Tyler Goodwin

C202

Program 5

November 11, 2016

For our assignment we are to implement the second type of spell checker. We are to use the randomized dictionary and create an array of 26 Binary Search Trees. Each of the Binary Search Trees contains only words starting with ‘a’ then ‘b’ and so on for every words until the last one being ‘z’. After this we are to read the book given (oliver), we are to use the first character of each word and search for the word in the correct Binary Search Tree for the first character. If the word is not found we are to output the word because it is either mis-spelled or not located in the dictionary. We are to implement the string parser used in assignment 4 and run the words through it. We are to only store the dictionary in the Binary Search Tree, not the book. We are to count the number of words found, not found, comparisons of words found, and number of comparisons of words not found, just like we did in assignment 4. We do this by maintaining counters, and at the end of the program we print out the average number of comparisons for words found and not found. Lastly we are allowed to implement any other methods we see necessary.

For this I created three specific methods other than the class, and main method to run. The first of the methods was a dictionary method I created to populate each of the 26 Binary Search Trees with each word in the dictionary starting from the character ‘a’ to the character ‘z’. The second method I created was an oliver method used to read oliver and compare it with dictionary. It counted the amount of words found and not found, then also returned the average comparisons of words found and not found. The third method I created was to overload the search method of assignment 4 and implement a counter for the amount of comparisons and words found.

The results of my program was drastically different from assignment 4. The time of assignment 4 ranged from 30-40 seconds, while the time of assignment 5 took only 1-2 seconds. The reason for this drastic change is because the average amount of comparisons for words found in assignment 5 is only 15, while in assignment 4 it was 3,000 this is a large change. The reason for this large drop in comparisons is because in searching a tree it only has to take into account either a left or right subtree and just ignore the other side, while in a linked list it had to take into account all information in the list until reaching the desired word. The words not found was the same because it still had to search through the whole tree in order to conclude the word was not found.

In my observation of this assignment I did not believe the change in comparisons would be as drastic as it was. I believed it would be around 1,000 but after seeing how the program ran and the efficiency of the program compared to the Linked List I was surprised how simply the code could be change to allow a search tree to be implemented and severely increase the efficiency of the code. I did the math on my own to make sure the amount of comparisons found was correct and found that it was. I thought also perhaps the comparisons of words not found would change also, but after running it I noticed it would not change as it would always have to go through the whole tree and search for that word.

Outputs:

```
run:
Here is the number of words found: 914054.0
Here is the number of words not found: 67937.0
Here is the average number of comparisons for words found: 15.356375
Here is the average number of comparisons for words not found: 7368.0107
BUILD SUCCESSFUL (total time: 1 second)
```