AVL Tree Insertion

As discussed in class, AVL trees are height-balanced (HB[1]) trees which allow insertion and deletion of nodes as well as search for a node with a particular key in $O(\log n)$ time, worst case. To do this, we need to add some rules for balancing AVL trees when an insertion of a new item WOULD lead to an unbalanced tree. You may (should) recall that there are 4 different types of rotations required, based upon where the inserted node appears relative to the lowest-level tree node which becomes unbalanced. While the Weiss text describes the rotations, I think another text (Data Structure Techniques by Thomas Standish) does a better job. The following description of the balancing rotations is taken (loosely) from Standish.

We should remember that:
- balancing is only required when an insertion of a new node creates a (local) unbalance (a balance field for a node whose absolute value = 2) and
- the rotation required to rebalance the tree is performed locally (with respect the THE (first) node which becomes out of balance.)

In the following descriptions, A refers to the lowest-level node which becomes unbalanced (balance factor = 2 or = -2), B is a node into whose subtree the new item is inserted, and C (sometimes omitted) is another node in the tree which has a relationship to A and B. I'll denote general subtrees with $t_1$, $t_2$, $t_3$, and $t_4$. (Note that $t_1$, $t_2$, $t_3$, and $t_4$ are themselves height balanced (avl) trees.) In a sense A, B, and C are the only "special" nodes here. That is why the subtrees $t_1$, $t_2$, $t_3$, and $t_4$ are treated as individual entities. During the rotations each of $t_1$-$t_4$ is moved as a whole (by exchanging pointers ?), while A, B, and C are individual nodes that get moved around in the tree in order to make the balancing come out right. So, given all of that, the following diagrams (from Standish) show the 4 rotations.

1) LL

```
       A                          B
      / \                          / \  
     B   t3  ==>  t1  A
    / \                      / \   
   t1  t2                  t2  t3
```

In this case, it must be that an insertion into $t_1$ has increased the height of $t_1$ and thus caused the tree to become (momentarily) unbalanced (A’s balance becomes +2).

(Why must we have inserted into $t_1$ and not $t_2$ I hear you ask? Because this is an LL rotation. Had it been a new node in $t_2$ which caused A to become unbalanced, then we’d need to use the LR rotation shown below).

Note that $t_1$ can be a tree of arbitrary depth, but that inserting the new node must have increased $t_1$’s depth.
2) RR (symmetric to LL)

```
A                     B
/ \                  / \
t1 B  ==>           A  t3
/ \                  / \
t2 t3               t1 t2
```

In this case, it must be that an insertion into t3 has caused the tree (at A) to become (momentarily) unbalanced (with balance = -2.) Again, t1, t2, and t3 can be trees of arbitrary depth.

Now, for the (slightly) trickier LR and RL cases...

3) LR

```
A                     B
/ \                  / \
C  t4                / \ C  A
/ \                  / \ / \ t1 t2 t3 t4
/ \                  t1 t2 t3 t4
```

4) RL

```
A                     B
/ \                  / \
t1 C                / \ A  C
/ \                  / \ / \ t1 t2 t3 t4
B  t4  ==>           / \ / \ t1 t2 t3 t4
/ \                  t1 t2 t3 t4
```

Note that (unlike the RR and LL rotations) we cannot tell which subtree of B (t2 or t3) the "unbalancing" insertion fell into. It does not matter in terms of these rotations.