UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

1

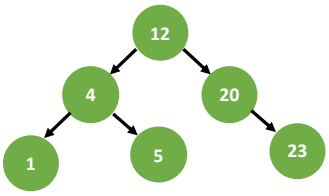
Quick Recap

- Trees
  - Root, Parent, Child, leaf node, Depth/level, etc
- Binary Search Trees
  - Each node can have at most two child
  - Parent is greater than the left and smaller than the right
- Operations –
  - Insert
  - Search – Max, Min
  - Delete – Leaf node, Node having one child, Node having two children
- Traversal – Pre-order, In-order, Post-order

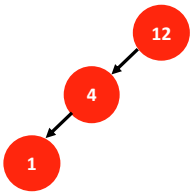
2

Quick Recap

- Balanced and Imbalanced Trees



BALANCED TREE



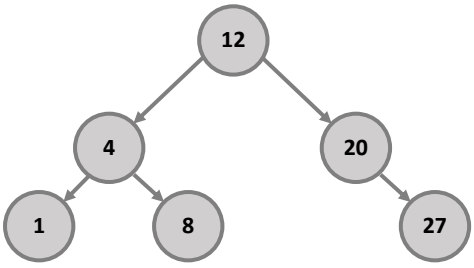
IMBALANCED TREE

Binary Search Trees

- To determine whether the tree is balanced or not, we have to measure its height
- Height of a tree (or a node) is the longest path from the root (or from the node) to a leaf node

height = max (left child's height , right child's height) + 1

The height of a NULL node is -1
➔ leaf nodes have height 0.

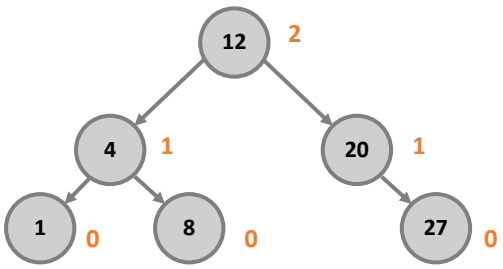


Binary Search Trees

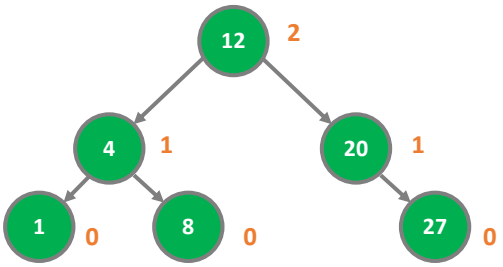
- To determine whether the tree is balanced or not, we have to measure its height
- Height of a tree (or a node) is the longest path from the root (or from the node) to a leaf node

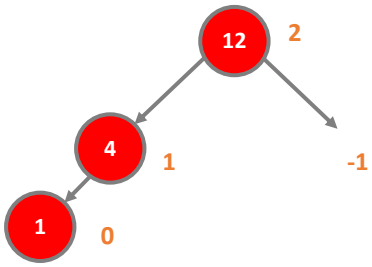
height = max (left child's height , right child's height) + 1

The height of a NULL node is -1
➔ leaf nodes have height 0.



Binary Search Trees

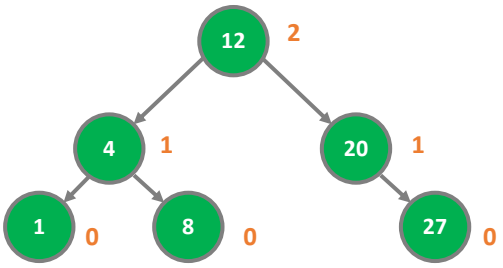


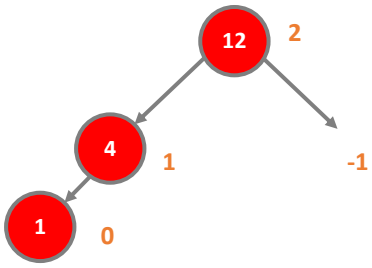


- Height of both trees is 2!

How can we determine programmatically which one of these trees is imbalanced?

Binary Search Trees



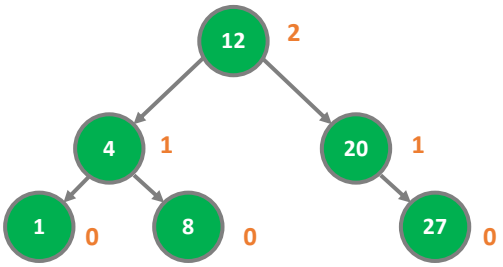


- Compare the heights of left and right subtrees by calculating the balance factor below:

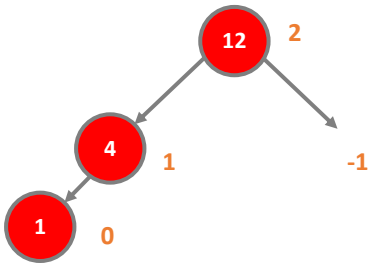
Balance factor: $\mathbf{h_{left} - h_{right}}$

- The tree heights on both sides can differ at most by 1

Binary Search Trees

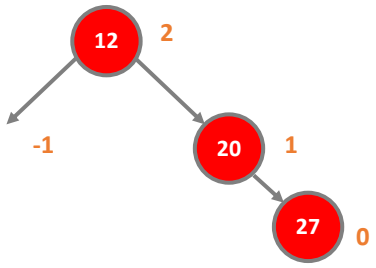


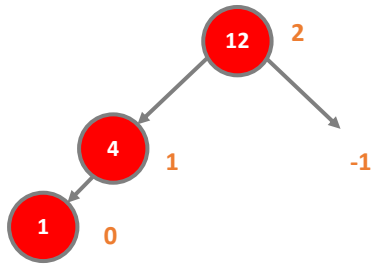
Balance factor: **1 – 1 = 0**



Balance factor: **1 – (–1) = 2**

Binary Search Trees





Balance factor: **-1 – 1 = -2**

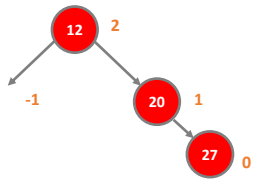
Right-heavy case
→ Rotate **left** to balance

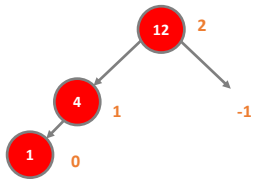
Balance factor: **1 – (–1) = 2**

Left-heavy case
→ Rotate **right** to balance

Binary Search Trees

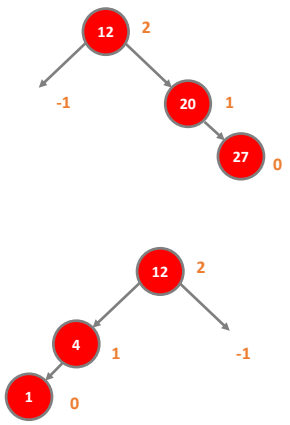
- Keep tracking the height of each node in BST
- Calculate the balance factors for the nodes
- Make rotations if necessary to balance search trees
- -ve balance factor → right-heave tree → rotate left
- +ve balance factor → left-heave tree → rotate right





Binary Search Trees | AVL Trees

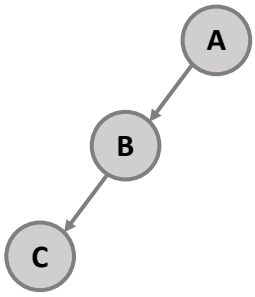
- Keep tracking the height of each node in BST
- Calculate the balance factors for the nodes
- Make rotations if necessary to balance search trees
- -ve balance factor → right-heave tree → rotate left
- +ve balance factor → left-heave tree → rotate right
- Balanced data structure invented by Adelson-Velsky and Landis
- Has guaranteed O(logN) running time for all operations



AVL Trees | Rotations

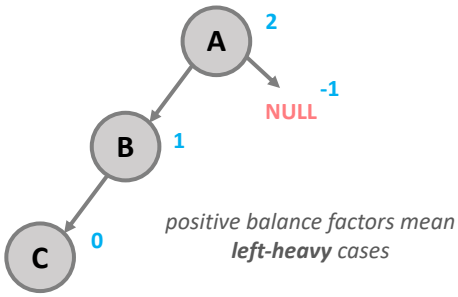
- Right rotation on root node (with value 5)
- Left rotation on root node (with value 3)
- The BST properties remain same
- The in-order traversal remains same after rotations

AVL Trees — Rotations — Case 1



13

AVL Trees — Rotations — Case 1



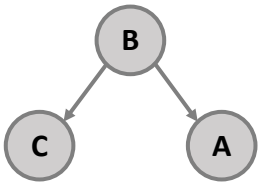
positive balance factors mean ***left-heavy*** *cases*

14

AVL Trees

Rotations

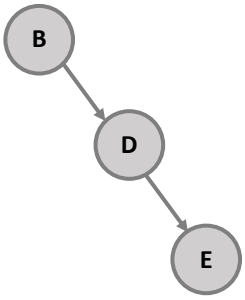
Case 1



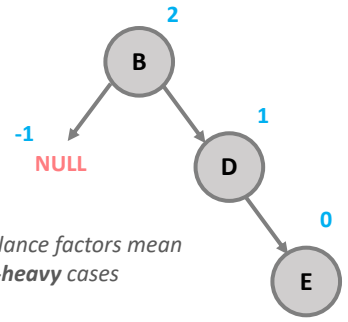
AVL Trees

Rotations

Case 2

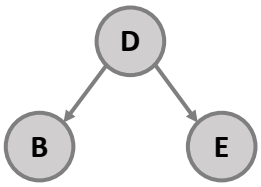


AVL Trees — Rotations — Case 2

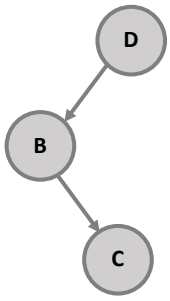


negative balance factors mean ***right-heavy*** *cases*

AVL Trees — Rotations — Case 2

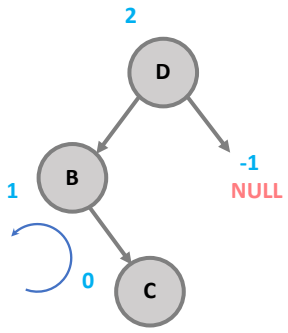


AVL Trees — Rotations — Case 3



AVL Trees — Rotations — Case 3

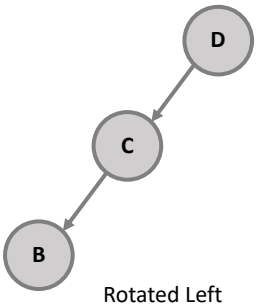
Balance factor of root → 2
→ left-heavy
- Determine BF of left child
- if –ve
  - → left-right heavy
  - → Rotate parent left
  - → Rotate grand-parent right
- if +ve
  - → left-left heavy
  - → Rotate grand-parent right



AVL Trees

Rotations

Case 3

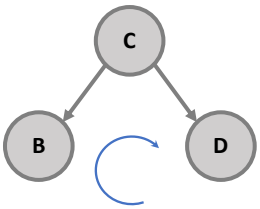


Rotated Left

AVL Trees

Rotations

Case 3

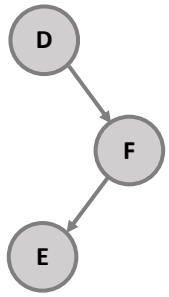


Rotated right

AVL Trees

Rotations

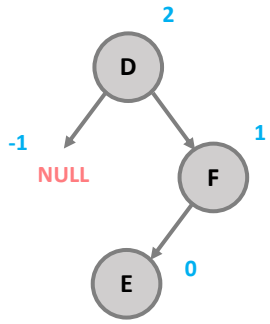
Case 4



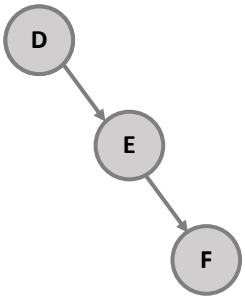
AVL Trees

Rotations

Case 4

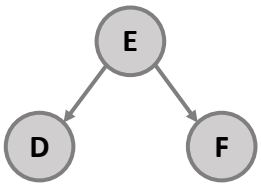


AVL Trees | Rotations
Case 4



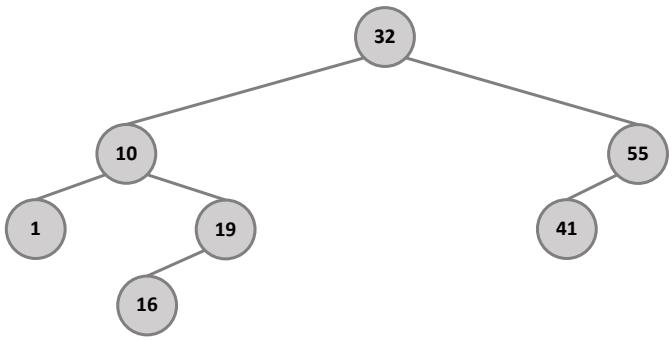
25

AVL Trees | Rotations
Case 4



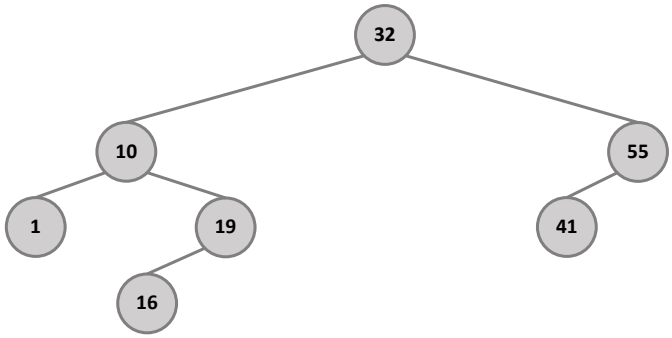
26

AVL Trees Insertion



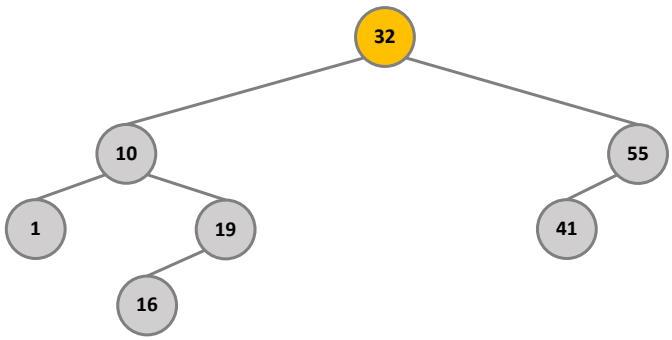
AVL Trees Insertion

INSERT(12)



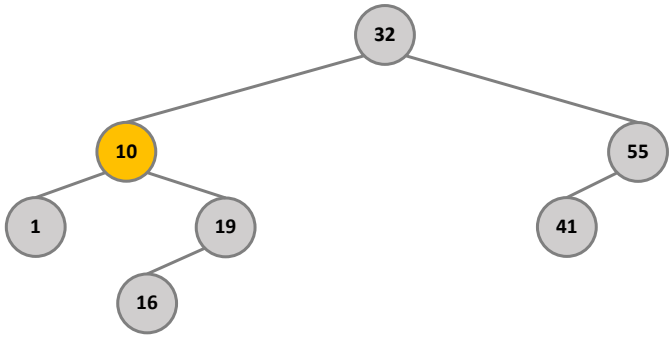
AVL Trees Insertion

INSERT(12)



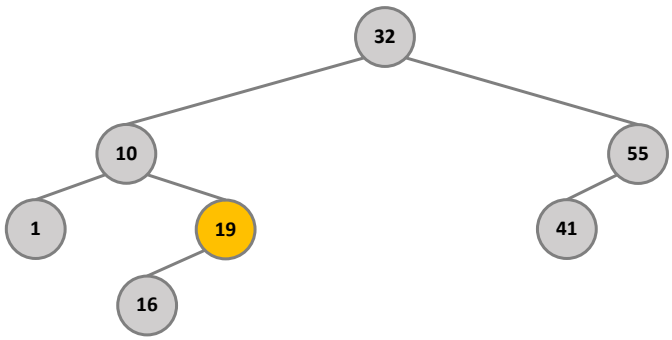
AVL Trees Insertion

INSERT(12)



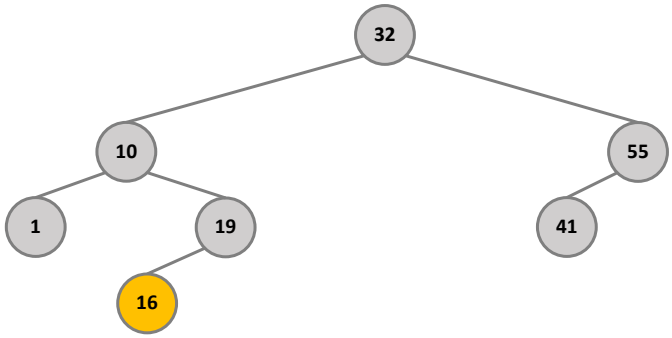
AVL Trees Insertion

INSERT(12)



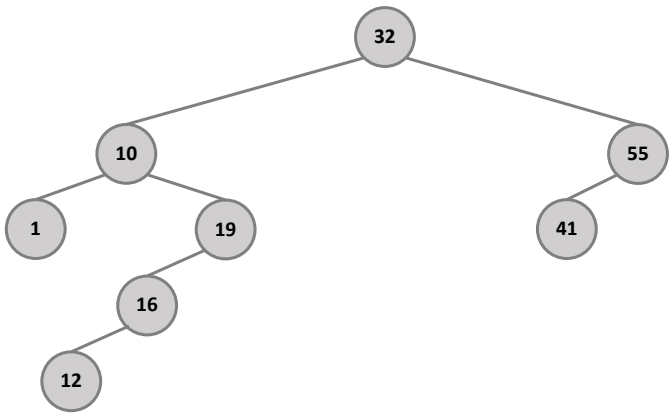
AVL Trees Insertion

INSERT(12)



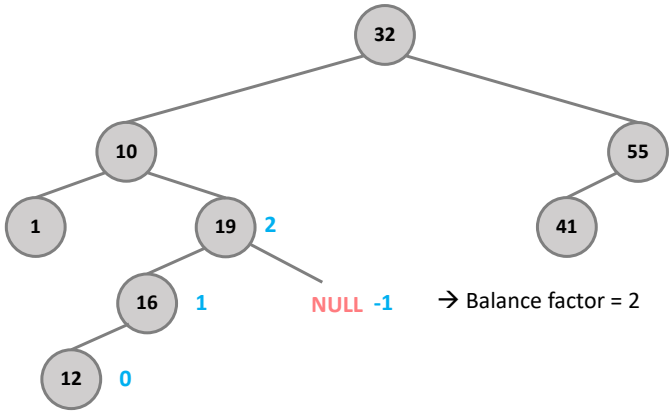
AVL Trees Insertion

INSERT(12)



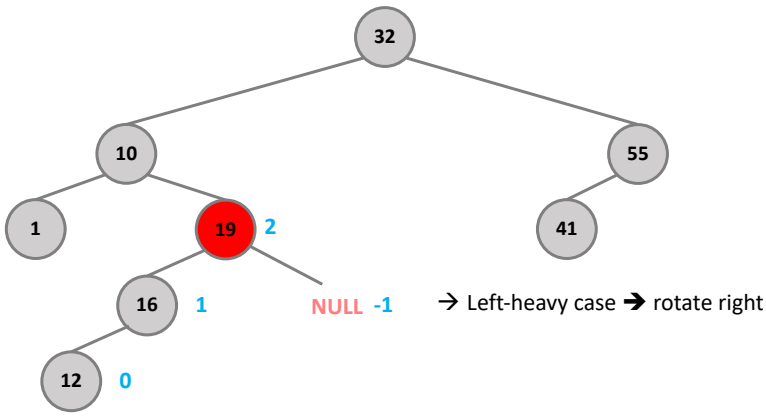
AVL Trees Insertion

INSERT(12)



AVL Trees Insertion

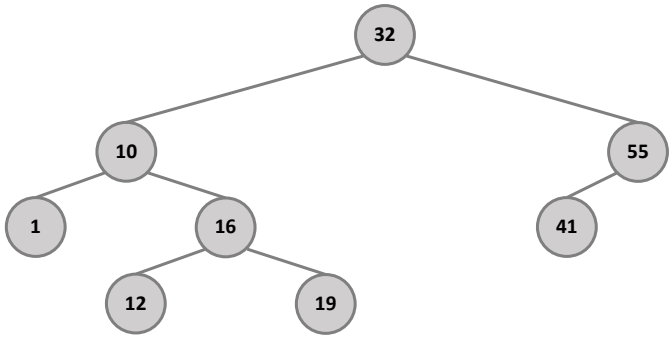
INSERT(12)



→ Left-heavy case ➔ rotate right

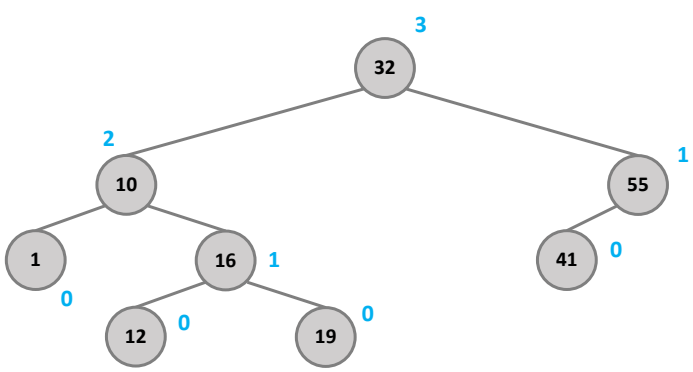
AVL Trees Insertion

INSERT(12)



AVL Trees Insertion

INSERT(12)



Balance factor for every node differs by 1

AVL Trees

- Implementation will be a part of assignment 3
  - Same as BST implementation except:
    - Height and Balance factor must be calculated for every node