Lab Report

Lab 3

BST / AVL / RBT

Tree Comparisons

EECS 2510

Nonlinear Data Structures

by

David Naelitz

3/30/2017

**Writing the Code:**

When writing the code I experienced less mistakes and stumbling blocks then other projects. This was to be expected as a large portion of the code was provided as well as the binary search tree being already written.

**Execution of the code:**

The code for all the trees ran much faster than I expected. Part of this is having learned about running the release as opposed to the debug version. The debug version has symbols and other overhead. To be able to sort a file 34,948KB (combined files) in roughly 12000 clocks is pretty amazing.

**Results:**

This is where I started to run different graphs to see what kind of information I could gather from the results. My first instinct was to run total words vs. work done. As shown below in Fig.1 I tried one tree before realizing this was not going to give me the information I desired. I determined what would really show me what I wanted was unique words. This is when new nodes would be created, balance factors would change, colors would change, and fixups would run. In Fig.2 you can much clearly see the relation between words and work. BST and RBT were so closely related that they overlap in this graph.

**Fig.1**

**Fig.2**

I then moved on to the next result I wanted to focus on. In class we discussed trees being bounded by their height. That is why we can only run at n lg n time. I decided to look at tree height vs. unique words. As shown in Fig.3 the RBT and AVL keep the tree heights low in relationship to the heights of the BST algorithm. At some points AVL and RBT seem to make the same height trees but overall AVL stayed shorter. This was to be expected AVL is guaranteed to be of height ≤ 1.4404 lg n and RBT is height ≤ 2 lg n.

**Fig.3**

After reaffirming my thoughts about the heights of the tree I decided to look at the height vs runtime but was not creating a very informative graph. As talked about above work was not really shown by the amount of total words but by the amount of unique words. I found the exact opposite when it came to runtime. This is also to be expected because a lot more pointer changes and fixups (work) happens when correcting a tree. Correcting a tree only happens on the insert of new nodes. You could have a very large number of the same words. Runtime, however, is effected by that because you still have to search and insert into the tree as many times as there is a word in a file. This is shown below in **Fig. 4**. Also shown in **Fig.4** I showed unique words vs. runtime for AVL and BST. This is where I noticed that the relationship was not as much related to unique words as it was to total words. What did surprise me was that my BST algorithm was running faster than RBT. One reason this could be is the work to fix the red black tree takes longer then to traverse a larger binary search tree. This could easily change at higher numbers. **Fig.5** is a cleaner version of **Fig.4**.

**Fig.4**

**Fig.5**