# **Basics**

TL – Left subtree

TR – Right subtree

hR – Right tree height

hL – Left tree height

Balance factor = hR – hL

Tree is balanced if |Balance factor| <= 1

Balanced tree: balance factor = 0, 1 or -2

Unbalanced tree: balance factor = 2 or -2

An AVL node is "**left­heavy**" when bf = **­1**, "**equal­height**" when bf = **0**,

and "**right­heavy**" when bf = **+1**

Note:

**If balance < 0 – there is *at least 1 left* child.**

**If balance > 0 – there is *at least 1 right* child.**

# **Balancing**

## **Insert balance**

**Current is Right** – balance += 1

**Current is Left** – balance -= 1

Rotation criteria:

Balance is -2 (unbalanced left)

Left.Balance = 1: Rotate Left then Right

Otherwise: Rotate Right

Balance is 2 (unbalanced right)

Right.Balance = -1: Rotate Left

Otherwise: Rotate Right then Left

INSERT

Update all balances above while balance != +2 or -2

Current node is left? -1 : +1

Cases:

Z – 1st unbalanced node;

Y – child of Z

X – child of Y (grandchild of Z)

**Case 1: Left Left**

T1, T2, T3 and T4 are subtrees.

z y

/ \ / \

y T4 Right Rotate (z) x z

/ \ - - - - - - - - -> / \ / \

x T3 T1 T2 T3 T4

/ \

T1 T2

RotateRight(node)

**Case 2: Left Right**

z z x

/ \ / \ / \

y T4 Left Rotate (y) x T4 Right Rotate(z) y z

/ \ - - - - - - - - -> / \ - - - - - - - -> / \ / \

T1 x y T3 T1 T2 T3 T4

/ \ / \

T2 T3 T1 T2

1. RotateLeft(node.Left)
2. RotateRight(node)

**Case 3: Right Right**

z y

/ \ / \

T1 y Left Rotate(z) z x

/ \ - - - - - - - -> / \ / \

T2 x T1 T2 T3 T4

/ \

T3 T4

RotateLeft(node)

**Case 4: Right Left**

z z x

/ \ / \ / \

T1 y Right Rotate (y) T1 x Left Rotate(z) z y

/ \ - - - - - - - - -> / \ - - - - - - - -> / \ / \

x T4 T2 y T1 T2 T3 T4

/ \ / \

T2 T3 T3 T4

1. RotateLeft(node.Right)
2. RotateLeft(node)

+2: Rotate Left

-2: Rotate Right

## **Delete balance**

# **Rotations**

## **Rotate Right**

## **Rotate Left**