Ashley Nguyen - apn2my
3/2/16
Section 102

testfile4.txt contents:
ashley
betty
carlos
daniel
eric
huzzah

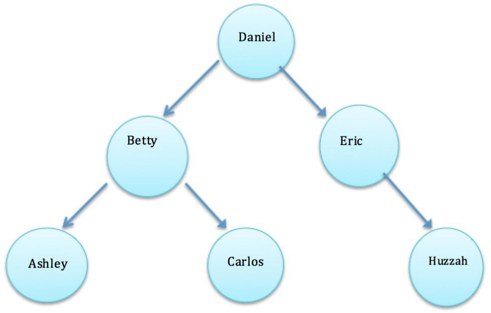

Figure 1

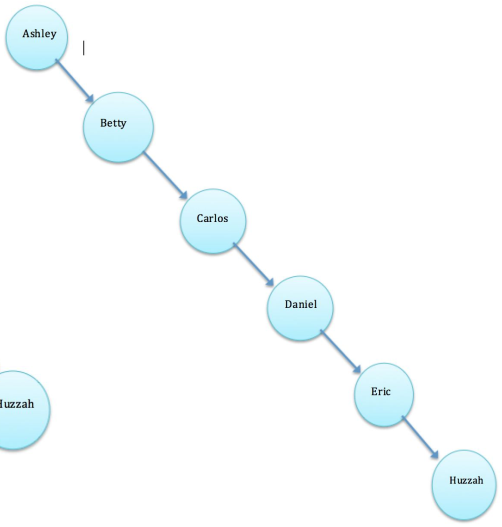

Figure 2

This file works because when searching for "huzzah", the leaf node in the BST, the test code prints that 5 right links have been followed (see figure 2). This makes sense because in a BST the test file will construct the words as a linear right subtree with "ashley" as the root. Furthermore, when searching "huzzah" in the AVL tree, 2 right links have been followed since the tree is balanced, which makes "daniel" the root node, and "huzzah" the leaf in the right subtree that has a height of 2 (see figure 1).

Results

testfile1.txt

Lookup word: **thinking** **BST:** Left Links Followed = 2 Right Links Followed = 3 Total Number of Nodes = 19 Avg. node depth = 3 **AVL:** Left Links Followed = 1 Right Links Followed = 1 Total Number of Nodes = 19 Single Rotations = 6 Double Rotations = 2 Avg. node depth = 2	Lookup word: **kind** **BST:** Left Links Followed = 3 Right Links Followed = 1 Total Number of Nodes = 19 Avg. node depth = 3 **AVL:** Left Links Followed = 1 Right Links Followed = 0 Total Number of Nodes = 19 Single Rotations = 6 Double Rotations = 2 Avg. node depth = 2

testfile2.txt

Lookup word: **bee** **BST:** Left Links Followed = 0 Right Links Followed = 1 Total Number of Nodes = 16 Avg. node depth = 6 **AVL:** Left Links Followed = 2 Right Links Followed = 0 Total Number of Nodes = 16 Single Rotations = 9 Double Rotations = 0 Avg. node depth = 2	Lookup word: **had** **BST:** Left Links Followed = 1 Right Links Followed = 7 Total Number of Nodes = 16 Avg. node depth = 6 **AVL:** Left Links Followed = 1 Right Links Followed = 3 Total Number of Nodes = 16 Single Rotations = 9 Double Rotations = 0 Avg. node depth = 2

testfile3.txt

Lookup word: **misty** **BST:** Left Links Followed = 3 Right Links Followed = 2 Total Number of Nodes = 13 Avg. node depth = 3 **AVL:** Left Links Followed = 2 Right Links Followed = 1 Total Number of Nodes = 13 Single Rotations = 5 Double Rotations = 2 Avg. node depth = 2	Lookup word: **clockwork** **BST:** Left Links Followed = 2 Right Links Followed = 0 Total Number of Nodes =13 Avg. node depth = 3 **AVL:** Left Links Followed = 2 Right Links Followed = 0 Total Number of Nodes = 13 Single Rotations = 5 Double Rotations = 2 Avg. node depth = 2

testfile4.txt (Experiment)

Lookup word: **betty** **BST:** Left Links Followed = 0 Right Links Followed = 1 Total Number of Nodes = 6 Avg. node depth = 2 **AVL:** Left Links Followed = 1 Right Links Followed = 0 Total Number of Nodes = 6 Single Rotations = 3 Double Rotations = 0 Avg. node depth = 1	Lookup word: **eric** **BST:** Left Links Followed = 0 Right Links Followed = 4 Total Number of Nodes = 6 Avg. node depth = 2 **AVL:** Left Links Followed = 0 Right Links Followed = 1 Total Number of Nodes = 6 Single Rotations = 3 Double Rotations = 0 Avg. node depth = 1
Lookup word: **carlos** **BST:** Left Links Followed = 0 Right Links Followed = 2 Total Number of Nodes = 6 Avg. node depth = 2 **AVL:** Left Links Followed = 1 Right Links Followed = 1 Total Number of Nodes = 6 Single Rotations = 3 Double Rotations = 0 Avg. node depth = 1	Lookup word: **daniel** **BST:** Left Links Followed = 0 Right Links Followed = 3 Total Number of Nodes = 6 Avg. node depth = 2 **AVL:** Left Links Followed = 0 Right Links Followed = 0 Total Number of Nodes = 6 Single Rotations = 3 Double Rotations = 0 Avg. node depth = 1

AVL trees may be considered over Binary Search Trees when a faster run-time performance is desired. AVL trees find, insert and remove operations are generally performed at log(n) whereas BSTs generally have linear (n) performance for the same operations. For example, when trying to find an item in an AVL tree, since the tree is balanced such that each node's balance factor is either -1, 0 or 1, it will most likely have a shorter internal path length compared to an unbalanced BST. Thus, finding an item in an AVL tree is not only faster but also efficient in terms of path length. Searching for an item in a Binary Search Tree on the other hand would require a node by node search, determining if the node is less than or greater than the target and then proceeding down the left subtree if less than or down the right if greater than until the target is found. This is the reason why AVL trees are preferable to BSTs for find operations because these data structures are guaranteed to have a log(n) running time.

One of the costs involving AVL implementation involves the data structure's memory consumption. Since each node has to remember and possible update its balance factor, it requires memory to hold this value. The cost may end up being very high for large data structures with many nodes/elements since each node will take up memory to hold the balance factor value. Furthermore, for large AVL trees, the operations for insert and remove may end up being slow, costing time. How this can occur is when inserting many nodes one by one, since the tree may go through multiple changes as each node is inserted. By the time all the nodes have been inserted, the aggregate amount of the number of times a single and/or double rotation has occurred as each node was inserted may affect the average running time. Although this self-sorting capacity maintains the tree's balance, if done multiple times when inserting many nodes, then the overall running time may not be worth the cost of just a fast search.