Avi Montanez

10/13/2016

Lab Section: 6:30-7:45PM

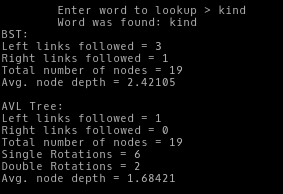
Testfile4.txt :

a b c d e f g h i j k l m n o p q r s t u v w x y z

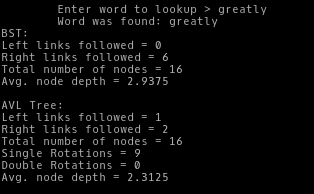
From my example, you can tell that AVL trees are better than Binary Search Trees when it comes to finding information. Due to the manner in which AVL trees are constructed it takes less time to find information within it than BST. One can see this through the results of my test file in relation to the number of links followed. My test file contained each letter of the alphabet in successive order (a b c d e f …). When looking up the word “l”, there were 11 total links followed in the BST and 2 total links followed in the AVL tree. This is due to the fact that in the Binary search tree since the root was “a”, everything inserted into the tree was “greater than” the last insertion so the tree essentially ended up being a linked list. However, the AVL tree continually adjusted itself to ensure that the height of every child subtree differed at most by one which in turn balanced the tree and prevented a linked list situation from occurring. So when I decided so search for “l”, it was only two levels down while it was 11 levels down in the BST.

Results:

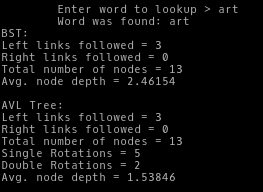
Testfile1.txt



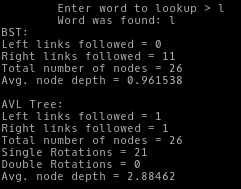
Testfile2.txt



Testfile3.txt



Testfile4.txt



Both AVL trees and binary search trees are useful for their own respective reasons. However, AVL trees are usually preferable to BSTs due to its balanced structure. In situations where an already sorted list is being implemented into a tree, AVLs are much more preferable than BSTs. This is due to the fact that an AVL tree will continually readjust its structure to ensure that for each node, the heights of its child subtrees differ at most by 1 and result in a lower number of links needed to get to what you’re looking for. This also ensure logarithmic lookup time. However, this is not the case for BSTs. Inserting a sorted list into a BST will result in a linked list which ends up being inefficient in terms of time (in this case: linear vs. logarithmic) when searching for items within the tree when compared to AVL trees. Also, AVL trees are just better than BSTs in general when you expect the number of lookups to be greater than the number of updates to the tree and this was exemplified by the more optimal results of the AVL trees for all the test files used in the lab.

Even though AVL trees are usually more optimal than BSTs, there are some costs associated with them. Though lookup in AVL trees are typically faster, it comes at a cost of slower insertion and deletion of nodes due to the multiple rotation operations that need to occur. Another cost of using AVL trees is the pure difficulty of implementation. When designing a program implementing an AVL tree, it can be very difficult and more time consuming to write code accounting for methods such as insertion and deletion due to their involvement with rotation operations. Lastly, another cost of implementing AVL trees can be space. Along with other characteristics of AVL trees that take up memory, one has to also store memory for the balance factor of each node in the tree.