Justin Choi (jc8mc)
March 3rd 2017
CS 2150 – 106

Lab 5: Trees Post Lab

testfile4.txt: a bee caught dont ever forget great hazardness if jessica knowingly looks making nice oregano plates quickly returning to underwood veals

For testfile4.txt, I chose to make my test file in the order of alphabets since the subsequent alphabet increases in value by each character. For a binary search tree (BST), all of the nodes will be placed in one long path—each node having only one child, except for the leaf node because there is no reordering. Thus, the BST results in a very poor balance. However, an AVL tree sorts my test file in a more efficient and balanced way. This is clearly shown in the average number of nodes for the two different trees. The average node depth is significantly lower for the AVL tree than the average node depth for BST.

Results for all tests:

testfile1.txt (19 words)
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Avg. node depth = 3.15789

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 > ls
ls was not found in testfile1.txt
BST:
Left links followed = 4
Right links followed = 2
Total number of nodes = 19
Avg. node depth = 3.15789

AVL Tree:
Left links followed = 3
Right links followed = 1
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
Avg. node depth = 2.26316

Justin Choi (jc8mc)
March 3rd 2017
CS 2150 – 106

testfile2.txt (16 words)
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 > flying
Word was found: flying
BST:
Left links followed = 0
Right links followed = 5
Total number of nodes = 16
Avg. node depth = 6.0625

AVL Tree:
Left links followed = 1
Right links followed = 1
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2.5

testfile3.txt (13 words)
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 > landscape
Word was found: landscape
BST:
Left links followed = 2
Right links followed = 1
Total number of nodes = 13
Avg. node depth = 3.23077

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

testfile4.txt (21 words)
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 21
Avg. node depth = 10

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 21
Single Rotations = 16
Double Rotations = 0
Avg. node depth = 2.7619

Enter word to lookup > oregano
Word was found: oregano
BST:
Left links followed = 0
Right links followed = 14
Total number of nodes = 21
Avg. node depth = 10

AVL Tree:
Left links followed = 1
Right links followed = 3
Total number of nodes = 21
Single Rotations = 16

Justin Choi (jc8mc)
March 3rd 2017
CS 2150 – 106

Double Rotations = 0
Avg. node depth = 2.7619

Discussion

The AVL tree is clearly more preferable than the BST for balancing. The results of the test files, which show the performance of the two different trees, prove this notion. For example, in my testfile4.txt, which is the unsorted binary search tree, the worst case of a BST becomes the equivalent of the number of nodes as the height of the tree. In contrast, the worst-case scenario for the AVL tree for find is θ(log n) while it is θ(n) for a BST. Secondly, for inserting a node, when the BST is unsorted like in testfile4.txt, it continuously adds each successive node onto one side of a tree, making it unbalanced. However, the AVL tree quickly rotates to create balance. It is a more efficient data structure and it is also sorted with lesser average node depth. Thus, due to these reasons, AVL tree is more preferable since it will be self-balancing and it does not require elements to be sorted first.

Thus, it seems that AVL trees are more preferable. However, when we consider the cost involved for creating the AVL and BST tree, the results become different. As AVL trees are self-balanced (inherently balanced) trees, AVL trees are not so memory efficient. While a simple BST is a tree without any balancing requirements, AVL trees require balancing. Thus, when modifying an AVL tree, each time when a node is inserted or removed, there are single or double rotations, made up of two single rotations, since it needs to achieve the requirements of being a self-balanced tree. So, for operations such as insert and remove, operations can be slower for AVL tree, making BST a better tree in terms of operation cost for insert and remove, in rare cases.

Justin Choi (jc8mc)
March 3rd 2017
CS 2150 – 106

In conclusion, it is better to implement an AVL tree for reading or printing due to its logarithmic time. However, for inserting and removing, the BST or other trees that was not mentioned in this lab becomes a better tree since AVL tree needs to sacrifice time cost due to its extra step of reordering and balancing.