Michael Popplewell

Programming Assignment 5

Spell Check Utilizing Binary Search Trees

17 November 2016

Abstract

For this program the problem primarily deals with extracting the correct material—in this case, words—from a file and comparing them with a dictionary as efficiently as possible. Another issue is making sure that the four primary counters (words found, words not found, average number of comparisons for words found, and average number of words not found) are correctly incrementing at the correct time.

The program will need to have a way to extract the correct material from both the dictionary and the other text file and a way of storing the extracted material—well, not *need* exactly but it does make things a bit easier. To cut the time needed to execute the program, a Binary Search Tree will be dedicated to each letter of the alphabet. Thus, a word that begins with a certain letter will only have to be checked against one BST.

Instantly, I noticed that Programming Assignment 5 ran much more quickly than 4. The numbers—and I'm speaking strictly about the average number of comparisons for words found since that counter in my assignment 4 is a little wonk—definitely reflect this run time. That number was 3493 in assignment 4 while, in 5, it was dramatically reduced to about 8. That is a staggering decrease in the comparisons needed to find a word. And, despite my assignment 4 being messed up, I'm going to assume that the same holds true for the average number of comparisons for words not found.

The reason for this vast increase in efficiency comes down to the structure of the Linked List in assignment 4. For a word to be found in a Linked List, every element up to that particular word needs to be analyzed. So, if the word the code is searching for happens to be the thousandth element, every element before needs to be searched. The entire searching structure changes when dealing with BSTs. When searching a BST it is not necessary to search in a particular order; there are numerous ways to progress through a BST. Within assignment 5's code, it checks to see if the value of the element—in this case a word—is equal to or greater than the element being looked for. If it is less, the elements to the left of the current element is searched. If more, the right is searched. Essentially, the elements needed to be searched in any given BST is halved each time a match is not found, meaning the chance that a match will be found during the next iteration is dramatically increased. It's an exponential cut of time that the Linked List simply cannot provide and is why the results are vastly different.

OUTPUT

```
Words Found: 831143
WordsNotFound: 37450
Average number of comparisons for words found: 7.828711640261477
Average number of comparisons for words not found: 19.244604316546763
```