Alex Yang (ahy3nz)

February 25, 2014

Lab 104

Testfile4.txt:

apple beta card delta epsilon fiji gamma hello intel jaunt knife last milk nor orange pear quiz red super tan urn velocity will xylophone yam zygote

3. This is one of the worst-case scenarios for the binary search tree, since the words are in order. Consequently, the binary search tree becomes a list since each node only has one child. However, the AVL tree, since it balances itself, is a more packed tree and its nodes fill up the both children. Since the structure of the trees is different, the depths and iterations to find nodes are also different. Specifically, the AVL tree demonstrates average node depth and requires, on average, fewer link-follows to find a node.

4.

Testfile1.txt results:

Enter word to lookup > thinking

Word was found: thinking

BST:

Left links followed = 2

Right links followed = 3

Total number of nodes = 19

Avg. node depth = 3

AVL Tree:

Left links followed = 1

Right links followed = 1

Total number of nodes = 19

Single Rotations = 2

Double Rotations = 2

Avg. node depth = 2

In Testfile1.txt, we can see fewer links are followed to find a word in the middle of the textfile, and the average node depth of the AVL tree is smaller than that of the BST, indicating the AVL tree is more compact than the BST.

Testfile2.txt results:

Enter word to lookup > a

Word was found: a

BST:

Left links followed = 0

Right links followed = 0

Total number of nodes = 16

Avg. node depth = 6

AVL Tree:

Left links followed = 3

Right links followed = 0

Total number of nodes = 16

Single Rotations = 9

Double Rotations = 0

Avg. node depth = 2

In Testfile2.txt, the BST actually requires fewer links to follow than the AVL tree. This is one case where the BST will be faster than the AVL tree. Since the BST does not balance, the root node is the first node that appears in the textfile. When we searched for the first node “a,” the BST was quicker because it did not have to follow any links. However the average node depth of the AVL tree is a third of the average node depth of the BST, indicating fewer links followed to find nodes, on average.

Testfile3.txt results:

Enter word to lookup > surreal

Word was found: surreal

BST:

Left links followed = 2

Right links followed = 3

Total number of nodes = 13

Avg. node depth = 3

AVL Tree:

Left links followed = 1

Right links followed = 3

Total number of nodes = 13

Single Rotations = 1

Double Rotations = 2

Avg. node depth = 2

In Testfile3.txt, we searched for one of the last words, and we observe that the AVL tree finds the node faster than the BST (follows fewer links). Again, the AVL tree has shorter average node depth due to its balancing.

Testfile4.txt results:

Enter word to lookup > orange

Word was found: orange

BST:

Left links followed = 0

Right links followed = 14

Total number of nodes = 26

Avg. node depth = 12

AVL Tree:

Left links followed = 1

Right links followed = 3

Total number of nodes = 26

Single Rotations = 21

Double Rotations = 0

Avg. node depth = 3

In Testfile4.txt, we see a marked difference between the BST and AVL tree. The text is sorted, so the BST inserts nodes at the right child of the parent node each time, effectively creating a list. On the

other hand, the AVL tree's balancing leads to a more compact tree with much smaller average node depth. When searching for a node, the BST essentially traverses each node in tree (or list) until it encounters the desired node. The AVL tree, since it is balanced, requires fewer links to follow, making it more efficient in this case. However, with 26 nodes, 21 single rotations were required. In constructing this tree, a considerable number of rotations were required, which will become computationally demanding for much larger trees as opposed to the BST that requires no balancing.

5. AVL trees are preferable to BSTs in a few cases – particularly during the BST's worst-case scenario. The worst-case scenario is when the data items are already sorted, so each child will be exclusively smaller or greater than the parent node. Consequently, there will be only one child to each parent. In the worst-case scenario, BSTs become lists (O (n)) and all operations (insert, find, remove) become O(n) because you will not be halving the size upon each comparison due to possessing only one child. AVL trees, due to the balancing at each step, has a faster worst-case scenario running time, O(log n). This ensures the height of AVL trees are O(log n)

The closer a set of data is to becoming a linear BST, the more preferable an AVL tree would be. In other words, if a set of data is already sorted or close to being sorted. When a set of data is sorted, the data is in an increasing or decreasing order. As a result, the construction of the BST leads to a parent and right nodes or a parent and left nodes. Then, comparisons do not halve the size and operations simplify to traversing a list linearly. In these cases, when data is already sorted, the AVL tree would be preferred. The AVL tree's balancing would prevent the construction of a linear list and cause operations to become O(log n). However, in average cases, both trees have O(log n) operations.

6. When constructing an AVL tree, however, balancing occurs at each step. This means every node must also contain a balance factor, increasing the amount of memory each node requires. Furthermore, after every operation, the AVL tree must re-balance itself. The re-balancing process will slow down each operation – re-calculating balance factors will mean comparing heights, and rotating will involve moving pointers to parents and children while re-calculating depths and balance factors. In performing runtime analyses, rotations are constant time because they do not vary with the size of the tree. However, recalculating balance factors involves looking at the heights of the children. Depending on the node implementation, calculating balance factors may be linear time if height is not a field stored in each node. If height is stored in each node, then calculating balance factors are constant time. Regardless, AVL tree operations become slightly slower due to re-balancing (balance factors and rotations).