AVL Tree Pseudocode

Node Class

Should have a height variable.

Left and a Right pointer

And an Item

Insert method

```
Pass through the tree to find the correct position for insertion like in a normal BST.
Once you find it, adjust the height of the node.

You must then check the tree to see if it is unbalanced, this is easy with the height variable stored in the nodes.
If the height of the right subtree is larger than the left, we must rotate Left.
    Otherwise, we rotate right.

After doing so, we re-check the balance of the tree again
    If the tree is still unbalanced, reverse the rotation by rotating in the opposite direction
    Then rotate the child in the opposite direction, then rotate the node itself in the same direction.
        For example, if we rotate left and its unbalanced, rotate right, rotate child right, rotate node left.
```

Delete Method

```
Find the item, delete it.
If it's a leaf, we can just remove it.
If not, we must traverse down its children, if it only has 1 child go that way.
    Else, we must get the leftmost node of the right subtree, the inorder successor.
        Swap the two items, and then delete this leaf.

At this point we can adjust height and check for imbalances.
If an unbalance is found, you do the same procedure as in insertion
```

Insert and Delete are definitely recursive methods.

Rotate Method

Rotate Left:

Make a new root, which is the right child of the original root.

The new roots left child is the original root.

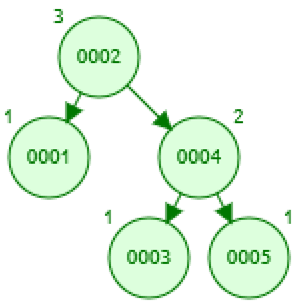
The new roots left child (the original root)'s right children are now the original roots left children.

Rotate Right:

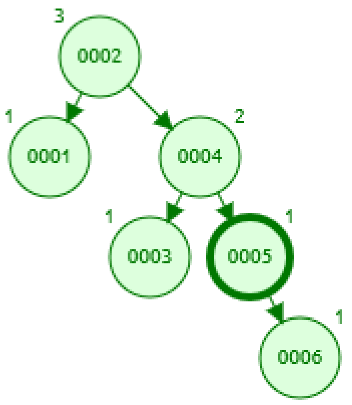
The opposite of Rotate Left

Adjust heights after both rotate operations.

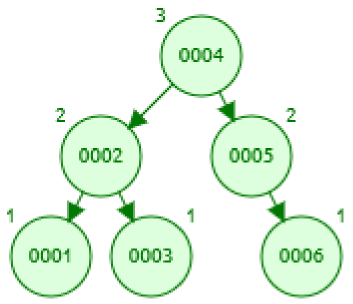
Example 1 – Insert 6



Initial Tree

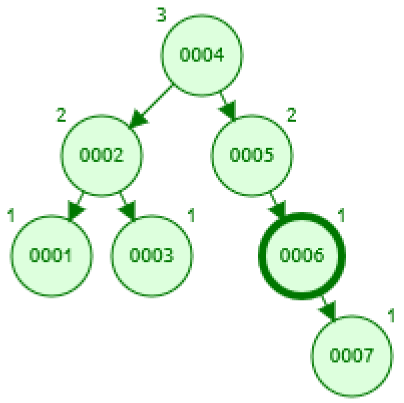


Inserted 6

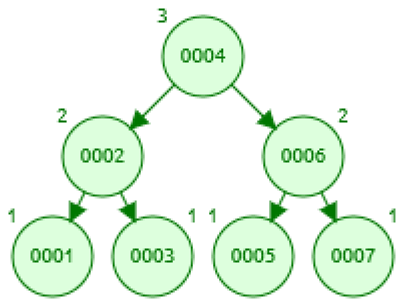


Single Rotate Left

Example 2 – Insert 7

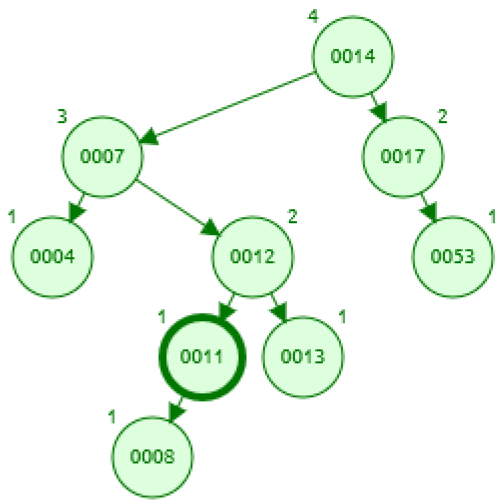


Inserted 7

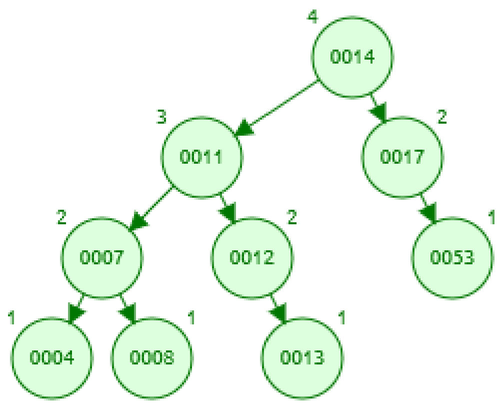


Single Rotate Left

Example 3 – Insert 8



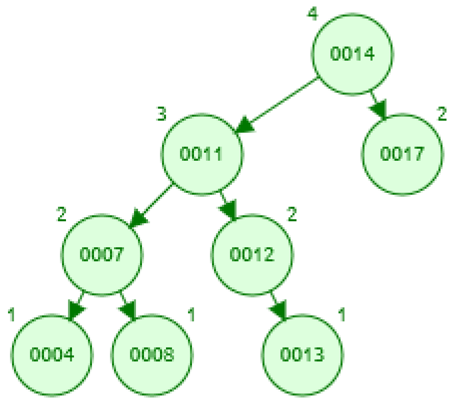
Insert 8



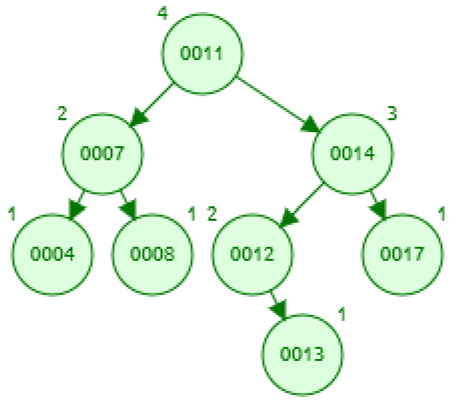
Double Rotate Left

Tyler Wilding

Example 4 – Delete 53

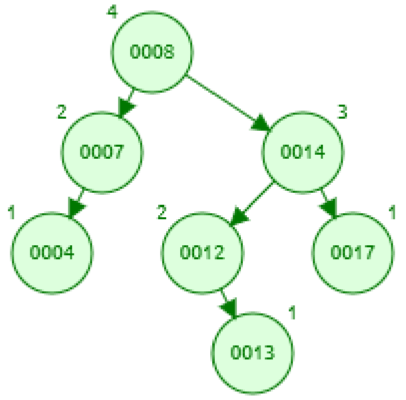


Deleted 53

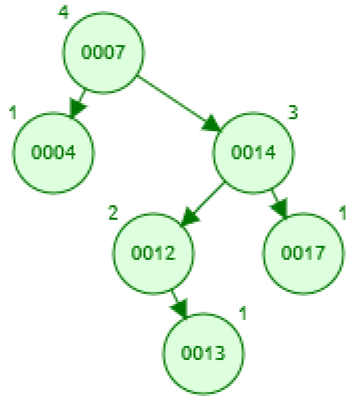


Single Rotate Right

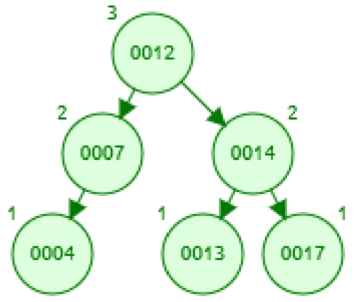
Example 4 – Delete 11 and 8



Deleted 11



Deleted 8



Double Rotate Left

Definition of Full:

A tree is full all of it's non-leaf nodes have the maximum amount of children it can have, it is not when the lowest level has the maximum number of nodes, that is a tree that is both complete and full.

About Debugging:

At this point, not learning how to use a debugger is a very big disadvantage, in problems like these, it makes finding small errors such as in insertion or deletion much much easier. The era of using print lines to debug a program is pretty much over, especially in a language like Java where every IDE has a competent debugger, you should get comfortable using it. Eclipse's isn't the best I've seen, but I can't recommend IntelliJ's debugger enough. Additionally, using a debugger will help you out a lot when you take Assembly language or other classes which throw you into a language you aren't that strong in, the debugger will allow you to not only see your problems, but understand how the language works.