Landon Speer

Assignment Five

Programming Assignment Five

7/21/15

Problem:

In this assignment you will implement the second version of your spell checker. Using the randomized dictionary (*random_dictionary.txt*) given, read in the dictionary into an array of 26 Binary Search Trees (BST) , one for each letter of the alphabet. Thus the first BST would contain only those words starting with letter 'a', while the last would contain only those words starting with letter 'z'. Then, when you read in the book (*oliver.txt*), you examine the first character of each word, and search (this method is already implemented in BinarySearchTee class) the word in one of the BST. For each word read from the book, see if it is in the corresponding BST. If it is not found, then output the word. This word is either mis-spelled, or not in the dictionary. The words in the dictionary and the book should be run through the String Parser you wrote in Assignment 4. It is not necessary to store the book in a BST, only the dictionary should be stored. You are allowed to implement any other methods you see necessary.

Algorithms:

An algorithm for loading the dictionary, which takes in an array of BinarySearchTrees, was created to take in words from the dictionary file and place them into an index of the array based on the value of the first character of the word. There is another algorithm which is the parser that reads each word from the Oliver text file. Based on the first character of the word it will search for the word in the array at the index associated with that first character. If that BinarySearchTree contains the word then wordsFound will be incremented as well as compsFound will be added as a running total. If the word is not found wordsNotFound is incremented and compsNotFound will be added to. At the end of the method it will compute the averages for wordsFound and wordsNotFound by dividing compsFound by wordsFound and compsNotFound by wordsNotFound. There is also a toDisplay method that will display all of the data to the user at the end of the program.

Design:

The program is designed with three different methods. There is a loadDictionary, parser, and a toDisplay method. The loadDictionary loads the array of BinarySearchTrees with words. The parser reads the Oliver file and checks if the words are in the trees or not and will increment the attributes respectively. The toDisplay method will display all the data after it has been computed at the end. In the main method the BinarySearchTrees are instantiated before anything else can be done. Then the loadDictionary method is called, after that the parser is called, then finally the toDisplay is called.

Results/Observations:

The words from the oliver.txt file are searched correctly in the corresponding BinarySearchTree and the counters are correctly incremented. The comparisons give lower numbers than in the linked list implementation of this problem. The averages are also much lower than the linked list version.

Differences:

There are substantial differences between programming assignment four and assignment five. In assignment four with the linked list implementation of the spell checker it takes a lot longer to execute the program. It took just over a minute from start to finish. It also gives larger numbers for the comparison values. However, in the binary search tree implementation the program executes very fast. It only takes about fifteen seconds on average to execute. The comparison values are also much smaller. This happens because the binary search tree is much more efficient. It divides the words into two separate branches whether they are larger or smaller than the root of the tree. This means when it searches for a word it knows already whether to look in the left or right branch of the tree making its search time decrease by half. Since the search required is cut into half this also means that the amount of comparisons are drastically decreased as well. This translates to the averages for comparisons being very low.