CS231: Project 6

Word Frequency

Parth Parth | CS231L-A | Dr. Alan Harper

Abstract

In the project, a Generic Binary Search Tree has been created and implemented. A Binary Search Tree is a recursive, node based data structure that stores data according to the following rules: the left subtree has values smaller than the root, the right subtree has values larger than the root, and both subtrees are also binary search trees.

This data structure is specially useful for the use case of this project as it allows us to store and manipulate data in a more efficient way than a list. This data structure has then been used to analyze Reddit comments from years 2008 – 2015. The results have then been tabulated and analyzed.

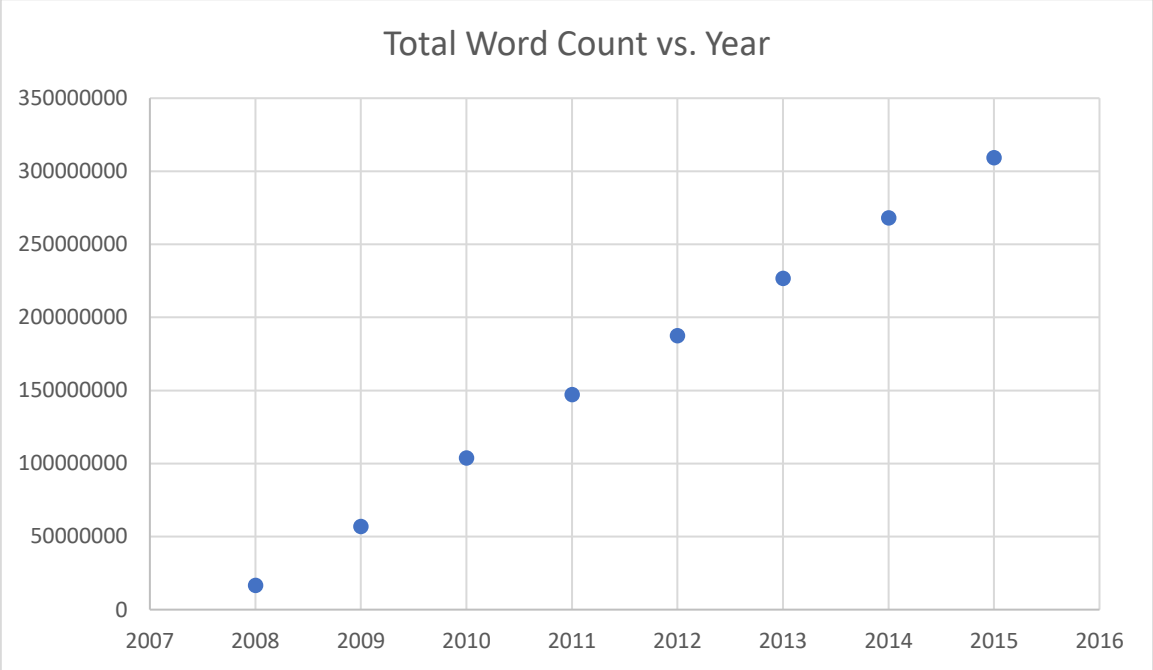
Analysis

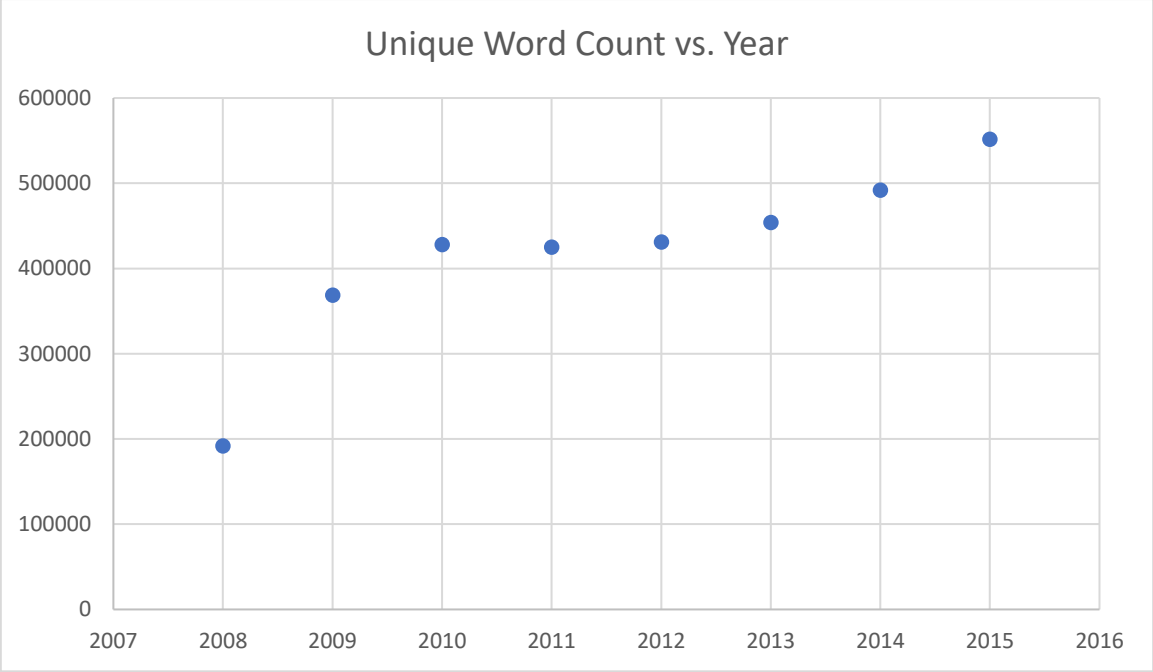
The diff generated from the files obtained from *WCTester.java* is shown in the image below:

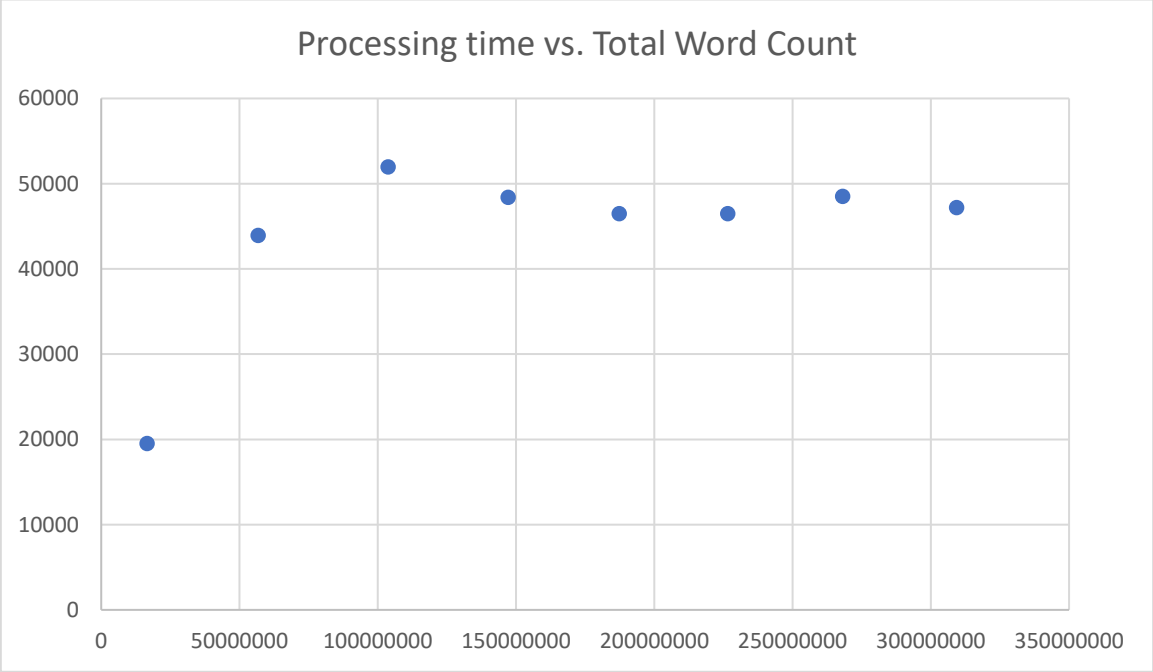
```
C:\Users\Parth> FC "C:\Users\Parth\OneDrive\College Classes\CS231\Projects\Project 6\counts_ct.txt" "C:\Users\Parth\OneD
rive\College Classes\CS231\Projects\Project 6\counts_ct_v2.txt"
Comparing files C:\USERS\PARTH\ONEDRIVE\COLLEGE CLASSES\CS231\PROJECTS\PROJECT 6\counts_ct.txt and C:\USERS\PARTH\ONEDRI
VE\COLLEGE CLASSES\CS231\PROJECTS\PROJECT 6\COUNTS_CT_V2.TXT
FC: no differences encountered
```

So, no differences were found.

The graphs required are as follows:







It is clear from the last graph that the times come closer together as words increase. Since all the words are selected from Reddit, they should be randomly uniformly randomly distributed, giving us a somewhat balanced tree.

The time being recorded is the time taken to read a file (which should be almost constant), add all the words into the tree (which is O(n*log(n))) for the average case). Therefore, it makes sense that the values somewhat look like the graph of n*log(n).

The reason the graph is not exactly like n*log(n) might also be because the number of unique words stay almost constant for 2010, 2011, and 2012 where a dip is observed in the third graph.

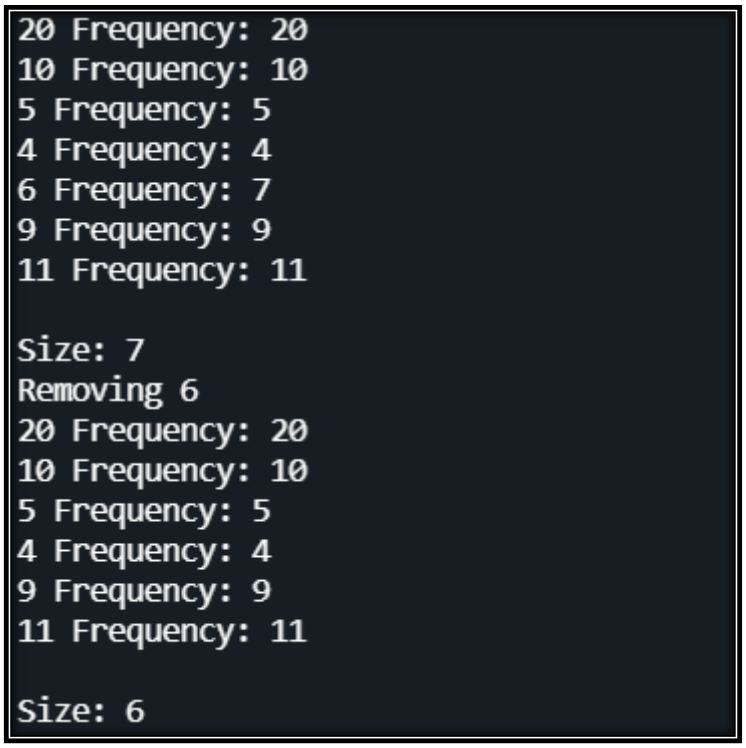
A separate observation is that the number of total words has grown uniformly, in a linear fashion.

Extension 1

In this extension, I have implemented a *remove(K key)* method that takes in a key of type *K* and removes and returns the node where the key was found if it exists, otherwise it returns null. It uses the following logic:

1. If node has no children, just remove the reference to it in the parent.
2. If it has only one child (or subtree), replace the reference to that node with a reference to the subtree root in the parent
3. If node has both children
    a. Find minimum in right subtree
    b. Copy the value to the target node
    c. Remove the minimum in the right subtree

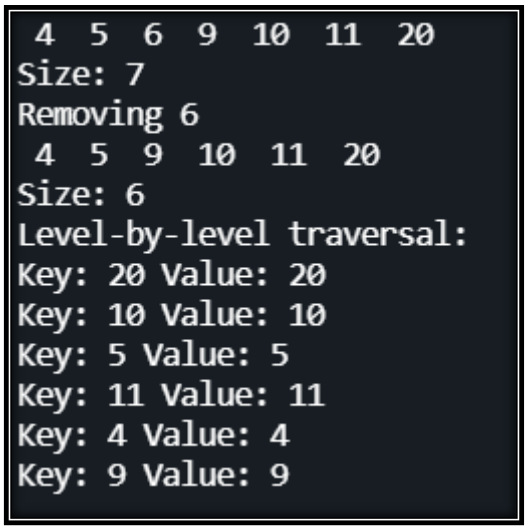
The following image shows a demonstration:



Extension 2

In this extension, I have edited the *toString()* method to return a *String* that contains a breadth-first traversal of the tree i.e. from left to right, level-by-level.

The following image demonstrates this:



The first two lines are the inorder traversal of the BST before and after removing the value 6 from the tree.

References/Acknowledgements

The algorithms for the remove method for extension 1 and the level-by-level traversal that I have employed were covered by Dr. Al Madi in class.