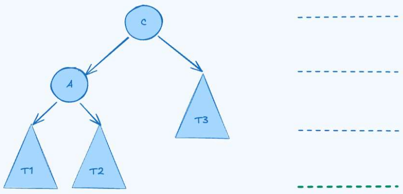
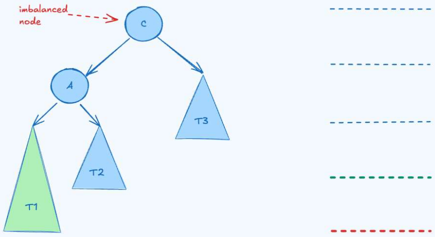
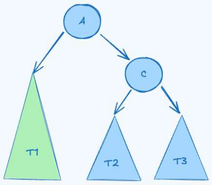
STARTING TREE for LL and LR



Before an insertion, height(C.left) and height(C.right) differ by 1.

Case 1: Left-Left Imbalance (Single Rotation)





Then, there is an insertion somewhere in T1 that causes A's height to increase by 1. Now, 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.

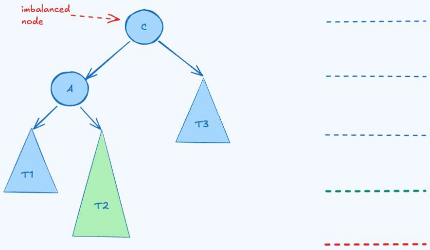
--> Since all values in T2 are bigger than A but smaller than C, that subtree could live to the right of A or the left of C

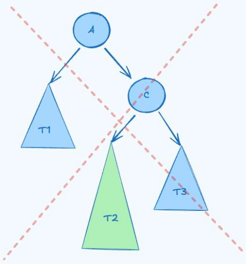
```
fun rotateWithLeftChild(C, parentOfC):
    A = C.left
    C.left = A.right
    A.right = C

    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)
```

We can adjust for this imbalance with a SINGLE ROTATION by rotating the node of imbalance with its left child.

Notice after the re-balancing, A has the same height as C did in the original tree.

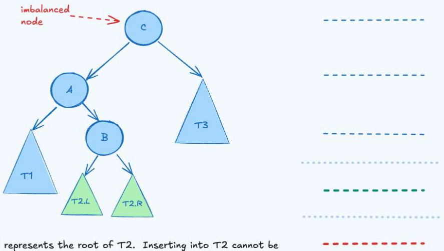
Case 2: Left-Right Imbalance (Double Rotation)

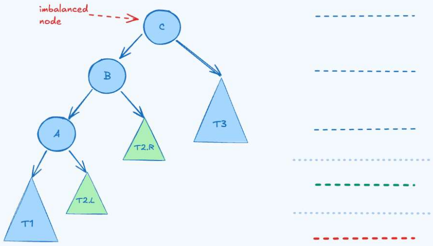


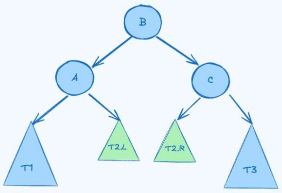


An insertion in T2 causes C to become the node of imbalance.

Notice that a Single Rotation wouldn't fix things.







B represents the root of T2. Inserting into T2 cannot be fixed by a single rotation (shown above). T2.L and T2.R are shown as half way to the next level because only 1 (where the insertion happened) would 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).

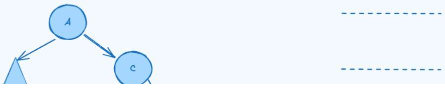
Step 1: Rotate A with Right Child B.

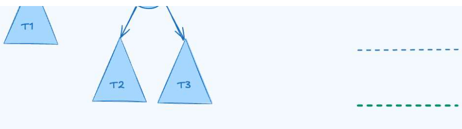
```
fun doubleRotateWithLeftChild(C, parentOfC):
    rotateWithRightChild(A, parentOfA)
    rotateWithLeftChild(C, parentOfC)
```

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.

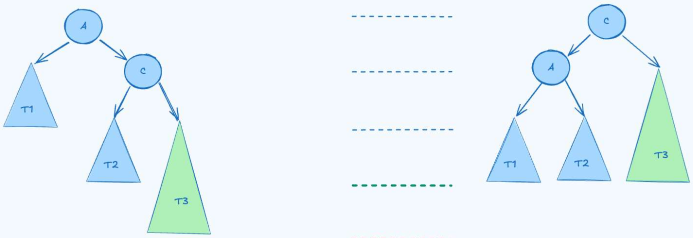
STARTING TREE For RR and RL





Before an insertion, height(A.left) and height(A.right)
differ by 1.

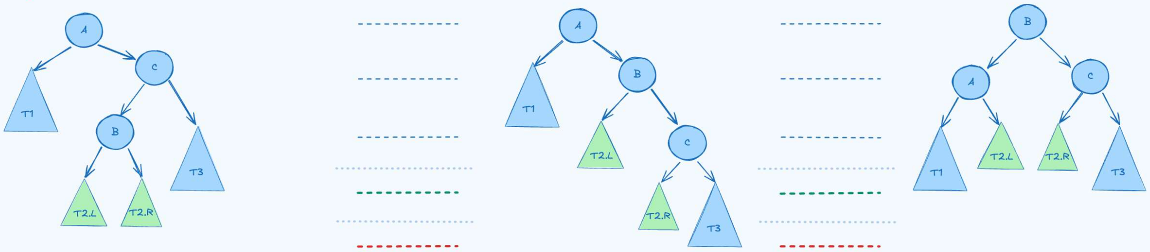
Case 4: Right Right Imbalance (Single Rotation)



Insertion into T3 causes it to be 2 levels deeper than T1,
making A the node of imbalance.

Remember: All values in T2 fall between A and C. So T2 could
be connected to the left of C or the Right of A.

Case 3: Right Left Imbalance (Double Rotation)



As with Case 2, an insertion into T2 cannot be solved with
a single rotation. So, we need to explicitly consider the root of
T2, labeled as B here.