Programming Assignment 5

Name: Dewey Kincheloe

Assignment Number: Programming Assignment 5

Assignment Title: Programming Assignment 5

Due Date: November 11, 2016

Programming Assignment 5

Abstract

Problem Description:

This program creates a spell checking “dictionary” then uses it to spell check a book. An array of 26 Binary Search Trees (BST) is created. One for each letter of the alphabet. Thus the first element would contain only those words starting with letter 'a', while the last would contain only those words starting with the letter 'z'. This acts as the directory for all the words from A to Z. The program reads the “random_dictionary” file, converts all the words to lower case then uses these words to populate all the words in the dictionary from a to z.

The Oliver book is then read one word at a time. As each word is read, it is converted to lower case then ran through an algorithm to ensure it is a word containing only the letters a-z. Another algorithm is used to select the first character of this word then compare it to the words in the dictionary based on the words available in the dictionary starting with that letter. If that word is not found, it is either misspelled, or not in the dictionary. It is then counted as a word not found. If the word is found, it is counted as a word found.

The program also counts the number of string comparisons that are done during the search using two counters. One is for the number of comparisons for words that were found in the dictionary, and the other is for the number of comparisons for words that were not found in the dictionary. For example, if the word is “apple” and “apple” is a word in the dictionary, the counter for words found will be incremented. Similarly, if the word “apple” is not found in the dictionary then that counter would be incremented.

At the end, the program computes and displays the average number of string comparisons for words that were found and the words that were not found.

```
/**
 * C202 Assignment 5
 * TestBinarySearchTree.java
 * Purpose: This program creates a spell checking “dictionary” then uses it to
 * spell check a book.
 *
 * @author Dewey Kincheloe
 * @version 1.0 11/11/2016
 */

import java.util.*;
import java.io.*;



public class TestBinarySearchTree {

    //Attributes
        //public int count = 0;
    //Constructors
    //Methods

    /**
    * The removeSpecials method receives a word from the oliver text file and
    * removes all special characters thus ensuring it is only composed of the
    * letters a-z. If it is a word composed entirely of special characters, it is
    * converted to an x so it will still count as a word to be checked.
    *
    */

    public String removeSpecials(String word)
```

```
 {
    String cleanWord = "";
    for (int i=0;i<word.length();i++)
    {
       //Ascci range for a-z 97-122
       if (word.charAt(i)>96&&word.charAt(i)<123)
       {
          cleanWord += word.charAt(i);
       }
    }
    if (cleanWord == "")
    cleanWord = "x";
    return cleanWord;
 }

/**
 * The main method controls program execution.
 *
 */
public static void main(String[] args) {

   float wordsFound = 0;
   float wordsNotFound = 0;
   float compsFound = 0;
   float compsNotFound = 0;
   int[] count = new int[1];
   BinarySearchTree[] list = new BinarySearchTree[26];
   TestBinarySearchTree obj = new TestBinarySearchTree();
   BinarySearchTree<String> dictionary = new BinarySearchTree<>();
```

```
for(int i = 0; i<list.length; i++)
{
   list[i] = new BinarySearchTree<String>();
}

File f = new File ("random_dictionary.txt");
String inputWord;
try{
Scanner in = new Scanner(f);
while ( in.hasNext())
{
  inputWord = in.next();
  inputWord = inputWord.toLowerCase();
  int i = (inputWord.charAt(0)-97);
  list[i].insert(inputWord);
}
System.out. println("Processing  dictionary complete");
in.close();
}
catch(IOException e)
{
  System.out.println("Unable to read file");
}

File book = new File ("oliver.txt");
try{
Scanner in = new Scanner(book);
while ( in.hasNext())
{
  inputWord = in.next();
```

```
    inputWord = inputWord.toLowerCase();
    inputWord = obj.removeSpecials(inputWord);
    int i = (inputWord.charAt(0)-97);
    count[0] = 0;
    if (list[i].search(inputWord, count))
        {
            wordsFound++;
            compsFound += count[0];
        }
    else
        {
            wordsNotFound++;
            compsNotFound += count[0];
        }
  }
  System.out. println("Processing Oliver complete");
  in.close();
}
catch(IOException e)
    {
        System.out.println("Unable to read file");
    }
//System.out.println("count " + count);
System.out.println("Words found = " + wordsFound);
System.out.println("Words not found = " + wordsNotFound);
System.out.println("Comparisons for words found = " + compsFound);
System.out.println("Comparisons for words not found = " + compsNotFound);
double avgcompswordsfound = (double) compsFound / wordsFound;
System.out.println("avgcompswordsfound " + avgcompswordsfound);
double avgcompswordsnotfound = (double) compsNotFound / wordsNotFound;
```

```
    System.out.println("avgcompswordsnotfound " + avgcompswordsnotfound);
  }
}
```

```
run:
Processing  dictionary complete
Processing Oliver complete
Words found = 914054.0
Words not found = 67937.0
Comparisons for words found = 6749036.0
Comparisons for words not found = 528441.0
avgcompswordsfound 7.383629413579504
avgcompswordsnotfound 7.778397633101256
BUILD SUCCESSFUL (total time: 6 seconds)
```