William Emmanuel
wre9fz
February 27, 2012
Section 104
AVL and BST Comparison

testfile1.txt:

we can't solve problems by using the same kind of thinking we used when we created them -albert Einstein [17 nodes total]

Output for creation and lookup of the element ‘kind’:

	L Links Followed	R Links Followed	Avg. Node Depth	Single rotations	Double rotations
BST	3	1	3.52941	-	-
AVL	1	0	2.52941	2	2

testfile2.txt:

a bee caught dung everywhere flying greatly higher in mauve skies than we had ever flown [16 nodes total]

Output for creation and lookup of the element ‘skies’:

	L Links Followed	R Links Followed	Avg. Node Depth	Single rotations	Double rotations
BST	0	10	6.0625	-	-
AVL	1	2	2.5	9	0

testfile3.txt:

zany cobwebs littered the clockwork orange landscape like misty works of surreal art [13 nodes total]

Output for creation and lookup of the element ‘skies’:

	L Links Followed	R Links Followed	Avg. Node Depth	Single rotations	Double rotations
BST	2	2	3.23077	-	-
AVL	1	2	2.23077	1	2

testfile4.txt:

z y x w v u t s r q p o n m l k j i h g f e d c b a [26 nodes total]

Output for creation and lookup of the element ‘a’:

	L Links Followed	R Links Followed	Avg. Node Depth	Single rotations	Double rotations
BST	25	0	12.5	-	-
AVL	4	0	3	21	0

3 This test file contains the alphabet backwards. This simple test shows both the advantages and drawbacks of AVL and BST trees. Since the entire wordlist is sorted, and each element is smaller than the one before it, elements will always be inserted on the left sub tree in binary search trees. This will in short create a linked list—it will be an extremely unbalanced tree with 26 nodes all found straight down the left sub tree. It will thus preform like a linked list, with linear time for retrievals.

Since the original input is very unbalanced, the AVL tree will have to do many rotations when inserting to bring balance to the tree. This is a costly operation, but pays off for data retrieval. Data retrieval in the worst case for BST can be linear, whereas the data retrieval in the worst case for AVL is logarithmic. So, when searching for the letter 'a' in the BST, 25 steps are needed; when searching for 'a' in the AVL, only a handful are required. Again, this sample case touches on many of the advantages and drawbacks of AVL and BST trees.

5 As seen in testfile4.txt, AVL trees are preferable for several reasons. In short, AVL trees ensure that retrieval times will always be acceptable. Unlike the BST, an AVL can never yield linear retrieval times. It constantly boasts a logarithmic retrieval, no matter how unbalanced the order of node insertion was. So, in cases where nodes will likely be inserted in a very unbalanced fashion, the AVL is preferable.

Besides better search times, the AVL tree features good traversal. Since balance is a key feature of the AVL, traversal of the tree in any order is "neater" than it would be in its BST counterpart.

6 Although AVL trees do boast much better data retrieval times, there are several drawbacks. Due to the balance calculations and shifts, insertion and removal times are much more costly than the BST tree. For about half of all insertions, the tree has to be rewired with either single or double rotations. Granted, this just consists of the reassignment of pointers, a process of constant runtime—however, it is still a cost nonetheless. In addition, the act of implementing an AVL tree is difficult. The BST can be coded up very easily, whereas the AVL tree requires much more code complexity. Finally, the AVL tree takes up more memory, as each node needs a new field for its balance factor.