This method returns true if `myWord` is found in the hash table; otherwise it returns `false`.

```
public void insert(String myWord)
```

This method inserts `myWord` into the hash table if it is not already in it. After `myWord` is inserted into the hash table, you should make sure that there are still sufficiently many empty slots in the table. Specifically, suppose that *N* is the size of the table and *n* is the number of words in the table. After inserting a new word into the table, you should check if the load factor of the table (n/N) is less than or equal to 1.5 $\left(\frac{3}{2}\right)$. If not, it is time to resize (i.e., expand) the table. You can use the usual trick of expanding the table to be twice its current size. However, you will have to rehash all of the words into the table since new table size *N*, determines into which slots the different words are hashed.

```
public int numberWords()
```

This method returns the number of words in the hash table and it should run in constant time.

`public void sortedPrint()`

This method prints out all the words in the hash table in *lexicographic* order in a nice readable format. This will be used for testing only.

`public void sortedPrint(String fileName)`

This method is the same as the above method except that it writes into a file with the given fileName.

After `SpellCheck` has read the list of words from `myDict.dat` it should prompt the user for the name of a document she wants spell-checked. Let the user respond with the file `myEssay`. Then `SpellCheck` should read words from `myEssay` one-by-one, and check which words appear in its dictionary, and will produce a file `myEssay.out` that flags all words with wrong spellings by surrounding them with special character (=). Here is an example.

Input text in `myEssay`:	Lisa eats an aple everry day
`myEssay.out` will display:	Lisa eats an =aple= =everry= day

Documentation. You should document your code extensively. Your comments should be clear and precise but will not add unnecessary clutter to your code. Avoid the approach of ritualistically filling your code with comments, especially those that are pointless (e.g. "this is a variable declaration") or imprecise (e.g. "this loop goes around and around until it figures out the answer").

Extra Credit Problem

Problem 2. Generate spelling suggestions for misspelled words

This part expands on Part 2 of Problem 1 where for each misspelled word, your spellchecker will generate up to five suggestions about the correct spelling.

Here is an example:

Input text in `myEssay`:	Lisa sent a taxt
`myEssay.out` will display:	Lisa sent a =taxt=
List of suggestions:	[0] taxi [1] text [2] tact [3] tart [4] No change [5] Enter your own choice

To accept a suggestion, type in the corresponding index. In case you pick the last option, you have to type in the replacement word. Your spellchecker will make that change in `myEssay.out`, and will continue with the remainder of the text, until you are done. You can call it Version 2 of your spellchecker. You can generate at most five spelling suggestions for misspelled words by assuming that the most likely errors a user makes are incorrectly typing a single letter.

Suppose we have encountered a misspelled word ***w***. To find meaningful replacements for ***w*** we find all the correctly spelled words (in the dictionary) that can be generated from ***w*** by substituting a single letter in the misspelled word ***w***. So your program needs to look for possible replacements by substituting each letter of ***w*** by all possible letters, and checking if the generated words are present in your dictionary. This is somewhat simplistic, but a good first step. In the examples mentioned above, the correct word is perhaps text or taxi, and your program should suggest both, assuming that these are in your dictionary. However, if the correct word is tax (and 't' is an extra letter that appeared), then your program will not be able to figure that out. To find the all replacement words for ***w***, you should implement a search function with the following signature:

```
String[] replacementWord(String w)
```

This function will return all replacement words in the dictionary for ***w***, but you can stop after five words have been found (or sooner if there are fewer than five replacement words for ***w***). In addition, always allow two more choices, one for "`no change`," and another for "`enter your own choice.`"

Documentation. You should document your code extensively. Your comments should be clear and precise but will not add unnecessary clutter to your code. Avoid the approach of ritualistically filling your code with comments, especially those that are pointless (e.g. "this is a variable declaration") or imprecise (e.g. "this loop goes around and around until it figures out the answer").