AVL Tree Rotations

- In an AVL tree (Adelson-Velsky and Landis tree), rotations are used to maintain the balance property after insertion or deletion. An AVL tree is a type of self-balancing binary search tree where the difference in heights between the left and right subtrees of any node (called the **balance factor**) is at most 1.

When an imbalance is detected (balance factor becomes +2 or -2), rotations are performed to restore balance. There are four types of rotations:

- Right Rotation (Single Rotation)
- Left Rotation (Single Rotation)
- Left-Right Rotation (Double Rotation)
- Right-Left Rotation (Double Rotation)

Right Rotation (Single Rotation)

This is used when there is a **left-heavy imbalance** (the left subtree of a node has more height).

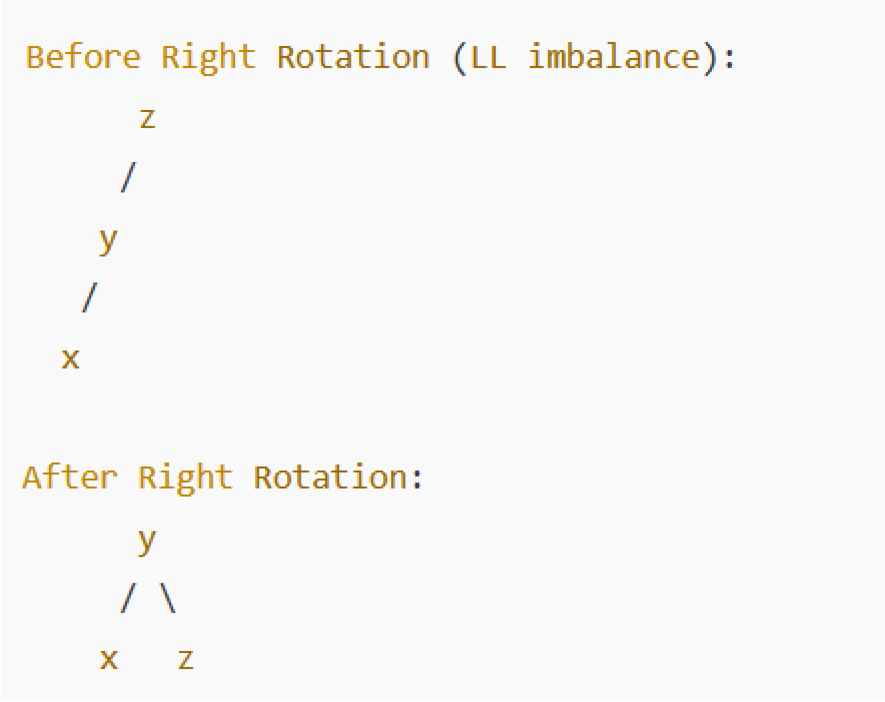
- **Case: Left-Left (LL) Imbalance**

If the imbalance occurs in the left subtree of the left child, we use a **right rotation**.

- **Steps**:

The left child of the unbalanced node becomes the new root.

The unbalanced node becomes the right child of the new root.



```
Before Right Rotation (LL imbalance):
      z
     /
    y
   /
  x

After Right Rotation:
      y
     / \
    x   z
```

Left Rotation (Single Rotation)

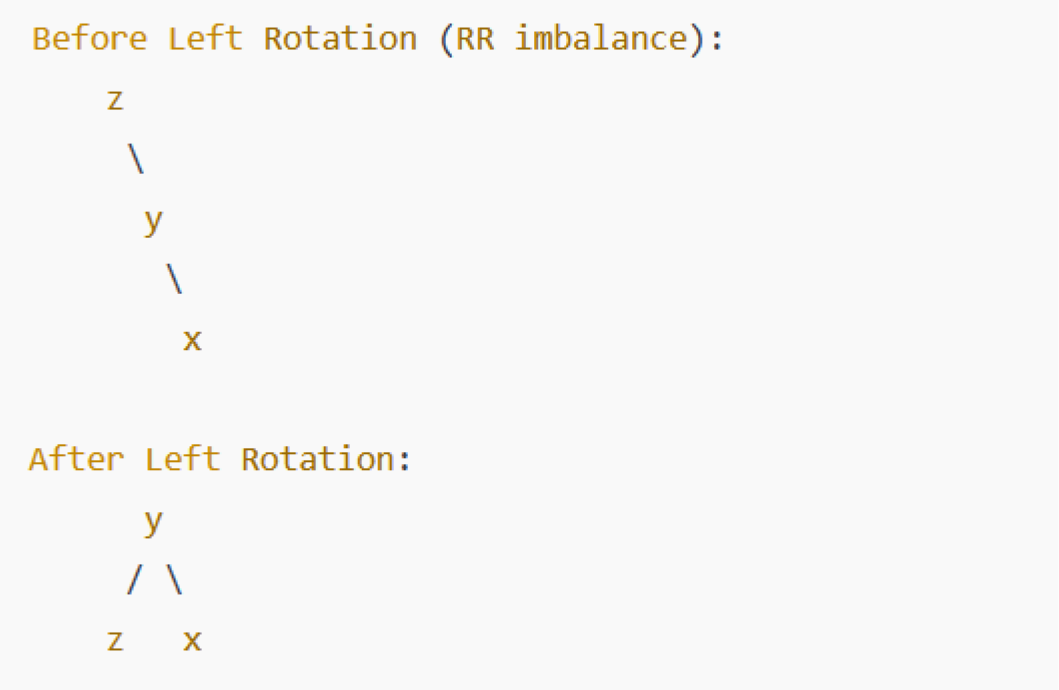
This is used when there is a **right-heavy imbalance** (the right subtree of a node has more height).

Case: Right-Right (RR) Imbalance

- If the imbalance occurs in the right subtree of the right child, we use a **left rotation**.

Steps:

- The right child of the unbalanced node becomes the new root.
- The unbalanced node becomes the left child of the new root.



```
Before Left Rotation (RR imbalance):
    z
     \
      y
       \
        x

After Left Rotation:
      y
     / \
    z   x
```

Left-Right Rotation (Double Rotation)

- This is used when there is a **left-right imbalance** (the left child has a right-heavy subtree).

Case: Left-Right (LR) Imbalance

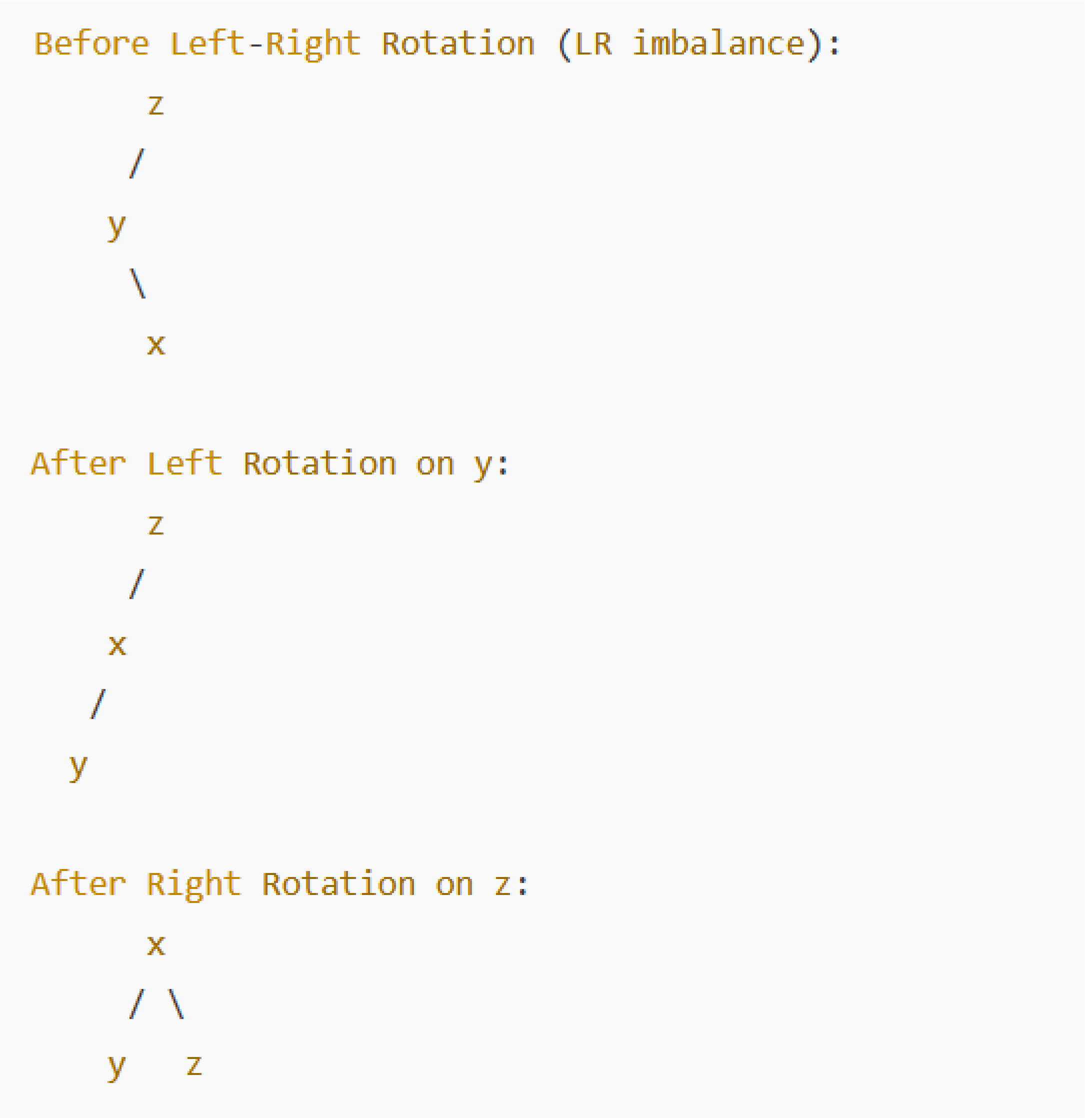
•The left subtree of the node has more height, but the right child of the left subtree is the problem.

•First, perform a **left rotation** on the left child, then a **right rotation** on the unbalanced node.

Steps:

1.Perform a left rotation on the left child.

2.Perform a right rotation on the unbalanced node.



```
Before Left-Right Rotation (LR imbalance):
      z
     /
    y
     \
      x

After Left Rotation on y:
      z
     /
    x
   /
  y

After Right Rotation on z:
      x
     / \
    y   z
```

Right-Left Rotation (Double Rotation)

This is used when there is a **right-left imbalance** (the right child has a left-heavy subtree).

Case: Right-Left (RL) Imbalance

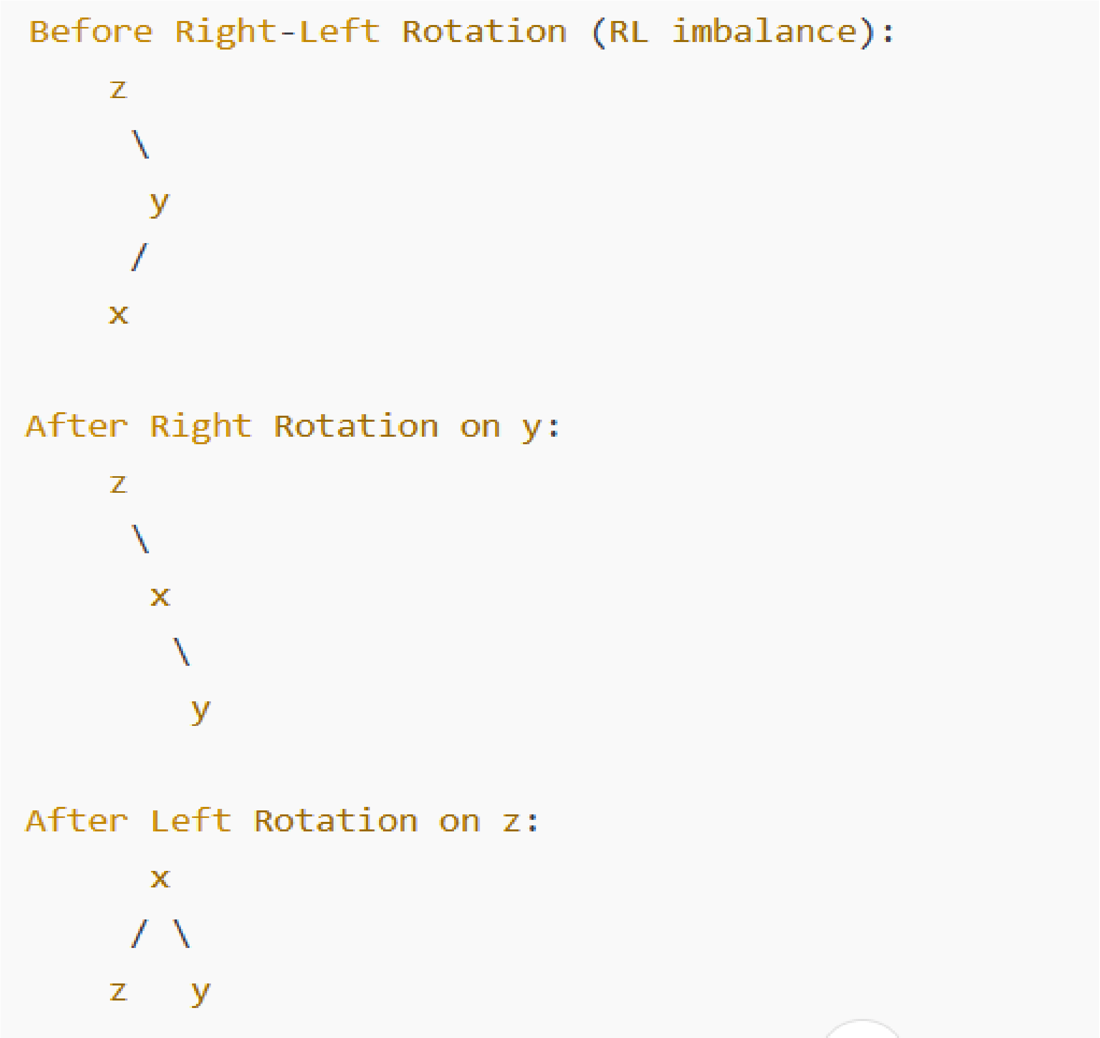
•The right subtree of the node has more height, but the left child of the right subtree is the problem.

•First, perform a **right rotation** on the right child, then a **left rotation** on the unbalanced node.

Steps:

1.Perform a right rotation on the right child.

2.Perform a left rotation on the unbalanced node.



```
Before Right-Left Rotation (RL imbalance):
     z
      \
       y
      /
     x

After Right Rotation on y:
     z
      \
       x
        \
         y

After Left Rotation on z:
       x
      / \
     z   y
```