Theory Problems 6

Tree Algorithms

Problem 1: In-order BST

Let T be a binary search tree containing elements $X = \{x_1, x_2, \ldots x_n | x_i \in \mathbb{R}, x_i \neq x_j \ \forall i, j\}$ and using the usual total order '$\leq$'. Prove that in-order traversal of any such tree produces the same sequence of vertices.

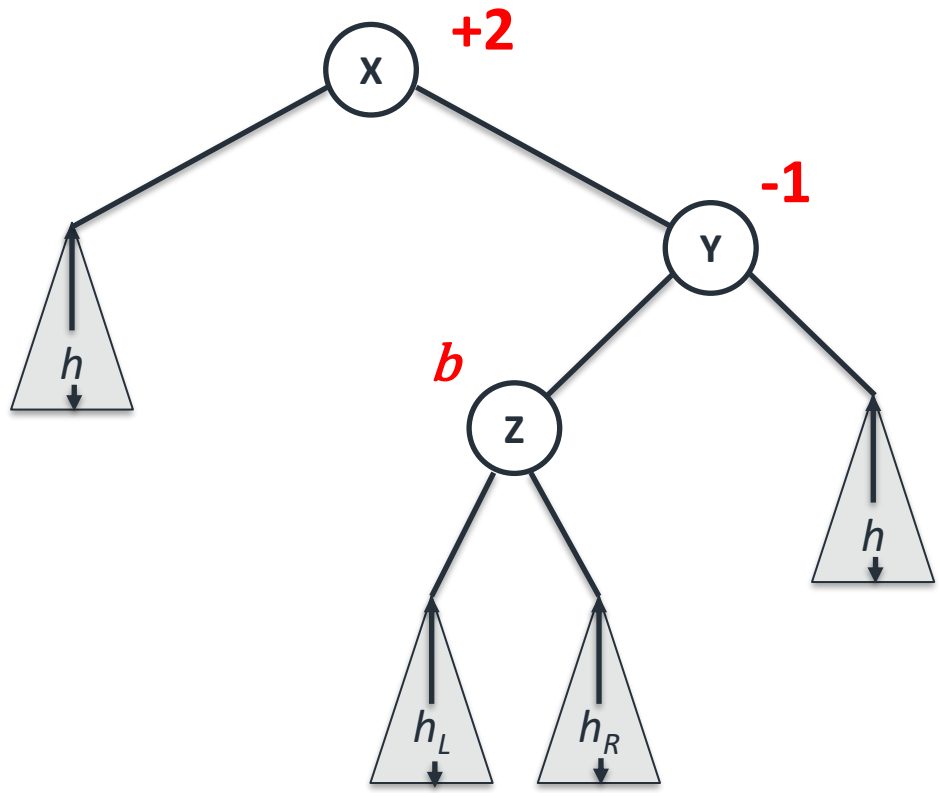
Problem 2: Tree Rotations

Using the given Java code, implement the functions **`LeftRotation`** and **`RightRotation`** as described in the lecture. The tree is doubly-linked, so you will need to fix the parent and child links.

Problem 3: Balance after rotation

The BST on the right is AVL-imbalanced. Balance figures are given in red, and the triangles represent subtrees of the given heights. Given that X is the only AVL-imbalanced vertex,

1. What values of h_L, h_R are consistent with the two known values of balance, and what is the value of b in each case?
2. Carry out the rebalancing procedure on this tree and give the new tree.
3. For each of the cases in (1), what are the new balances of X, Y and Z?



Problem 4: Rebalancing

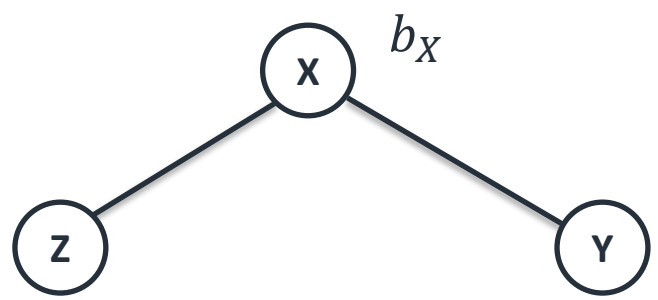
To rebalance the tree, we need to compute the balance. The naïve algorithm would compute the depth and balance of the tree at each insertion.

This would take $O(n)$ for each insertion (or even $O(n \log n)$ if the algorithm is implemented badly) so the complexity is $O(n^2)$ to construct the tree.

We need to do better than this.

On the right, we assume we add a new item Y on the right of X

The reverse argument applies on the left.



Before adding Y	**After Y**
Z does not exist, $b_X = 0$	$b_X = +1$, Tree X taller
Z exists , $b_X = -1$	$b_X = 0$, No change in height

Algorithm: Propagate the balance upwards, if Y on right, $b_X = b_X + 1$, otherwise Y is on the left and $b_X = b_X - 1$. If $b_X = 0$ we are done, otherwise tree X is taller and we propagate the change to the parent.

Problem 4: Rebalancing

If we propagated the change to the parent (W), then X is taller.

If X is a right child, we must increase $b_W = b_W + 1$

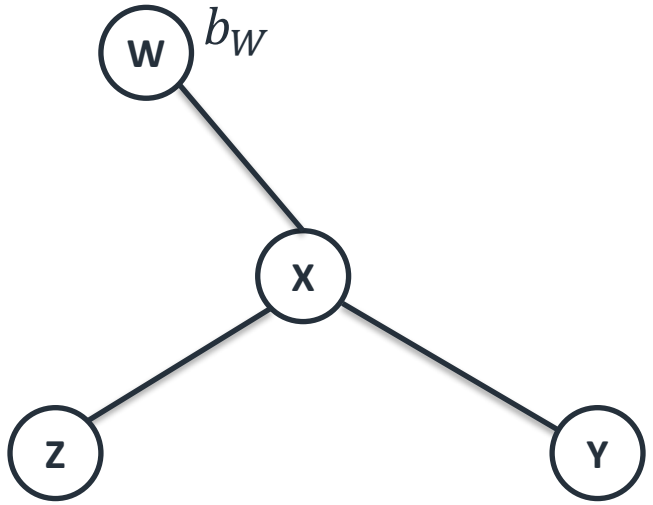
If X is a left child, we must decrease $b_W = b_W - 1$

Again, if we find $b_W = 0$, the height of tree W does not change, we are done.

If $|b_W| = 2$, we should rebalance.

Otherwise, propagate the change to the parent.

Since the total height is $O(\log n)$ this process will finish in $O(\log n)$ operations.



Algorithm: Propagate the balance upwards, if X on right, $b_W = b_W + 1$, otherwise X is on the left and $b_W = b_W - 1$.
If $b_W = -2$ or $b_W = 2$, rebalance W to get $b_W = 0$, we are done
If $b_W = 0$ we are done, otherwise tree W is taller and we propagate the change to the parent.

Problem 4: Rebalancing

In the the java function Rebalance, implement this algorithm and the algorithm described in the lecture to rebalance the tree.