Mike Koch
04/04/2015
EECS 2510 | Non-Linear Data Structures

Comparisons between "Vanilla" Binary Search Trees, AVL Trees, and Red-Black Trees When Parsing a Text File

Using a text file containing all of Shakespeare's works, a C++ program was compiled and ran, building three different trees. The first pass was a "dry run", designed to only pass through the file to achieve an "overhead" value, or the time taken to simply read the file. Then, the first "real" pass parsed the file and added each word found to a "vanilla" binary search tree, a tree that had no knowledge of re-balancing itself. As the file was parsed, each word was added to the tree. If the word did not exist in the tree yet, a new Node would be added to the tree with the word as the Node's key. If the word already existed, the Node's "`numberOfOccurrences`" value would be incremented by one.

Once all of the words were added to the tree, metrics information was gathered. Data gathered contained the following:

- Height of the tree
- Number of key comparisons made when searching for the word to be added
- Number of node pointer changes made (when adding a new node, rebalancing, etc.)
- Number of unique words
- Number of words, including duplicates
- (AVL only) Number of balance factor changes
- (Red-Black only) Number of re-colorings
- Elapsed time
- Elapsed time, excluding overhead

These metrics were recorded for all three tree types, as shown in the table below.

Tree Type	Height	# of key comparisons	# of node pointer changes	Unique words	Total Words
Vanilla	39	24,477,152	38,004	38,004	883,299
AVL	18	18,517,302	110,860	38,004	883,299
Red-Black	18	18,156,054	158,110	38,004	883,299

Tree Type	Elapsed Time[1]		# of Balance Factor Changes	# of Re-colorings
Vanilla	11,530	6,494		
AVL	9,677	4,641	45,572	
Red-Black	10,309	5,273		89,471

As seen in the table, both the AVL and Red-Black trees performed better than the vanilla BST; however the AVL tree performed significantly quicker than the Red-Black tree, which was quite surprising, since both trees have some degree of re-balancing themselves. Additionally, while the vanilla tree made a smaller amount of node pointer changes, both AVL and Red-Black had a much smaller tree height, both being at 18. Both AVL and Red-Black also had a lower amount of key comparisons required when searching for a value. As expected, all three trees had the same count of unique words and total words, considering they were reading from the same file all three times.

[1] Units are in clock cycles. The value to the left is the total time including overhead. The value to the right is the total time, not counting for overhead (overhead = 5,036 clock cycles). Times achieved were ran using an Intel® Core™ i5 processor (3.20 GHz)

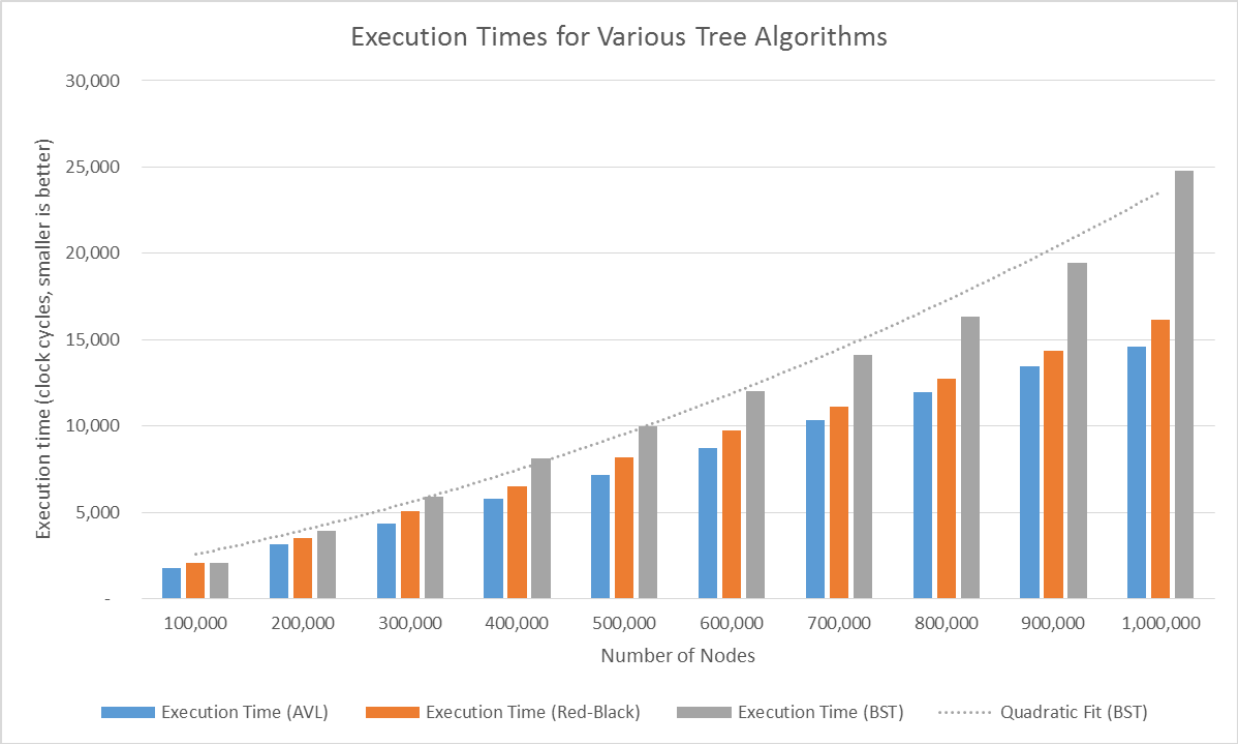
Appendix: Worst-Case Analysis

To further investigate how each of the three trees operate, each tree entered digits from 0 – x, where x was between 100,000 and 1,000,000, in intervals of 100,000. This provided the worst-case analysis in terms of time required to complete the entire construction of the tree. The data gathered is shown in the table below. No overhead was calculated for this analysis.

Tree	1E+5	2E+5	3E+5	4E+5	5E+5	6E+5	7E+5	8E+5	9E+5	10E+5
Vanilla	2,063	3,939	5,905	8,094	9,976	11,990	14,128	16,330	19,469	24,769
AVL	1,777	3,121	4,374	5,759	7,172	8,710	10,356	11,979	13,461	14,622
RB	2,047	3,507	5,048	6,524	8,166	9,763	11,106	12,754	14,385	16,141

Columns indicate number of nodes created in scientific notation. Values are in clock cycles.

As expected, the vanilla implementation longer than the other two tree types as the number of nodes increased. The vanilla tree degenerated into a linked-list due to the order that the data was being received, while the AVL and Red-Black implementations were able to re-balance themselves and keeping a tree-like structure, thus having a lower execution time. Graphing these three plots, along with their proper fits, resulted in:



It is clear that the vanilla tree performed far worse as the number of nodes started to increase, and thus proving that in a worst-case scenario, the vanilla tree becomes an $O(n^2)$ algorithm. Unfortunately Excel has a very limited amount of trend lines available, and thus it is not possible to produce trend lines on-screen for the AVL and Red-Black trees.