Amber Liu

Al7bf

10/18/2018

Section 106(3:30-4:45)

Based on the testfile results for AVLtree and BSTtree, both numNodes came out with the same output, but the path that led to the node was considerably different. For example, where BSTtree was “We can't solve problems kind created” while AVLtree resulted with “problems can't kind created”. This demonstrates the difference in insert operation and find operation in the two trees, where AVLtree have a shorter path to the node being found and seems to require less operation runtime.

AVL trees is considered over Binary Search Trees when a faster runtime is desired. AVL trees generally perform at log(n) when doing insert, remove, and find operations while BST have linear performance. When finding an item in AVL, because the tree is balanced with the balance factor already, it will have a shorted path length compared to an unbalanced BST. The memory consumption for AVL implementation however is much more. Since each node has to remember and update its balance factor, it requires memory to hold the value. Cost of space may be very high for large data structures since each node will take up memory. For large AVL trees, the operations for insert and remove will result in slower runtime. One insert could lead to multiple changes and rotations as the new node is inserted. By the time all the nodes have been inserted, it has resulted in multiple times of single and/or double rotations. The self sorting ability of AVL tree might maintain the tree’s balance, but it also results in usage of a lot of memory.

AVL tree would be preferable over Binary search trees where its time complexity is faster than BSTs in worse case scenarios. In addition, when you want a balanced tree, AVL tree is able to self balance it through rotations whereas binary search tree does not.