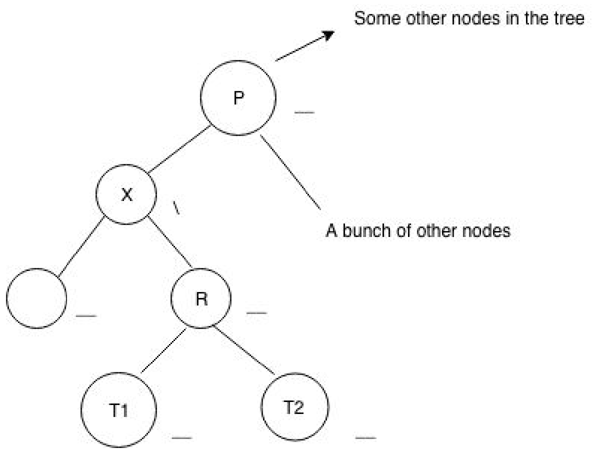
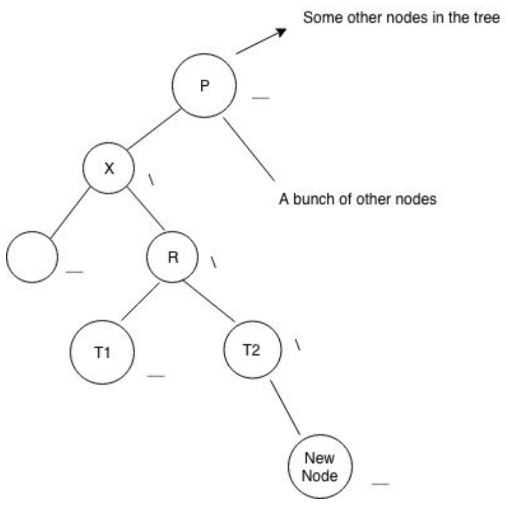
# Coding AVL rotations (Case 1)

Note: If you get a code on the exam, it'll most likely be case 1 but if you understand how to code case one (single left and right rotations), then the code for left-right and right-left should be intuitive. So let us get right into it.

One thing to keep in mind: Case one generally means that node X and r have the same balance factor ( after unblancing node has been inserted). Let us take a look at this AVL tree

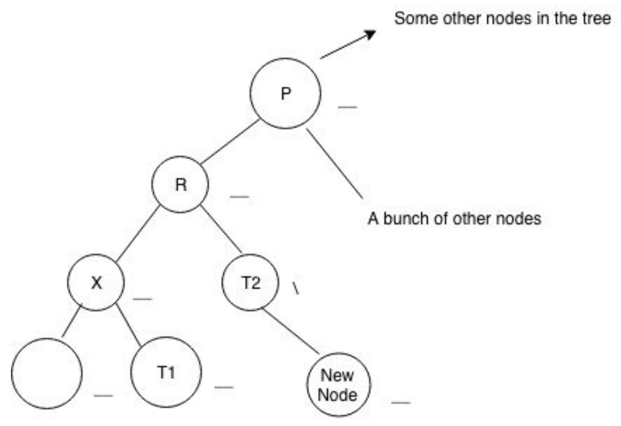


P is the parent node of the unbalanced node X and R is the right child of X. Now if we inserted a node on the right of R, we get this:



Now X is very unbalanced and needs a rotation around it. Well since X and R have the same balance factor, we only need to do a left rotation from R to X.

Algorithm: Well the first thing that we know for sure is that X is going to be the new left child of R. but wait, R already has a left child (T1), so what happens to it 🤔. Well the current left child of R is going to be the new right child of X. and also since R now takes the place of X in the tree, the parent of X now becomes the parent of R. This is what the tree looks like after the rotation:



You see, quite easy. In case of a right rotation, (1) X becomes the right child of X, (2) the old right child of R becomes the becomes the new left child of X (draw it out and see).

Now putting this in code should be trivial. How about we revisit the AVL tree question from recitation and give it a better/understandable implementation 😎 .

Assuming you have the following class:

```
public class AVLTreeNode<T extends Comparable<T>> {
    public T data;
    public AVLTreeNode<T> left, right;
    public char balanceFactor;   // '-' or '/' or '\'
    public AVLTreeNode<T> parent;
    public int height;
 }
```

After an AVL tree insertion, when climbing back up toward the root, a node x is found to be unbalanced. Further, it is determined that x's balance factor is the same as that of the root, r of its taller subtree (Case 1). Complete the following rotateCase1 method to perform the required rotation to rebalance the tree at node x. You may assume that x is not the root of the tree.

```
public static <T extends Comparable<T>>

 void rotateCase1(AVLTreeNode<T> X) {
         // so we are already given node X, great!
        // first thing we must do is get node R
        // we know that if X is right high, then R is its immediate right child, if it is left high then
       // R is the immediate left child
        AVLTreeNode<T> R = null;

        If ( X.balanceFactor == '\') {

                R = X.right;
        }

        else{
                R = X.left;
        }

        // now we must have the parent of X point to R but first we must figure out if X is the left
        // or right child of its parent

        If ( X.parent.right == X ) {     // right child

                X.parent.right = R;    // have R become the new child
        }

        else {
                X.parent.left = R;
        }

        R.parent = X.parent;

        // ok now time for the rotations. If X is right high then we must do a left rotation, else
        // right rotation

        If ( X.balanceFactor == '\') {   // left rotation

                X.right = R.left;   // left child of R becomes the right child of X
                X.right.parent = X;
                R.left = X;      // X becomes the left child of R
                X.parent = R;  // R is now the parent of X
        }
        else { // right rotation

                X.left = R.right // right child of R becomes left child of X
                X.left.parent = X;
                R.right = X;  // X becomes the right child of R
                X.parent = R;

        }

        //we are done with the rotation
        // decrease the height of X since it goes down a level in the tree
        X.height--;

        // update the balance factor of X and R to balanced. This is always the case after a
        // rotation. I do not have the time to prove it, so just know that is the case

        X.balanceFactor = '__' ;
        R.balanceFactor = '__' ;
}
```

We are done. This is probably the most difficult thing that can show up on the exam. If you fully understand this then you should walk into that exam feeling like Thanos. Goodluck.