Maddie Stigler – mgs4ff
10/8/14
analysis.pdf
2150 – 104

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

This test file works because it shows a large difference in performance between BST trees and AVL trees in a simple way. While the AVL tree will be continuously balancing itself to keep the difference in nodes no greater than 1, the BST will continue storing the elements in a linear structure. This results in a greater search time for the BST and a quicker one for the AVL tree. The AVL tree has a smaller node depth, thus limiting the number of links followed to access a particular node.

Results for testfile1:
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Avg. node depth = 3.157789

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
Avg. node depth = 2.26316
Enter word to lookup > einstein
Word was found: einstein

BST:
Left links followed = 4
Right links followed = 2
Total number of nodes = 19
Avg. node depth = 3.15789

AVL Tree:
Left links followed = 2
Right links followed = 2
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
Avg. node depth = 2.26316

Results for testfile2:
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16

Avg. node depth = 6.0625

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2.5
Enter word to lookup > bee
Word was found: bee

BST:
Left links followed = 0
Right links followed = 1
Total number of nodes = 16
Avg. node depth = 6.0625

AVL Tree:
Left links followed = 2
Right links followed = 0
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2.5

Results for testfile3:
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Avg. node depth = 3.23077

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Single Rotations = 1
Double Rotations = 2
Avg. node depth = 2.23077
Enter word to lookup > littered
Word was found: littered

BST:
Left links followed = 1
Right links followed = 1
Total number of nodes = 13
Avg. node depth = 3.23077

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Single Rotations = 1
Double Rotations = 2
Avg. node depth = 2.23077

Results for testfile 4:
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Avg. node depth = 12.5

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3
Enter word to lookup > l
Word was found: l

BST:
Left links followed = 0
Right links followed = 11
Total number of nodes = 26
Avg. node depth = 12.5

AVL Tree:
Left links followed = 1
Right links followed = 1
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3

Based just on the data above, you can see several benefits to choosing AVL trees over Binary Search Trees. One of the main situations where AVL trees are preferable to BSTs is when quick searches are necessary. Searching in AVL trees is O(log n) since AVL trees are always balanced. AVL trees use rotation of nodes to balance themselves and limit the node depth of the tree. This makes searches in AVL trees quicker than BST searches. Another reason we might prefer using AVL to BST is

if you already have a dataset and want to access large amounts of data quickly. While the insertion and deletion operations on an AVL tree may be costly in terms of time, the tree remains balanced whereas a BST might be more likely to be more heavily weighted on either the left or right side.

One of the costs incurred in AVL implementation include a longer insertion and deletion time. This is due to the fact that the tree does self-balance which can require node rotations. These node rotations are accomplished as either single node rotations or double node rotations in order to ensure the node height difference does not differ by more than one. Binary search trees do not self balance so insertions and deletions in BSTs don't have quite the time cost that AVL trees have. However, this cost is incurred in order to make search time more efficient and to make the tree itself more balanced and optimized. Another potential cost of AVL implementation is that while they are more balanced than BSTs, they are not perfectly balance. This just means they could be even more optimized for searching and probably less so for inserting and deleting. After running these test files, I would think that it isn't completely necessary for the tree to be perfectly balanced to be highly optimized.

In general, AVL trees are preferred over Binary Search Trees due to their self-balancing nature. The rotating property of the nodes in the tree and the inclusion of the balance factor in the node fields help to keep the tree balanced and optimized for searches. While there are a few costs incurred during the implementation of AVL trees, they are also necessary to maintain the balancing and optimizing properties of AVL trees.