# Splay Trees

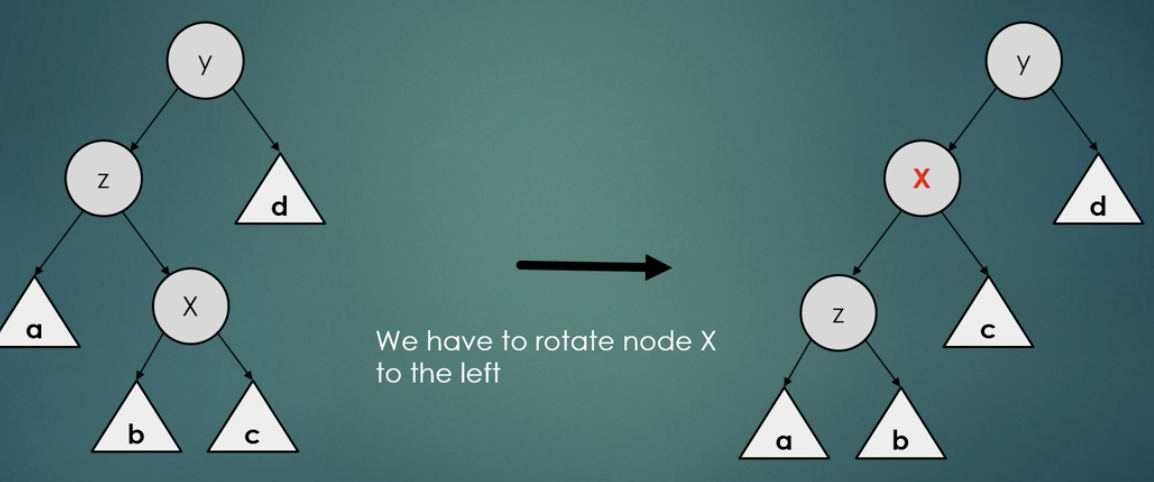
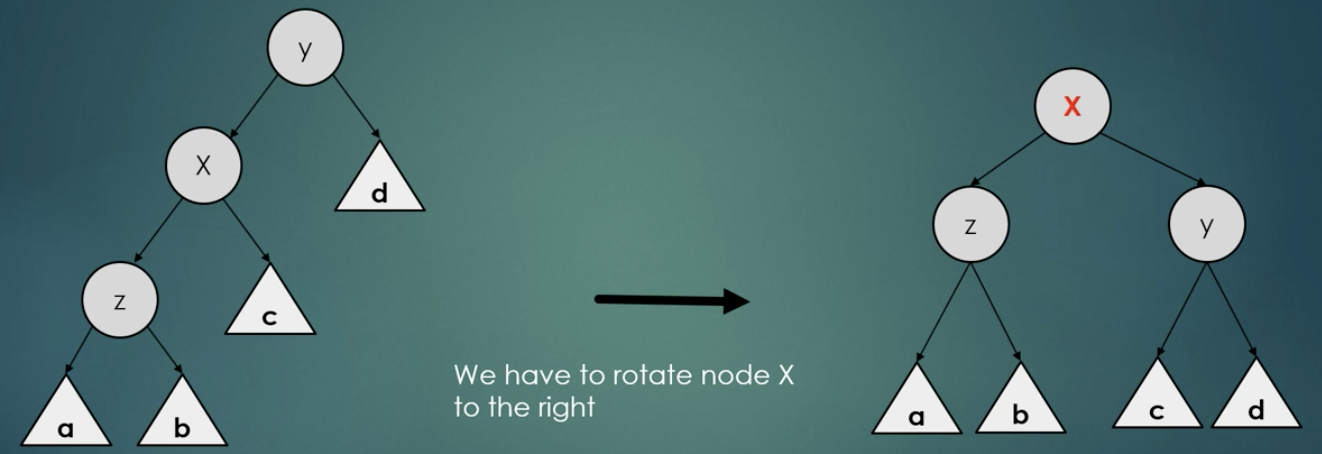
## Properties

* Used for caches.
* Its same as BST with two children for every node and left is small and right is large.
* Most of the operations have O(logN) time complexity but some operations are O(N).
* Unlike AVL Trees, they are not strictly balanced.
* Hence it is faster to construct.
* Easier to implement.
* Splay trees are kept balanced using rotations.
* **There is faster access to elements accessed recently.**
* **The aim of splaying is not to balance the tree but to get the node just accessed to the root.**

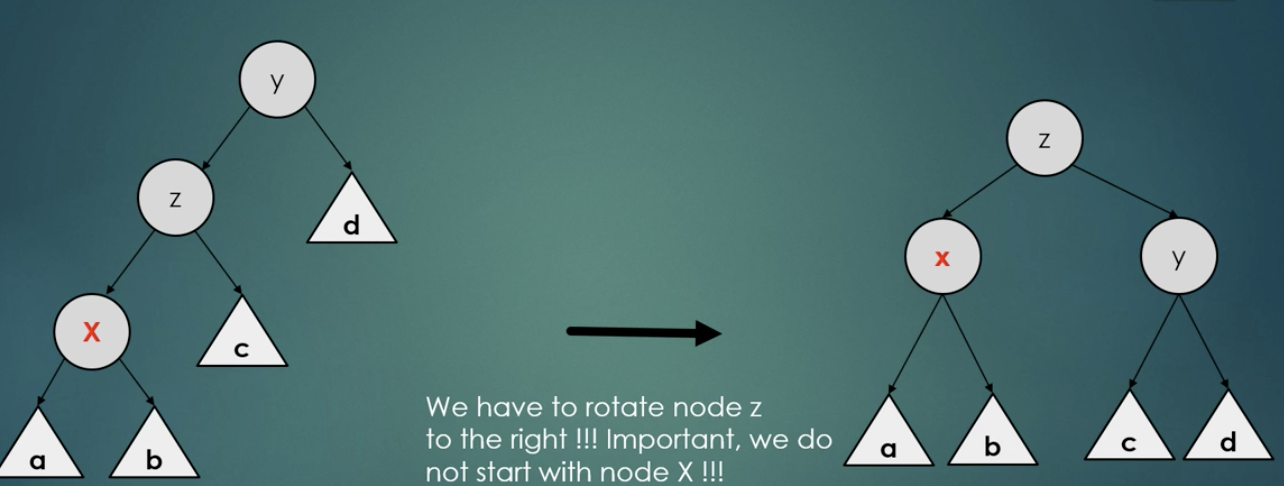
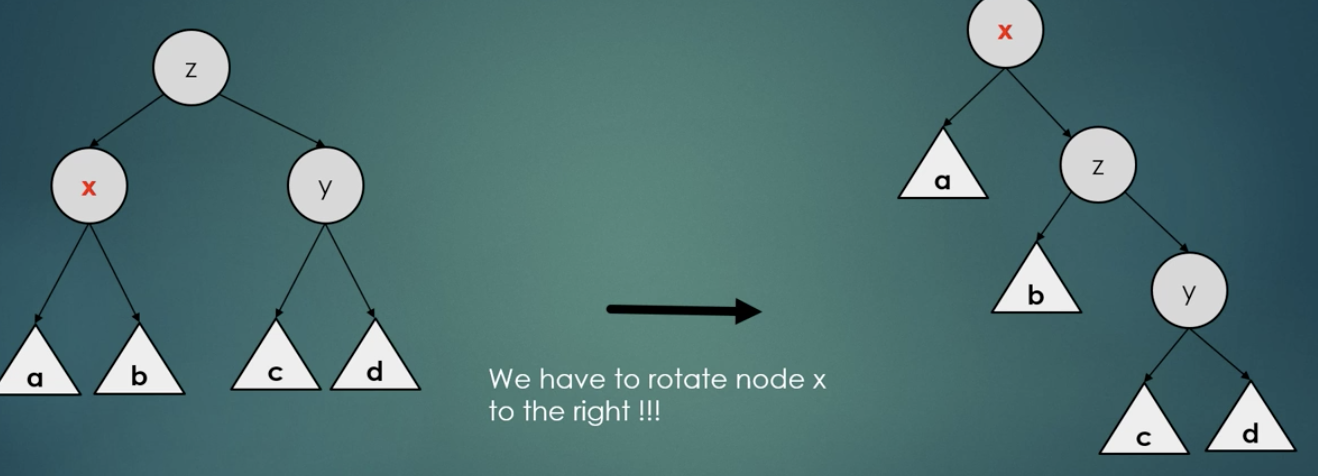
## Find Operation

* The search is like a standard Binary Search Tree Search operation.
* When we find an element, you make rotations so that the element becomes the root node.
* Hence for next search, it will be O(1) time.
* There are 3 ways we can make it happen:
  + Zig-Zag Situation
  + Zig-Zig Situation
  + Zig Situation

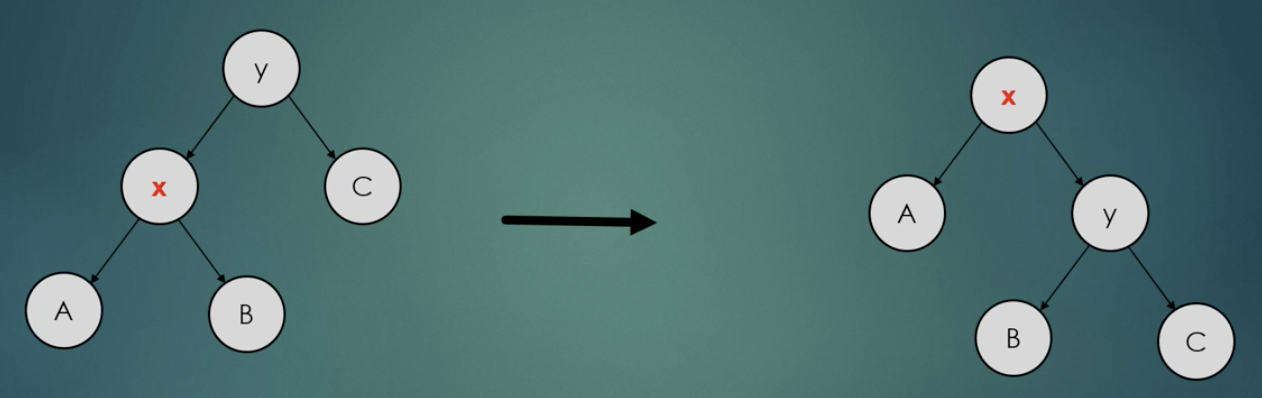
### Zig-Zag Situation

* The given node 'X' is a right child of a left child.
* The given node 'X' is a left child of a right child.
* 
* 'X' is first rotated left.
* 
* Then 'X' is rotated right.

### Zig-Zig Situation

* The given node 'X' is a left child of a left child.
* The given node 'X' is a right child of a right child.
* 
* We have to first rotate the parent right.
* 
* Then we rotate 'X' right.

### Zig Situation

* Sometimes when you are the left or right node of root, we just have to make single left/right rotation.
* So, in this case 'X' is the child of the root.
* 
* 