AVL Tree

By
Prof Ankush Kudale
Sinhgad Institute of Management

Introduction

- Till now we have studied binary tree and binary search tree with some operations.
- When we create any binary tree, with n elements, the tree will be generated as per the sequence of element.
- Such trees may have more elements either on left or right side, that means tree is **not balanced.**
- If the BST is complete balanced tree then this optimization can achieved.

Height Balanced Tree/AVL

- One of the popular balanced trees was introduced in 1962 by Russian Mathematicians **A**delson, **V**el’skii and **L**andis.
- They develop algorithm which balance BST.
- This tree has technique for efficient search and insertion, so AVL tree behaves near to complete binary search tree.

Height Balanced Tree/AVL

- Definition of **AVL**: An empty tree is always called height balanced. When tree T is non empty tree with subtree T_L and T_R as left sub tree and right sub tree, then T is height balanced if and only if T_L and T_R are height balanced . The tress T_L and T_R called height balanced if they satisfy conditions $| H_L - H_R | <= 1$, where H_L and H_R are height of trees T_L and T_R respectively.
- In other words, *tree is called height balanced if and only if balance factor of every node of that tree is either 0, 1 or -1.*

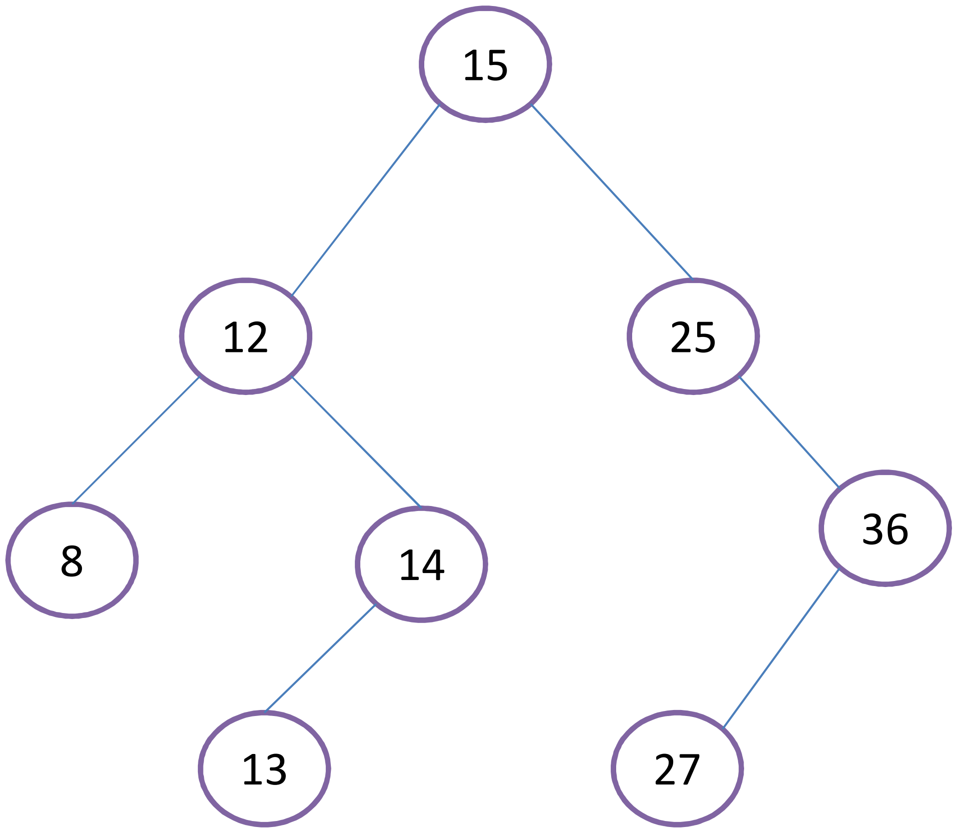
Balance Factor

- Where balance factor is difference between height of left sub tree and right sub tree.

$$\textbf{Balance Factor} = \left\{ \text{Height of left sub tree} \right\} - \left\{ \text{Height of right sub tree} \right\}$$

$$BF = H_L - H_R$$

Let us calculate BF



BF(15)=Height of left subtree-Height of right sub tee

BF(15)=Height(12)-Height(25)=3-3=**0**

BF(12)=Height(8)-Height(14)=1-2=**-1**

BF(14)=Height(13)-Height(NULL)=1-0=**1**

BF(13)=Height(NULL)-Height(NULL)=0-0=**0**

BF(8)=Height(NULL)-Height(NULL)=0-0=**0**

BF(25)=Height(NULL)-Height(36)=0-2=-**2**

BF(36)=Height(27)-Height(NULL)=1-0=**1**

BF(27)=Height(NULL)-Height(NULL)=0-0=**0**

Rule of tree rotations

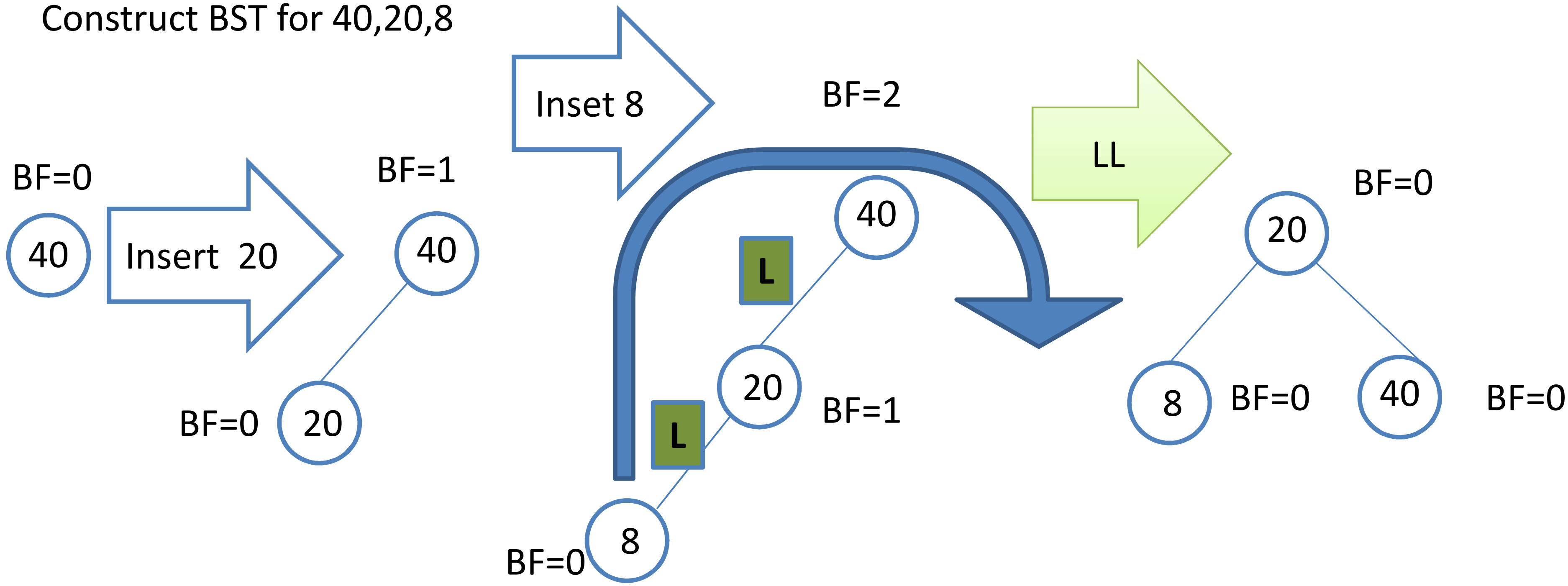
- If tree is not balanced, that means balance factor is not either 1 or 0 or -1. Then depending on imbalance the rotation rule is applied.
- LL rotation
- LR rotation
- RR rotation
- RL rotation

LL rotation

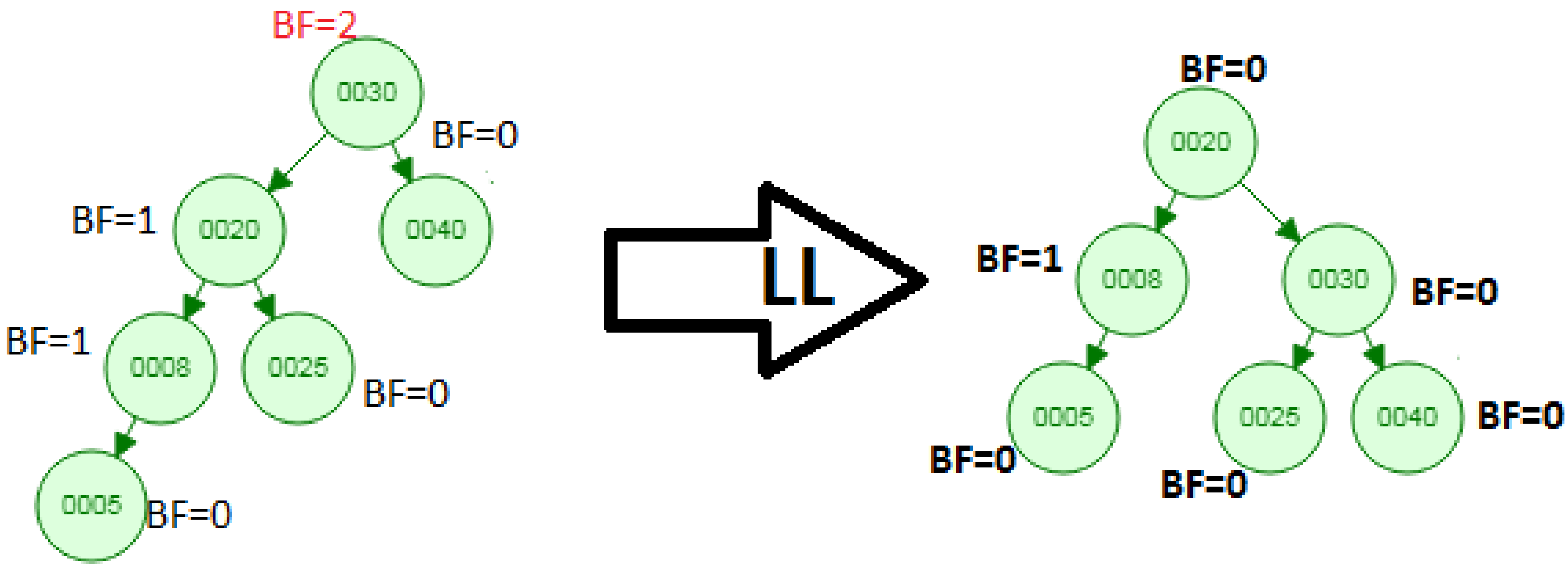
- After inserting a new node as a **left child in left subtree**, if the tree become unbalanced then the nodes in the tree must be rotate using LL rotation rule as follows:
- **Left child** of old root is becomes **NEW ROOT.**
- **Old root node** becomes **new right child** of new root
- Left child of new root is not changed
- If the **left child of old root** has right child, then it becomes **left child of old root**.

LL rotation example

Construct BST for 40,20,8



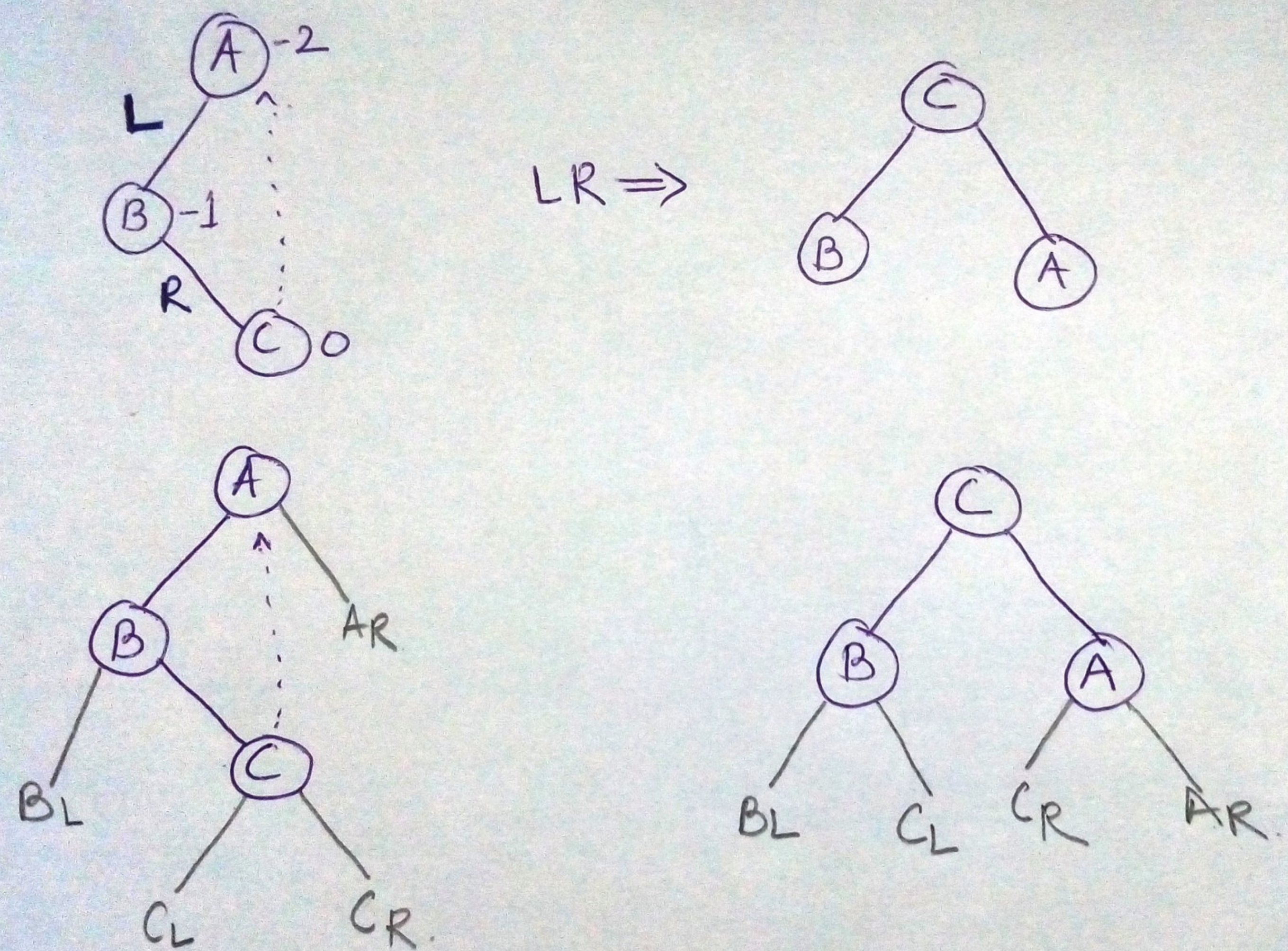
Another example



LR rotation

- If a new node is inserted as left or right in **right subtree of main left subtree**, if the tree become unbalanced then LR rotation rule as follows:
- **Right child of left child of old root** is becomes **NEW ROOT,** and old root becomes right child of new root.
- New left child of new root is becomes left child of old root.
- **Left Child of Right child of left child of old root** (i.e new root's old left child) is becomes **new right child** of old left of left child of old root (i.e which is now left child of new root)
- **Right Child of Right child of left child of old root** (i.e new root's old right child) is becomes **new left child** of old root (i.e new left child of right child of new root)

LR Rotation

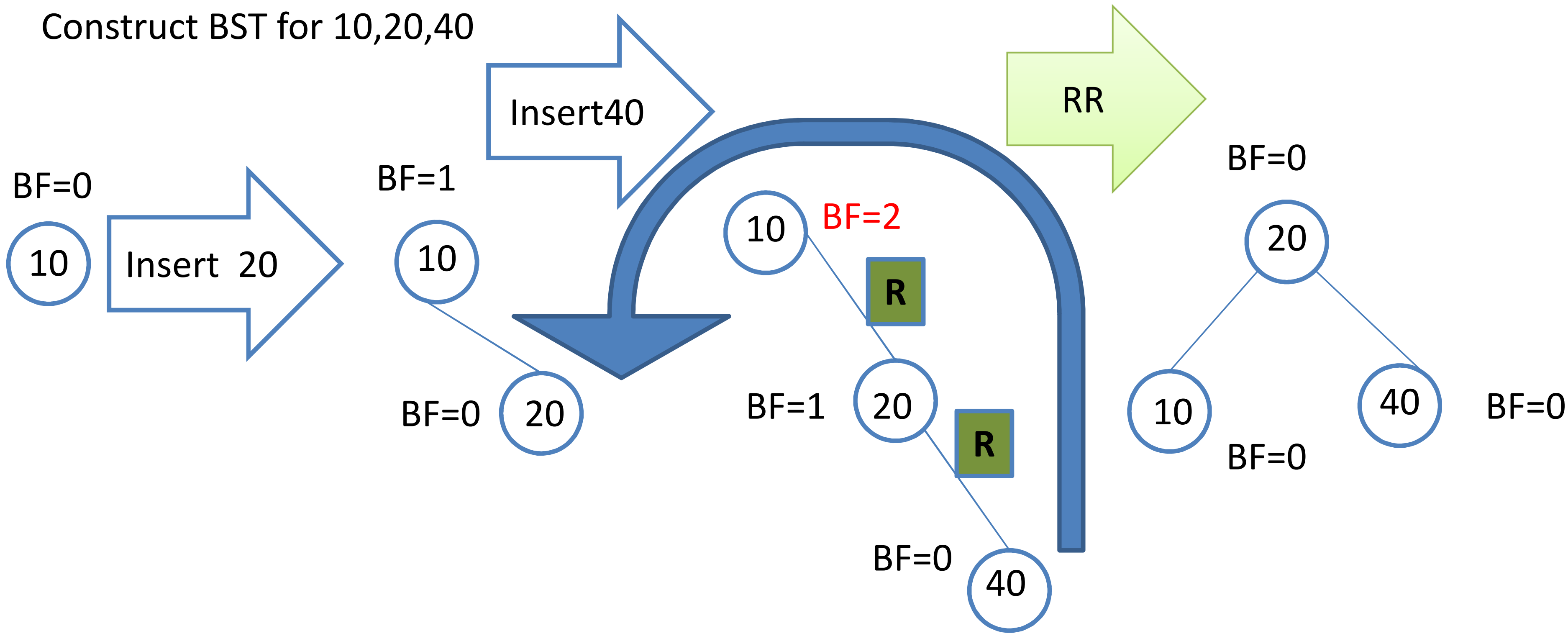


RR rotation

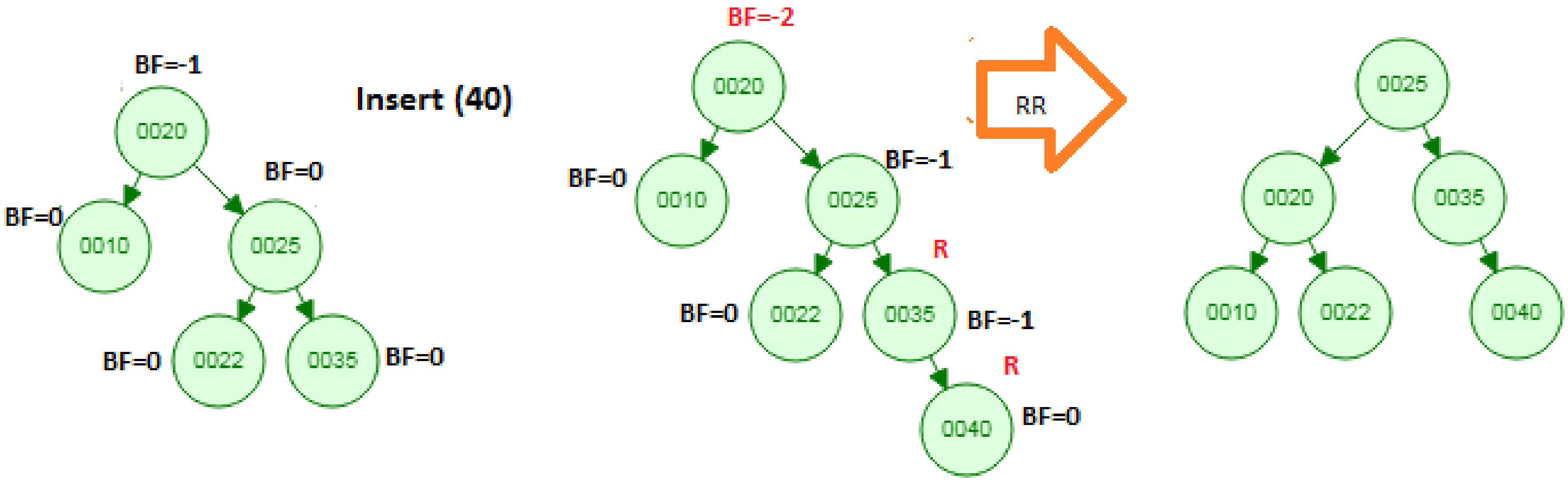
- After inserting a new node as a **right child in right subtree**, if the tree become unbalanced then the nodes in the tree must be rotate using RR rotation rule as follows:
- **Right child** of old root is becomes **NEW ROOT.**
- **Old root node** becomes **left child** of new root
- Right child of new root is not changed
- **Left child of right child of old root** becomes right child of old root (i.e. right child of left child of new root)

RR rotation example

Construct BST for 10,20,40



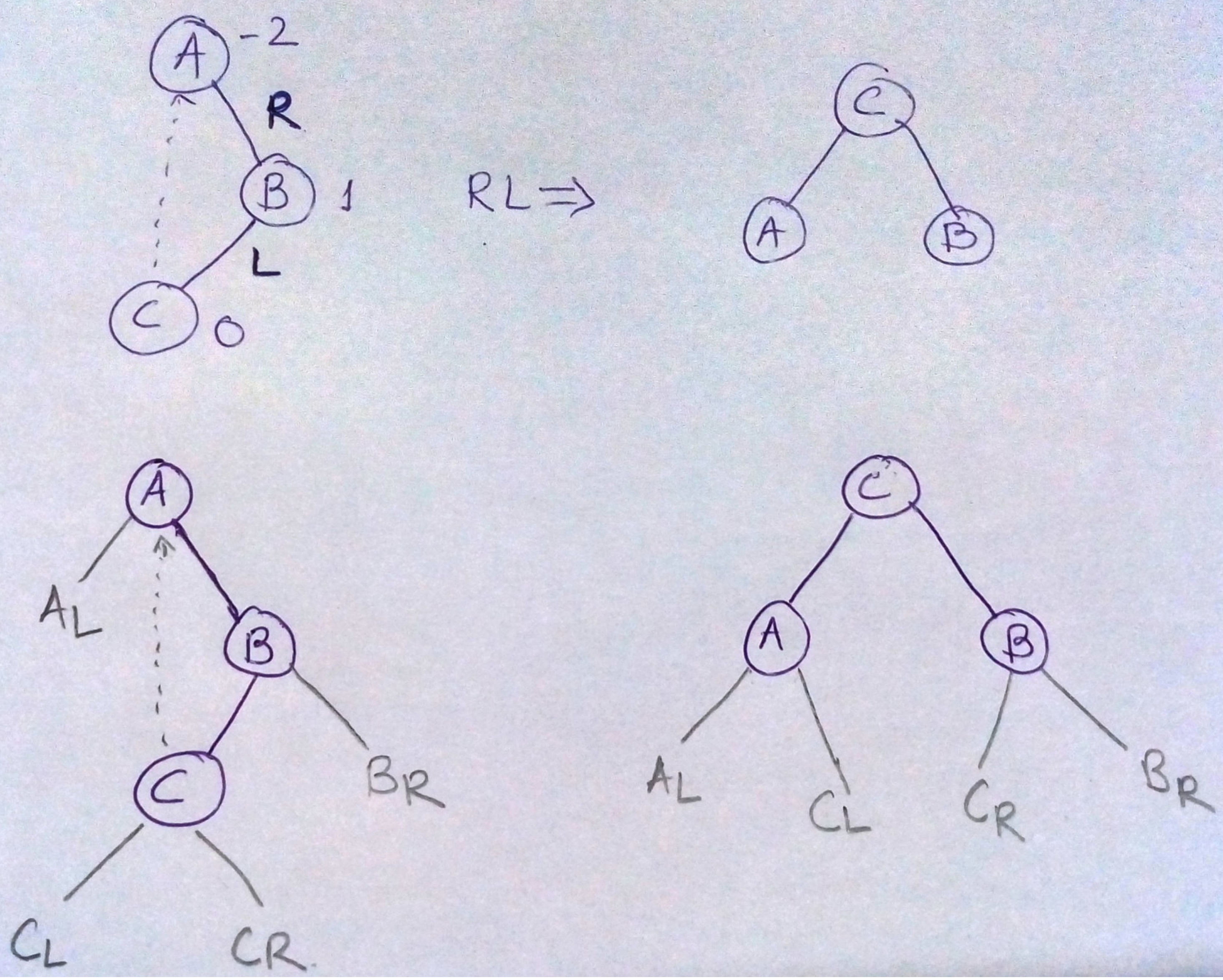
Another example



RL rotation

- If a new node is inserted as left or right in **left subtree of main right subtree**, if the tree become unbalanced then RL rotation rule as follows:
- **Left child of right child of old root** is becomes **NEW ROOT,** and old root becomes left child of new root.
- **Right child of left child of right child of old root** (i.e new root's old right child) is becomes **new left child** of old roots right child (i.e new left child of right child of new root)
- **Left child of left child of right child of old root** (i.e new root's old left child) is becomes **new right child** of old root.

R-L Rotation.



Construct AVL tree for the following

- 40,20,10,50,90,30,60,70,95
- 150, 155, 160, 115, 110, 140, 120, 145, 130, 147, 170, 180
- 69,80,73,40,33,70,1,86
- Nilu, Pranita, Princes, Raju, Soni, John, Akshay, Pavan, Oddy, Umesh.
- Input, Joystick, USB, Rom, Port, Ram, Windows, X-windows, Audio, Cache

Also write preorder, inorder and postorder traversal of tree