Universitetet i Oslo
Institutt for Informatikk

I. Yu, D. Karabeg

INF2220: algorithms and data structures Mandatory assignment 1

Issued: 27. 08. 2015

Due: 17. 09. 2015

General description

This assignment is to implement a simple dictionary with basic spell-check functionality for a subset of the English language. The words for your dictionary are provided to you in a text file, named `dictionary.txt` on the inf2220 web-page. The first part of the assignment is to read the file and insert the words into your dictionary data structure.

There are 75.000 words in this file and they come in random order (i.e., they are not sorted in advance, as you will see). The words are separated by newlines and blank spaces, and all the words are given in lowercase letters.

When the program starts it should read the input file and insert all the words into the dictionary. Once all the words have been insert in the dictionary, the word 'busybody' should be removed and then re-added (see details below). The program should provide the user with a simple user interface for looking up words in the dictionary (see details below). If the user tries to lookup a word that doesn't exist, your program should perform a spell-check by looking up similar words (see details below). When the user exits the program, it should output some statistics about the dictionary before quitting (see details below).

1 Data structure

You will have to implement the dictionary as a *binary search tree* containing all the words, including functionality to *insert*, *search* for and *delete* words from the dictionary. Given that that the words are stored randomly in our input file, no steps have to be taken to balance the tree. As one consequence of that: the first word read remains the root of the tree unless it is removed).

Remove method

To remove and reinsert the word 'busybody' you have to implement a functioning removal scheme, which does not destroy the properties that a binary search tree should have (it should still be searchable).

NOTE: Lazy removal is not an accepted removal scheme in this assignment.

2 User interface

We want to give the user a very simple interface. An endless loop that prompts for a word and tries to lookup that word will be sufficient. To abort the loop the letter 'q' may be used (the letter 'q' will not be a word in the dictionary).

Our definition of "spell-check"

When the user have written a word as input the program should search for it in the binary search tree (remember to convert the input into lower case letters).

If the word is found the program shall simply output that the word has been found and prompt for a new word.

If the word is not found the program shall look for similar words in the dictionary in case the user misspelled the word.

Similar words

In our definition of similar words there are four kinds of words similar to the user input:

- A word identical to the input, except that two letters next to each other have been switched. (Example search term: ehy; generated words: hey and eyh; where hey is an example of an English word that is generated)
- A word identical to the input, except one letter has been replaced with another. (Example search term: hby; generated words: aby, bby, ..., hbz; where hey is an example of an English word that is generated)
- A word identical to the input, except one letter has been removed. (Example search term: hy; generated words: ahy, bhy, ..., hyx; where hey is an example of an English word that is generated)
- A word identical to the input, except one letter has been added in front, at the end, or somewhere in the middle of the word. (Example search term: heiy; generated words: eiy, hiy, hey and hei; where hey is an English word that is generated)

If the user input is not found the program should print that the word is not found and then generate all similar words using the rules given above and search for them in the dictionary. If any of the similar words generated is found in the dictionary, print them out as suggestions to the user.

For those cases where the original word was not located in the dictionary, some statistics should be printed:

- The number of lookups that gave a positive answer.
- Time used to generate and look for similar words.

3 Tree statistics

Upon exiting the program it should output the following statistics about the binary search tree:

- The depth of the tree (length of the path to the node furthest away from the root)
- How many nodes are there for each depth of the tree.
- The average depth of all the nodes.
- The alphabetically first and last word of the dictionary.

4 Hints

Conversion between character arrays and strings is very simple in Java, to convert a string to a character array:

```
  char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
```

and back again by initializing a new String object:

```
String str_alphabet = new String(alphabet);
```

This will come in handy during generation of similar words.

Below is a function to generate all similar words according to the first of the four rules. You need to implement the last three.

```
public String[] similarOne(String word){

    char[] word_array = word.toCharArray();
    char[] tmp;

    String[] words = new String[word_array.length-1];

    for(int i = 0; i < word_array.length - 1; i++){
        tmp = word_array.clone();
        words[i] = swap(i, i+1, tmp);
    }

    return words;
}

public String swap(int a, int b, char[] word){
    char tmp = word[a];
    word[a] = word[b];
    word[b] = tmp;

    return new String(word);
}
```

Notice the use of the clone() function here, since arrays are passed by reference the original array would be destroyed during the swap function, and we get a fresh copy of the original word by calling clone() on the original. Remember to try to predict the length of similar words (if you swap the length will be the same, and if you add a letter it will be one longer etc.) These methods may also use other data structures than arrays to store Strings such as LinkedList.

5 How to deliver

The assignment should be carried out individually and delivered through `https://devilry.ifi.uio.no/`.

- The implementation language is JAVA.
- Your implementation must compile on the LINUX machines at the University.
- The delivery must only be in one single archive file: either a `.tgz` or `.zip` archive (with one of those two file extensions).
- Your archive should contain
  - Compilable (and afterwards runnable) source file(s) of your implementation.
  - Print-out of execution in a file called utskrift.txt which shows:
    - The different statistics
    - Spell-check of these words:
      - achiev
      - achiese
      - achievee
      - ahcieve
  - A `README`-file which contains:
    1. How to compile your program (ie. javac *.java)
    2. Which file includes the main-method

3. Any assumptions you have made when implementing the assignment
4. Any peculiarities about your implementation
5. The status of your delivery (what works and what does not)
6. Give credit if your code is heavily influenced by some source (ie. teaching material)

How to create an archive file

.tar

You should create a tar'ed or zipped archive where the top-level directory is your username.

If your username is 'myusername' and you want to tar it:

```
> mkdir myusername
> mv YourProgram.java utskrift.txt README.txt myusername/
> tar -czf myusername.tgz myusername/
```

This will give you a file called myusername.tgz which holds all the content of the assignment in a zipped (compressed) tar-file.

.zip

On machines that cannot create tar-files (eg. Windows) you can move the files you want to compress into a folder. Right click on the folder and click "move to" and then "compressed (zipped) folder".

A .zip-file will now be created containing all files from the folder you right clicked.