Balanced Binary Search Trees

Course on C-Programming & Data Structures: GATE - 2024 & 2025

Vishvadeep Gothi • Lesson 53 • Feb 1, 2023

Data Structure
Tree 8
Balanced & AVL Tree

By: Vishvadeep Gothi

Searching in Tree

BST ⇒ searching ⇒ O(h) → worst case ⇒ O(n)

---

searching in a BT ⇒ using traversal (preorder is best)

⇓

O(n)

```
struct BTnode * search_In_BT (struct BTnode *t, int key)
{
  if (t)
  {
    if (t->data == key)
      return t;
    Search_In_BT (t->Lc, key);
    Search_In_BT (t->Rc, key);
  }
}
```

Balanced Tree

A tree which has height always $\cong \log_2 n$ for n nodes.

=> A tree in which for each node the balance factor is in between -1 to 1.

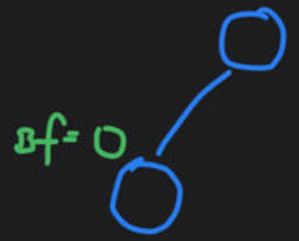
$$B.F. = L_H - L_R$$

L_H = height of left subtree

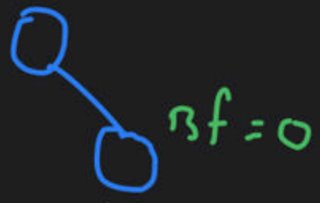
L_R = — " — Right — " —

B.f. = (-1) - (-1) = 0

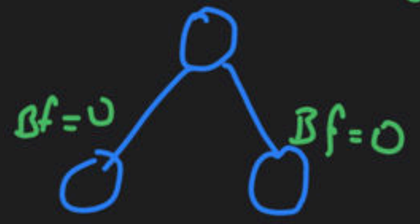
Bf = 0 - (-1) = 1



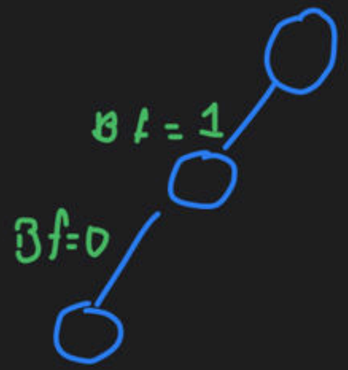
Bf = -1 - 0 = -1



Bf = 0 - 0 = 0

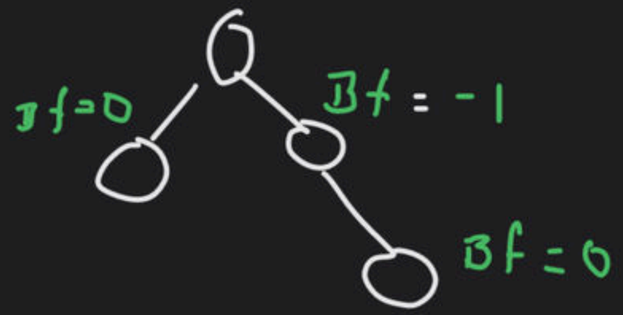


Bf = 1 - (-1) = 2



not balanced

Bf = -1



Balanced Tree :-

$h \leq \log_2 n$

AVL Tree

Self balancing BST

↳ $H \cong \log n$ always

Searching, insertⁿ, deletⁿ $\Rightarrow \log_2 n$

$\Rightarrow$ Insertⁿ or deletⁿ in BST

↳ AVL tree balances itself by applying some methods.

AVL Tree

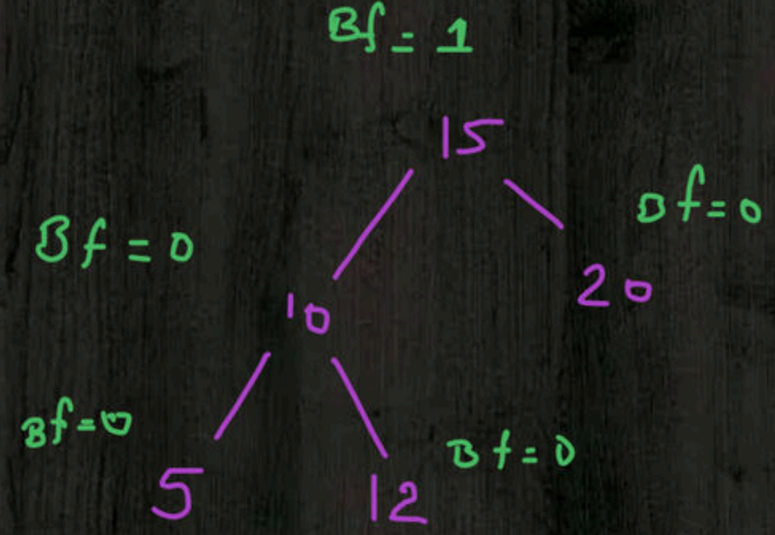
for each node

$$Balance\ factor = h_L - h_R = \begin{cases} 1 \\ 0 \\ -1 \end{cases}$$

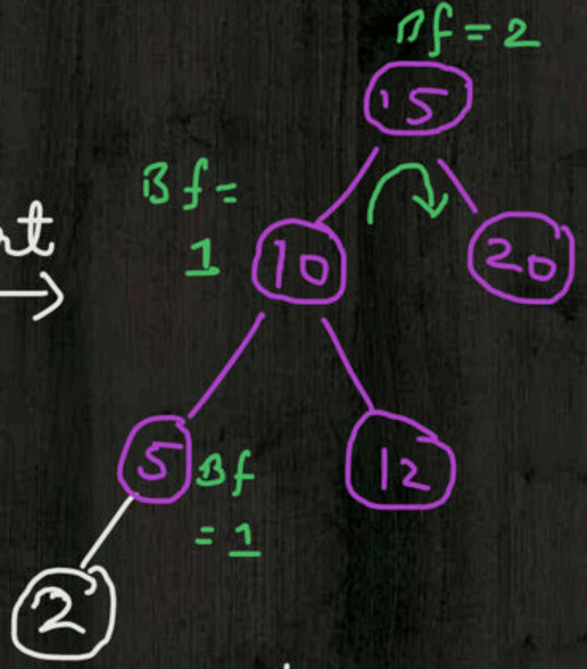
Types of Imbalances

1. LL
2. RR
3. LR
4. RL

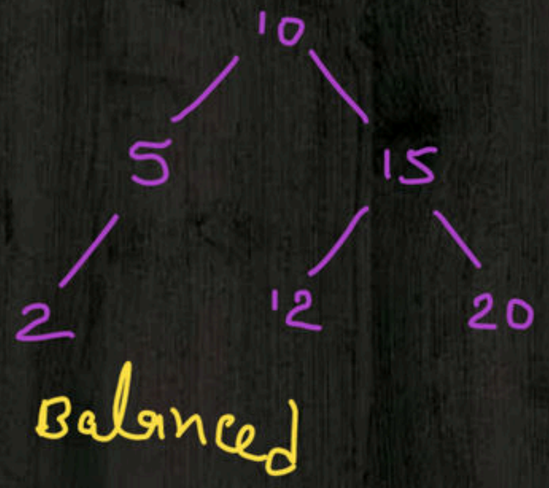
LL-Imbalance



Insert 2



LL rotation at 15



Balanced

Imbalance at node 15 because of insertion in 15's left and it's left subtree

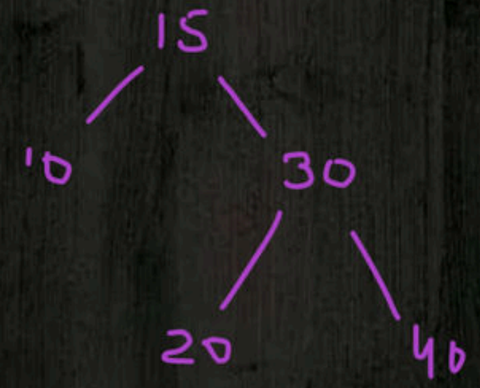
⇓

LL imbalance

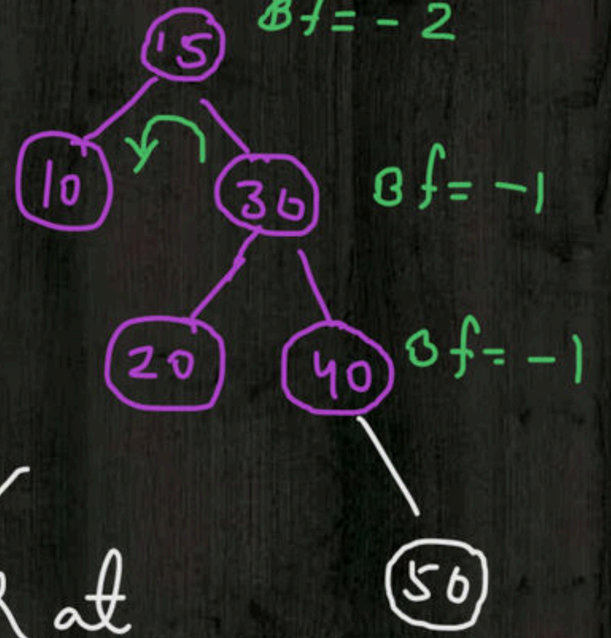
⇓

Soln => LL rotatn ⇒ single Right rotatn

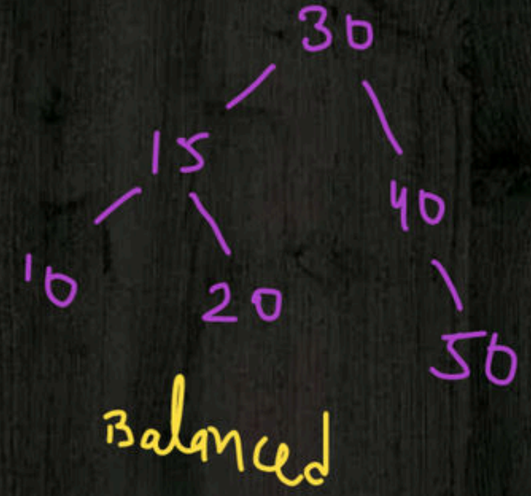
RR-Imbalance



Insert 50



RR at 15



Balanced

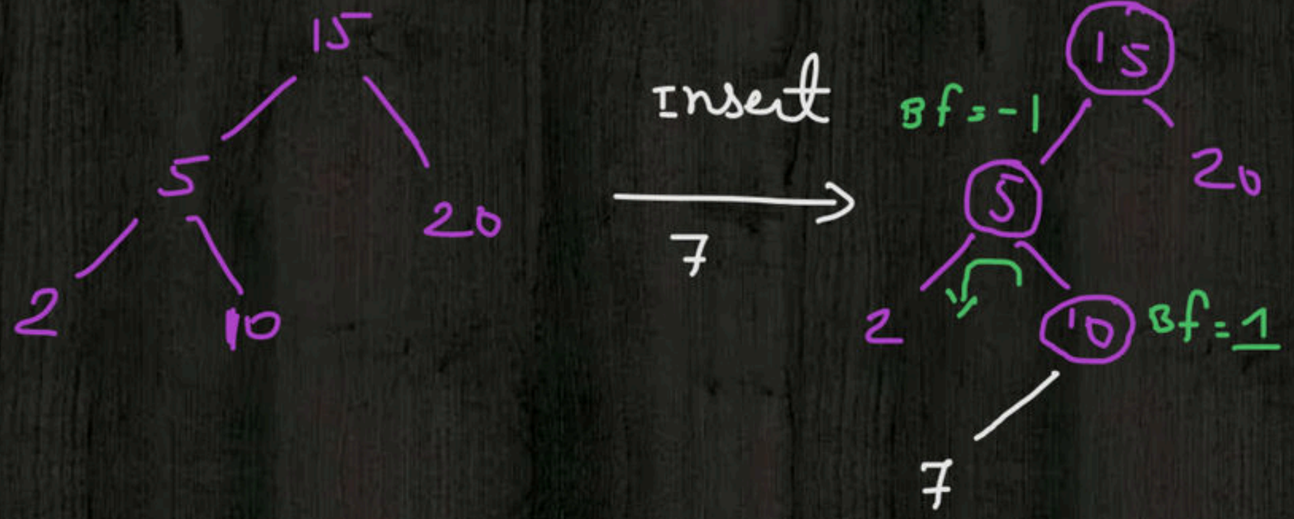
Imbalance at 15, because of insertion in 15's Right and it's Right subtree

⇓

RR imbalance

Sol^{n} ⇒ RR rotation or single left rotation

LR-Imbalance



Imbalance at 15, because of insertⁿ in 15's Left and it's Right subtree

⇓

LR imbalance

⇓

Solⁿ = LR rotatⁿ

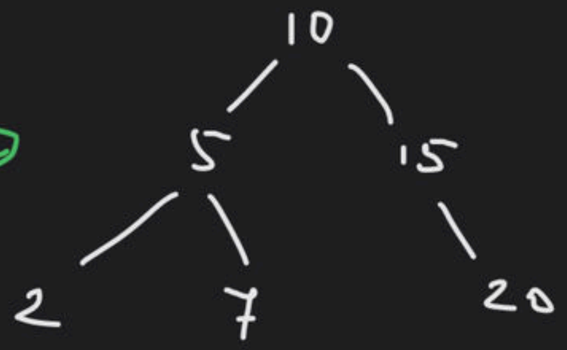
LR rotation :-

→ RR at Left child of imbalanced node

→ LL at imbalanced node

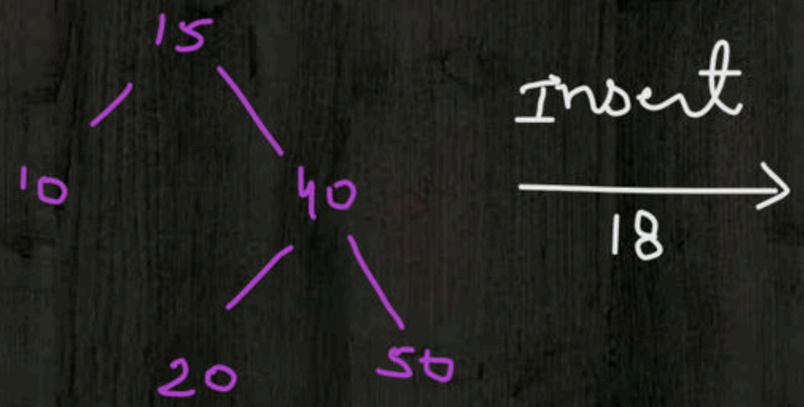
RR at 5

LL at 15



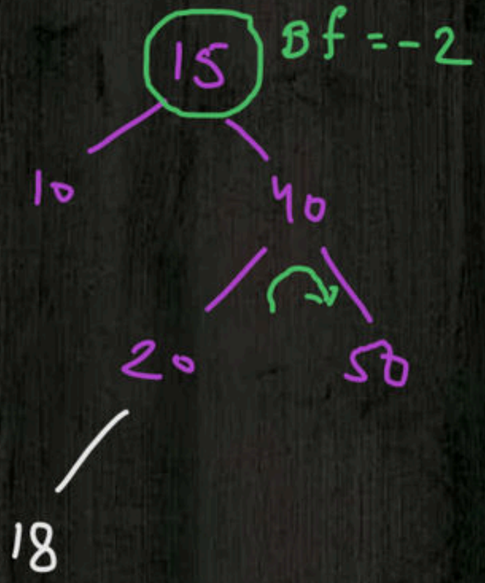
Balanced

RL-Imbalance



Insert

18



Imbalance at 15 because of insertion in 15's Right and it's Left subtree

⇓

RL imbalance at 15

⇓

RL rotatⁿ

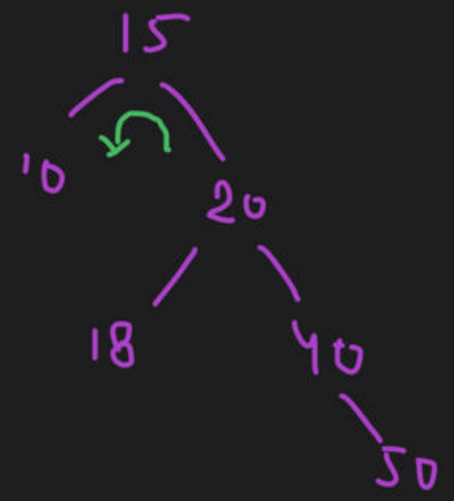
RL rotation :-

⇒ LL rotatⁿ at right child of imbalanced node

⇒ RR rotatⁿ at imbalanced node

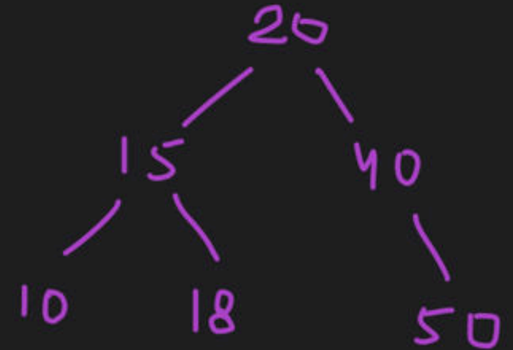
LL at

40



RR at

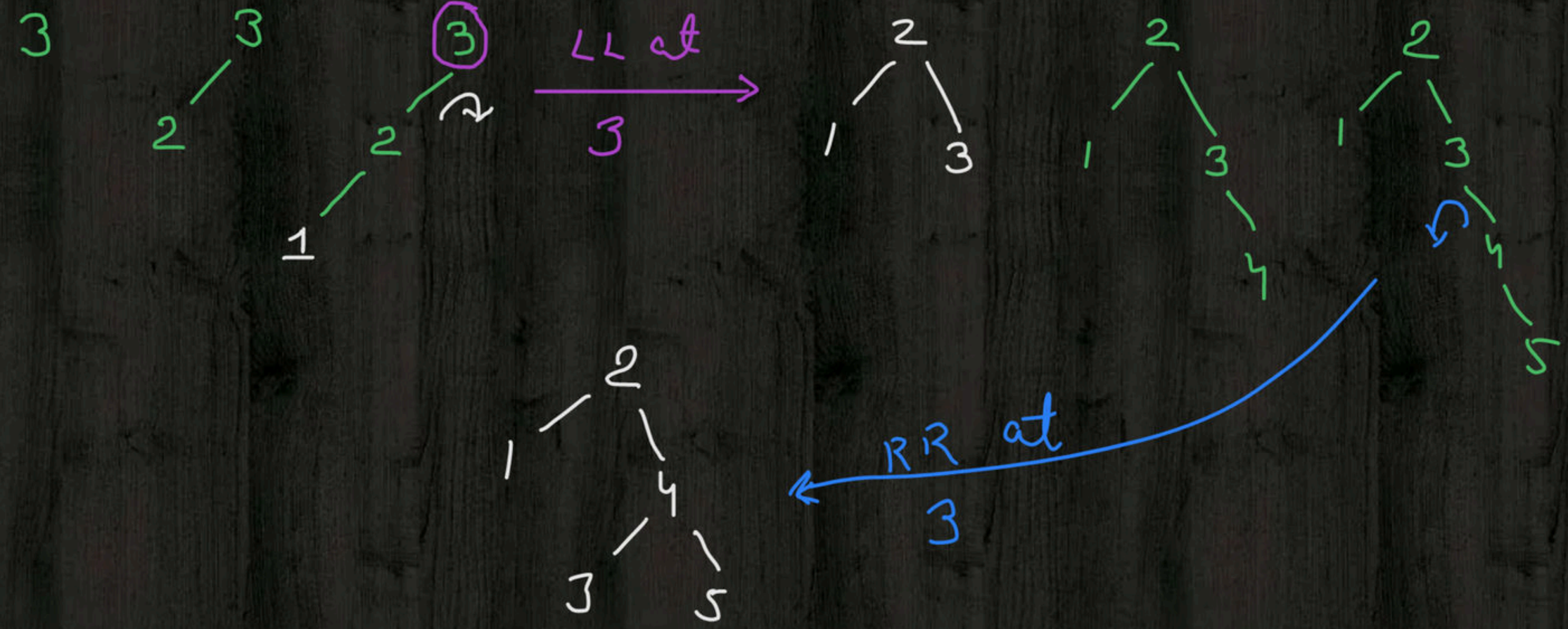
15

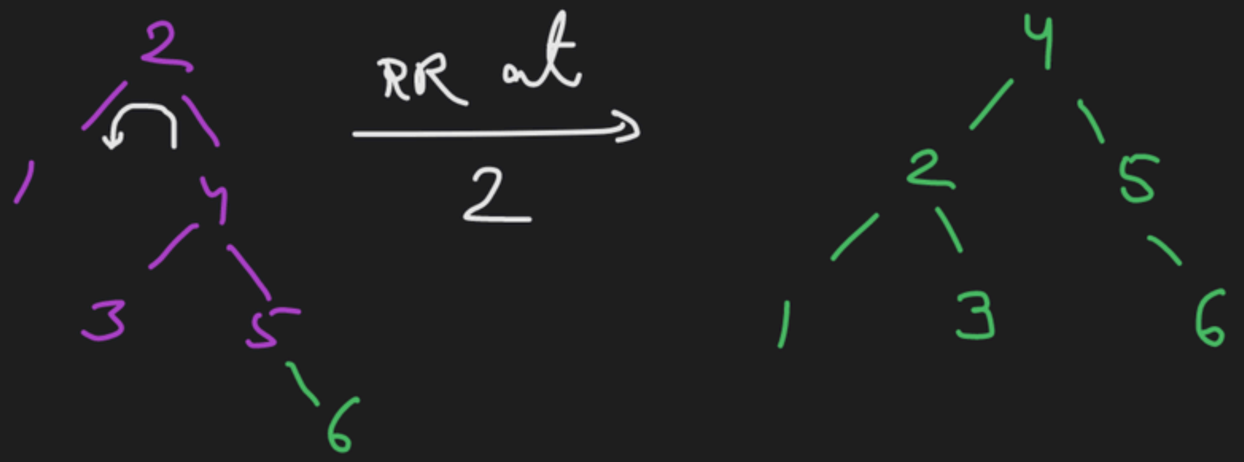


Balanced

AVL Tree Insertion 1

Create AVL tree using keys: ~~3~~, ~~2~~, ~~1~~, ~~4~~, ~~5~~, 6




2
1
4
3
5
6
RR at
2
4
2
5
1
3
6

AVL Tree Insertion 2

Create AVL tree using keys: 25, 27, 40, 8, 5, 13, 29, 17, 15, 11, 1, 2, 6

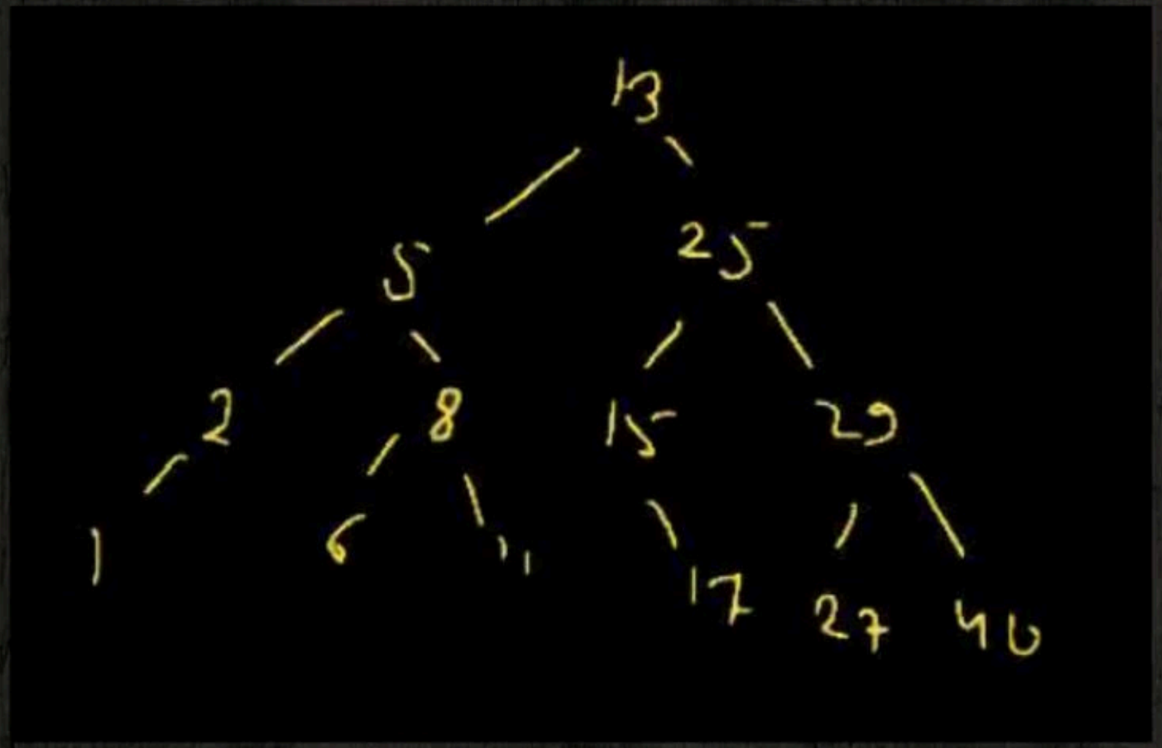
AVL Tree Deletion

6 cases for deletion:

1. R_0
2. R_1
3. R_{-1}
4. L_0
5. L_1
6. L_{-1}

AVL Tree Deletion

Delete keys from below AVL tree: 1, 2, 11, 17, 5, 8, 13, 6, 15



Question

The minimum number of nodes in ALV tree with height H is _____?

Question

The maximum height of an ALV tree with 7 nodes is _____?

DPP

Question 1

Create AVL tree using keys: A, V, L, T, R, E, I, S, O, K

Question 2

Insert the following sequence of elements into an AVL tree, starting with an empty tree: 10, 20, 15, 25, 30, 16, 18, 19

Question 3

Insert the following sequence of elements into an AVL tree, starting with an empty tree: 15, 20, 24, 10, 13, 7, 30, 36, 25

Question 4

Insert the following sequence of elements into an AVL tree, starting with an empty tree: 1,2,3,4,8,7,6,5,11,10,12

Question 5

Insert the following sequence of elements into an AVL tree, starting with an empty tree: 10, 20, 15, 25, 30, 16, 18, 19

Then delete the keys: 30, 25, 16, 19, 20

Happy Learning