AVL Trees

Defn: A binary tree is a height-balanced p-tree if for each node in the binary tree, the difference in the height of the left and right subtrees is at most p.

Defn: An AVL (Adelson-Velskii, Landis) tree is a binary search height-balanced 1-tree.

Defn: The balance factor of a node, BF(node), in a binary tree is the difference of the left and right subtrees, hL - hR.

For any node in an AVL tree, the balance factor is either -1, 0, or 1.

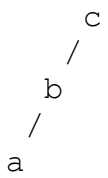
After inserting a new value into an AVL tree, if any node has a BF other than -1, 0, or 1, the AVL tree must be rebalanced.

The AVL tree is rebalanced at the closest ancestor, of the inserted node, that has a BF of -2 or +2. We will call the closest ancestor of the inserted node the pivot node, P.

Four basic rotations are possible where two are single rotations and two are double rotations. Mastering these rotations makes the balancing of an AVL tree pretty easy!!!

Rotation #1: LL (the new node is inserted in the left subtree of the left subtree of the pivot node)

Figure 1 (c is the pivot node)



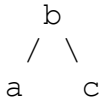
In the the case of an LL, a single rotation right around the pivot node is needed to balance the tree.

LL Rebalance Algorithm:

1. Rotate right so that b becomes the new root
2. The leftChild of c points to the rightChild of b
3. The rightChild of b points to c

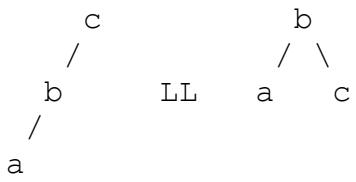
The result of balancing the tree in Figure 1 is:

Figure 2 (balanced Figure1 AVL tree)



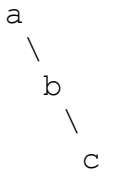
When rebalancing a tree, we will show the unbalanced tree folllowed by the type of rebalance necessary followed by the rebalanced AVL tree. Here is an example:

Figure 3 (proper rebalance notation)



Rotation #2: RR (the new node is inserted in the right subtree of the right subtree of the pivot node)

Figure 4 (a is the pivot node)

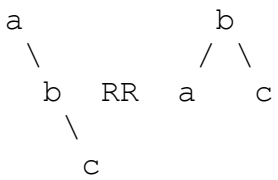


In the the case of an RR, a single rotation left around the pivot node is needed to balance the tree.

RR Rebalance Algorithm:

1. Rotate left so that b becomes the new root
2. The rightChild of a points to the leftChild of b
3. The leftChild of b points to a

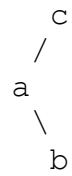
Figure 5 (balanced Figure 4 AVL tree)



Rotation #3: LR (the new node is inserted in the right subtree of the left subtree

of the pivot node)

Figure 6



Q1: What is the pivot node? Why?

LR Rebalance Algorithm

1. Rotate left around a
2. Rotate right around c

Figure 7 (balanced Figure 6 AVL tree)

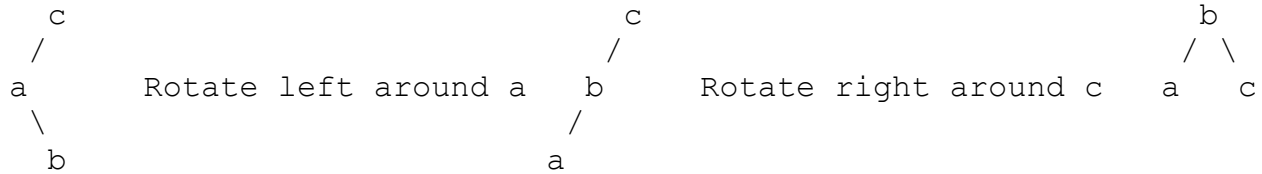
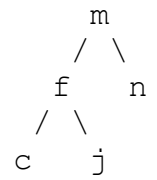


Figure 8



Q2:Is the tree in Figure 8 an AVL tree? Why or why not?

Q3: Does the tree need any kind of rebalancing? If so, rebalance the tree.

P1: Insert g into the tree.

Q4: Does the resulting tree need rebalancing? Why or why not? If so, rebalance

the tree.

P2: Starting over with the tree in Figure 8, insert a into the tree.

Q5: Does the resulting tree need rebalancing? Why or why not? If so, rebalance the tree.

Rotation #4: RL (the new node is inserted in the left subtree of the right subtree of the pivot node)

Figure 9

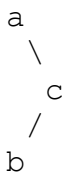


Figure 10 (balanced Figure 9 AVL tree)