Blake Rusteberg

Project #2: RBT and BST

February 21, 2019

CS 340 Algorithms

**Binary Search Tree:** Data for this data structure is stored onto nodes that have right and left pointer properties and a string(key) property that holds the word on that node. When a word is introduced into the algorithm, a new node is created with that data in it and gets properly placed inside the tree. When inserting into the tree, a temporary node is created and initialized. The tree is then traversed until the correct location is found for the temporary node to be inserted at. When the tree is done being built all nodes have a certain word in them, making a dictionary. When searching for a word, the tree gets traversed down checking each node to see if the word has been found or not. Once the word is found the search is complete.

**Red-Black Tree:** Data for this data structure is stored similarly to Binary Search Tree. Nodes are stored right, left and now have a parent pointer property. A string (key) property holds the word on that node and now have a Boolean (red) color property that is used to change the nodes color from red or black. To store information in this structure, the item being inserted is first placed on its own node (z). The tree is then searched through until the proper location is found. Once that item is initially placed, the tree must go through a series of checks and updates such as insertFixUp(), RotateRight(), RotateLeft() to ensure the tree is kept balanced. This stage could slightly change the color or position of one or more nodes in the tree. It then searches similarly to Binary Search Tree.

**Binary Search Tree Time Complexity:** The worst time complexity for inserting into or searching the tree is O(n). This time it would appear if the element being search for or inserted was at the bottom of the tree. The best case is O(log n) with “n” being the height of the tree. This is from the node at each level being searched through until the correct node is found at “n” level. The time complexity for the entire build of the dictionary of words is O(n log n) for “n” number of words being inserted “n” number of times. A disadvantage for a binary search tree is when the tree is being built with a list of sorted words. Even though the list is already sorted, it causes the run time to be greater than if the build was done with a non-sorted list. An advantage to binary search tree is that it is easy to implement, read, write and understand.

**Red-Black Tree Time Complexity:** Inserting into a Red-Black tree guarantees an O(log n) time complexity. The tree’s insert method insures that it stays balanced causing new items to always be added at “n” height. The same rules apply when searching through an RBT. A disadvantage to this structure is the complexity of the insertion method, it is difficult to understand and can take longer to construct and debug. An advantage is its organization and its guaranteed time complexity.

**Fast and Slow queries:** Every word that is found or not found in a permutated binary search took 0ms. This is also true for a Red-Black Tree. When searching in a Sorted list the search times varied slightly. When searching for the word “ANTISOCIAL” in a binary search tree in the file “sorted150K”, the search took .054ms, and searching for it in a red-black tree it took .065. When inputting a word that isn’t found it search times took as long as 1.054ms while in a red-black tree.

**Permutated List:** The unsorted list does not behave as expected. Red-Black trees are expected to run at O(log n) time which is fast than Binary Search Trees O(n) time complexity. The graph shows the at some points that it was faster at building the tree with the Binary Search tree algorithm.

**Sorted List:** The sorted listed behaved as expected. The execution times for both trees were high, and were even throughout until about 90K, then the RBT slowed down substantially. From the results of the graph it shows that when searching in a sorted list it takes longer to search for a word in a red-black tree. This is because the tree must rotate at least once during the insert once it got far enough from the tree.

**Query Test:** For my query test, I tested the word “ANTISOCIAL”. The times of finding the word in a permuted were too small to compute so it rounded them to 0. In a sorted list it took between .05-0.75 milliseconds to find the word. It was unexpected that the sorted lists took longer to find the word.