Eric Schulze
C202 Fall 2016
Programming Assignment 5
11/10/16

Abstract

Program Assignment 5 requires us to compare a text file, that holds the contents of a book, against a dictionary to see how many words in the book are not in the dictionary. We are also required to construct the dictionary based on a provided dictionary text file. This solution into two distinct parts, one for construction of the dictionary and a second for the reading and comparison of the book.

For the construction of the dictionary, an array of 26 BinarySearchTree objects is used, with each BinarySearchTree referring to a letter of the alphabet. A method called populateDictionary() is then called to read in the dictionary.txt file and insert each word into the appropriate tree based on the first letter of the word.

The majority of this solution is dedicated to reading in the words from the book and comparing them against the dictionary. This is accomplished by reading in each string into a holder variable, then examining each character in the string. If the character is a lower case letter, the letter is moved to a second comparison variable. If the character is an upper case letter, it is made lower case and then added to the comparison variable. If the character is a hyphen, it is assumed that the hyphen indicates two separate words, so the comparing method is called on the current comparison variable, the hyphen is ignored, the comparison variable is set to empty, and the loop continues with the second word. If the string has any other kinds of characters, they are ignored. Whenever the isInDictionary method is called, one of the counters, either wordsFound or wordsNotFound, is increased, depending on the return of the isInDictionary method.

The isInDictionary method uses an overloaded search method in the BinarySearchTree. This method takes in a value and an integer array with only one element. The arrays used because it can be passed by reference. In the overloaded method, the number of comparisons that are done are counted and place the value in the array. The isInDictionary method then has access to that count and can update the global variables accordingly.

Output:

```
File Successfully Compared to Dictionary

Total words compared to dictionary: 973648

Total words found in dictionary: 889409
Total words not found in dictionary: 84239

Total comparisons: 15522337
Total comparisons if word was found: 14496470
Total comparisons if word was not found: 1025867

Average number of comparisons if word was found: 16

Average number of comparisons if word was not found: 12

Time required for comparison: 1.324989864 seconds
```

The output showed a total of 973,648 words from the text file were compared to the dictionary. Less than 10% were not in the dictionary. There were only 15 million comparisons made on the BinarySearchTrees, compared to the 3.6 billion comparisons when using the linked lists. The average number of comparisons that were made if the word was found in the dictionary were only 16, compared

Eric Schulze
C202 Fall 2016
Programming Assignment 5
11/10/16
to the nearly 3500 for the linked lists. The average number of comparisons that were made if the word was not found in the dictionary were only 12, compared to the nearly 6500 when using linked lists.

For the average number of words found, if we were dealing with a balanced binary search tree, we could expect O(logN) as an average time complexity. However, since our trees are not balanced, we are looking at a little worse case situation because we might have to travel all they down a really long branch to find a particular word. However, for the words not found, since all we have to do is find a branch where the parent node is on one side of the word lexicographically, and the child node is on the other side. This allows us to maintain a fairly average time complexity, because we have an average of 6500 words on each tree and $\log_2(6500)$ is about 12.

After looking at the time complexities, it is easily seen how our time was dramatically reduced between the use of linked lists and binary search trees. Since we reduced the number of average comparisons in the words found and words not found by 99.54% and 99.81%, respectively, we were able to reduce the time. However, because of the increase in time required to insert each word into the dictionary in our populate dictionary method, we only cut the time by 95.92%.

Output From Linked List Program (for comparison):
```
File Successfully Compared to Dictionary

Total words compared to dictionary: 973648

Total words found in dictionary: 889409
Total words not found in dictionary: 84239

Total comparisons: 3685756655
Total comparisons if word was found: 3138151636
Total comparisons if word was not found: 547605019

Average number of comparisons if word was found: 3528

Average number of comparisons if word was not found: 6500

Time required for comparison: 32.53689657 seconds
```