**AVL Trees**

An AVL tree is a special kind of binary search tree that automatically makes sure the tree remains balanced at all times.

Define the *balance factor* for each node of a tree as

\[
\text{balance factor} = \text{height of the left subtree} - \text{height of the right subtree}
\]

A tree is considered “balanced” if the balance factor of all nodes is -1, 0, or 1. (The most unbalanced a tree can get is 2 or -2, because we would immediately rebalance it before adding/deleting any more nodes.)

Ex.
**Left rotation of a node**

If the balance factor of a node is -2, then the right subtree is 2 deeper than the left subtree. If the balance factor of the right child is also negative, do a **left rotation** on that node. (OF A NODE: have to do this recursively)

1. Promote the right child to be the new root.
2. Move the old root to be the left child of the new root.
3. If the new root already has a left child, make it the right child of the new left child. (Guaranteed to be vacant)
**Right rotation of a node**

If the balance factor of a node is +2, then the left subtree is 2 deeper than the right subtree. If the balance factor of the left child is also positive, do a right rotation on that node.

1. Promote the left child to be the new root.
2. Move the old root to be the right child of the new root.
3. If the new root already has a right child, make it the left child of the new right child. (Guaranteed to be vacant)
Special cases

In our previous example, the balance factors of the old root node and the new root node were the same sign. Consider the case when they are not:

• Instead of doing a left rotation, do a right rotation on the new root node first, and then the left rotation.
• Instead of doing a right rotation, do a left rotation on the new root node first, and then the right rotation.
Example:

\[
\begin{array}{cccccc}
30 & 40 & 20 & 35 & 50 \\
\end{array}
\]

Add 45

Add 15, 10  
always start rebalancing from the bottom up

RR on 20

LR on 30