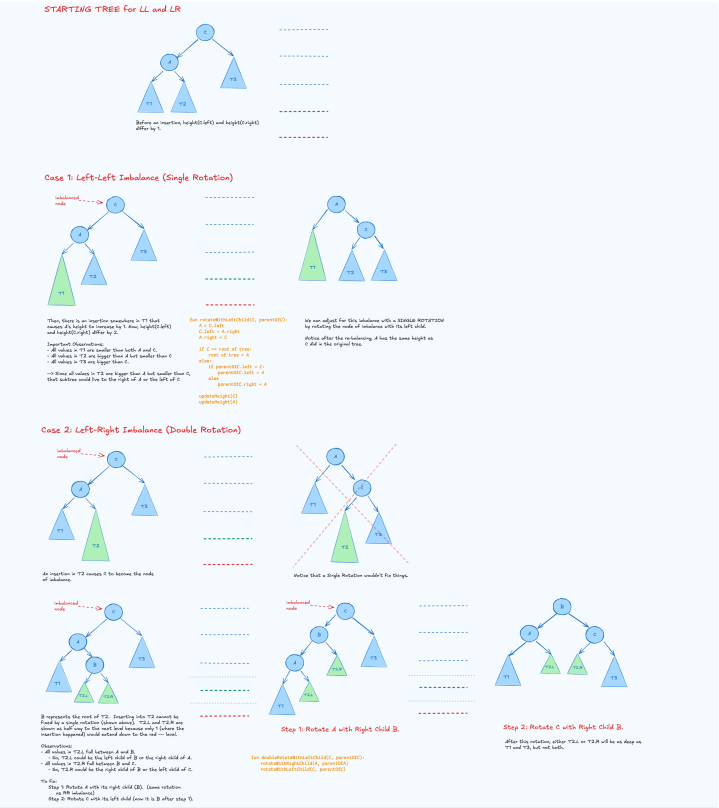
STARTING TREE for LL and LR
C
A
T1
T2
T3
Before an insertion, height(C.left) and height(C.right) differ by 1.
Case 1: Left-Left Imbalance (Single Rotation)
imbalanced node
C
A
T1
T2
T3
Then, there is an insertion somewhere in T1 that causes it's height to increase by 1 thus, height(C.left) and height(C.right) differ by 2.
Important Observations:
- All values in T1 are smaller than both A and C.
- All values in T2 are bigger than A but smaller than C
- All values in T3 are bigger than C.
--> Since all values in T2 are bigger than A but smaller than C, that subtree would live to the right of A or the left of C
fun rotateRight(A,C,parentOfC):
A = C.left
C.left = A.right
A.right = C
if C == root of tree:
root of tree = A
else:
if parentOfC.left = C:
parentOfC.left = A
else:
parentOfC.right = A
updateHeight(C)
updateHeight(A)
A
C
T1
T2
T3
We can adjust for this imbalance with a SINGLE ROTATION by rotating the node of imbalance with its left child.
Notice after the rebalancing, A has the same height as C did in the original tree.
Case 2: Left-Right Imbalance (Double Rotation)
imbalanced node
C
A
T1
T2
T3
An insertion in T2 causes C to become the node of imbalance.
A
C
T1
T2
T3
Notice that a Single Rotation wouldn't fix things.
imbalanced node
C
A
B
T1
T2.L
T2.R
T3
B represents the root of T2. Inserting into T2 cannot be fixed by a single rotation (shown above). T2.L and T2.R are the subtrees to the left and right of B respectively (where the insertion happened) could extend down to the red — level.
Observations:
- All values in T2.L fall between A and B.
- So, T2.L could be the left child of B or the right child of A.
- All values in T2.R fall between B and C.
- So, T2.R could be the right child of B or the left child of C.
To fix:
Step 1: Rotate A with its right child (B). (same rotation as RR imbalance)
Step 2: Rotate C with its left child (now it is B after step 1).
imbalanced node
C
B
A
T1
T2.L
T2.R
T3
Step 1: Rotate A with Right Child B.
fun doubleRotateRightLeftChild(C, parentOfC):
rotateRightRight(A,B,C), parentOfC)
rotateRightLeftChild(C, parentOfC)
B
A
C
T1
T2.L
T2.R
T3
Step 2: Rotate C with Right Child B.
After this rotation, either T2.L or T2.R will be as deep as T1 and T3, but not both.