Trees

Binary Search Tree (BST)

Binary Search Tree is a Binary Tree where for every node `v` in the tree:

```
v.left.value < v.value < v.righ t.value
```

Traversals:

Preorder	Current | Left | Right
Inorder	Left | Current | Right
Postorder	Left | Right | Current

Level Order Traversal:

- Create a queue and insert the root into it.

- Iterate over the queue until it's empty.

- In every iteration, we will pop from the top of the queue and perform the wanted action on the popped node. Then, we will add its left and right sons to the end of the queue.

Notes

Inorder traversal go over the values in a sorted order.

Time Complexity

Traversals: **O(n)** where n is number of nodes

Find, Insert, Delete: **O(h)** where h is the tree's height

AVL Tree

An AVL Tree is a balanced Binary Search Tree. That is, for every node `v` in the tree:

```
|BF| = |left.h eight - right.h eight| ≤ 1
```

Rotations:

An AVL Tree is kept balanced by performing rotations whenever needed.

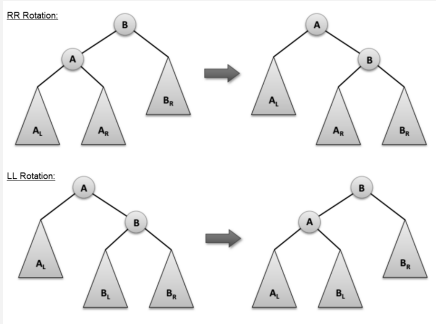
There are 4 types of rotations: LL, RR, LR, RL.

RR and LL are the base rotations and can be used to perform the other two.

Notes

The Height of an AVL Tree: h = **O(logn)**

AVL Base Rotations



Time complexity for every rotation: **O(1)**

BST Successor Algorithm

Successor algorithm finds the minimal value in a BST which is greater than a given value k.

The Algorithm:

- Initialize a varriable: `successor = null`

- Initialize a variable `current = root`

- While `current != null`:

- If `k < curren t.value`, set `successor = current` and go to `curren t.left`. Otherwise, go to `curren t.right`.

- Finally, return `successor`

Time Complexity

The algorithm goes from the root to a leaf in the worst case, so the time complexity is: **O(h)**.

If the tree is balanced (like AVL Tree) then the time complexity is: **O(logn)**.