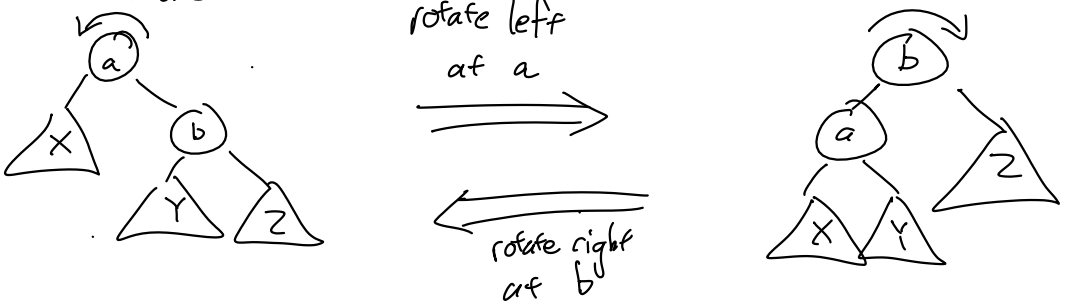
AVL tree:

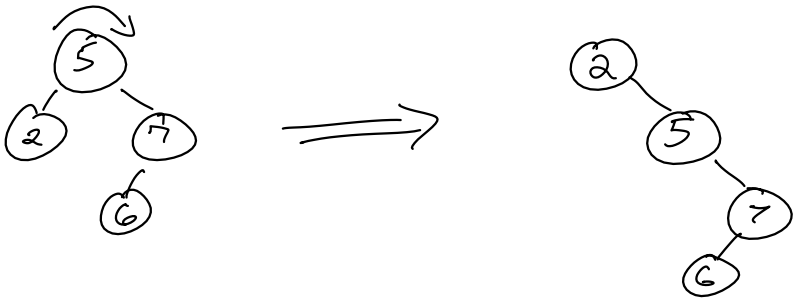
a BST s.t. at every node, heights of left & right subtrees differ by at most 1

BST:

a binary tree s.t. at every node, all left descendants are smaller & all right descendants are larger

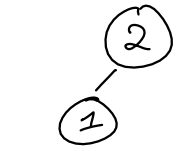
Tree rotation



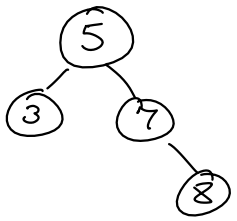


Rotation preserves the BST invariant

No maximal rotation; no guarantee that tree can be rotated

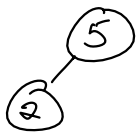


insert 0



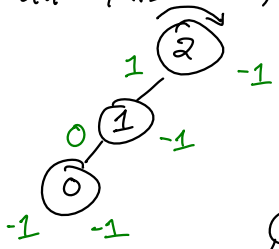
insert 9

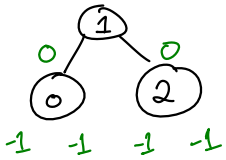
insert 3

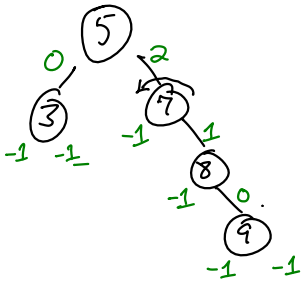


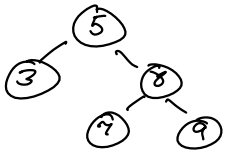
insert 4

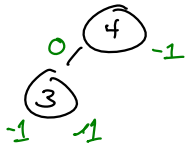
all AVL trees; should stay AVL trees

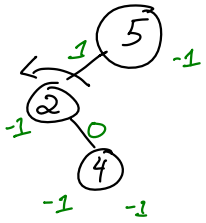


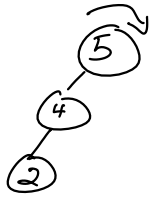


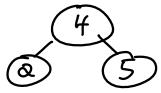




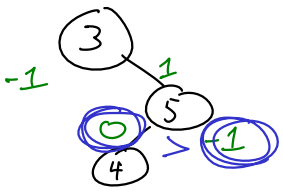


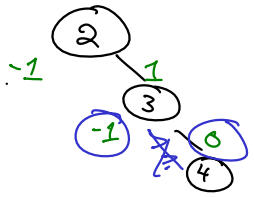


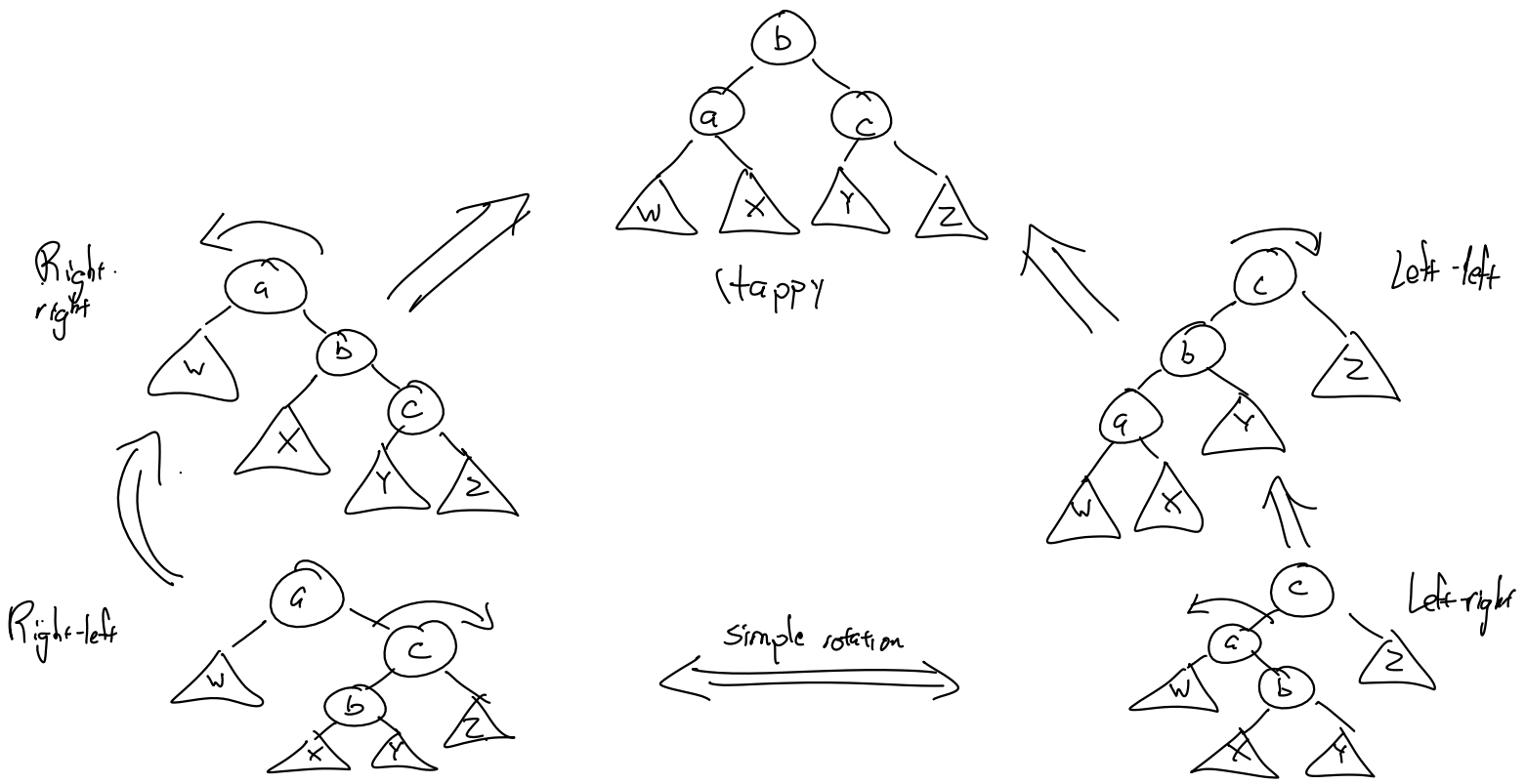




```
Function rebalance(node):
    hL ← node.left.height
    hR ← node.right.height
    If hL - hR ≤ -1:
        If node.right.left.height > node.right.right.height:
            node.right ← rotateRight(node.right)
        node ← rotateLeft(node)
    Else hL - hR > 1:
        If node.left.right.height > node.left.left.height:
            node.left ← rotateLeft(node.left)
        node ← rotateRight(node)
    EndIf
    Return node
EndFunction
```






b
a
c
w
x
y
z
Happy
Right-right
a
b
w
c
x
y
z
Left-left
c
b
z
a
y
w
x
Right-left
a
c
w
b
z
x
y
Simple rotation
Left-right
c
a
z
w
b
x
y

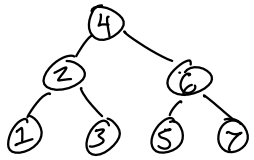
BSTs (balanced: height is $O(\log n)$)

BSTs are an implementation of dictionary (among others)

get(key)
insert(key, value)
update(key, value)
remove(key)

} $O(\text{height})$

Traversal



pre-order
post-order
in-order
level-order

Pre-order : recursively pre-order left; recursively pre-order right; visit root

1 3 2 5 7 6 4

Post-order : visit root, then recurse

4 2 1 3 6 5 7

In-order : recurse left, root, recurse right

1 2 3 4 5 6 7

Level-order : visit left to right on all levels top to bottom

4 2 6 1 3 5 7