Tree and AVL Tree

Computing Lab
CS 69201
Department of Computer Science and Engineering
IIT Kharagpur

Previously

- ➢ We have looked at Sorting, Greedy and DP algorithms.
- ➢ We will start a new topic -

TREE

Topics

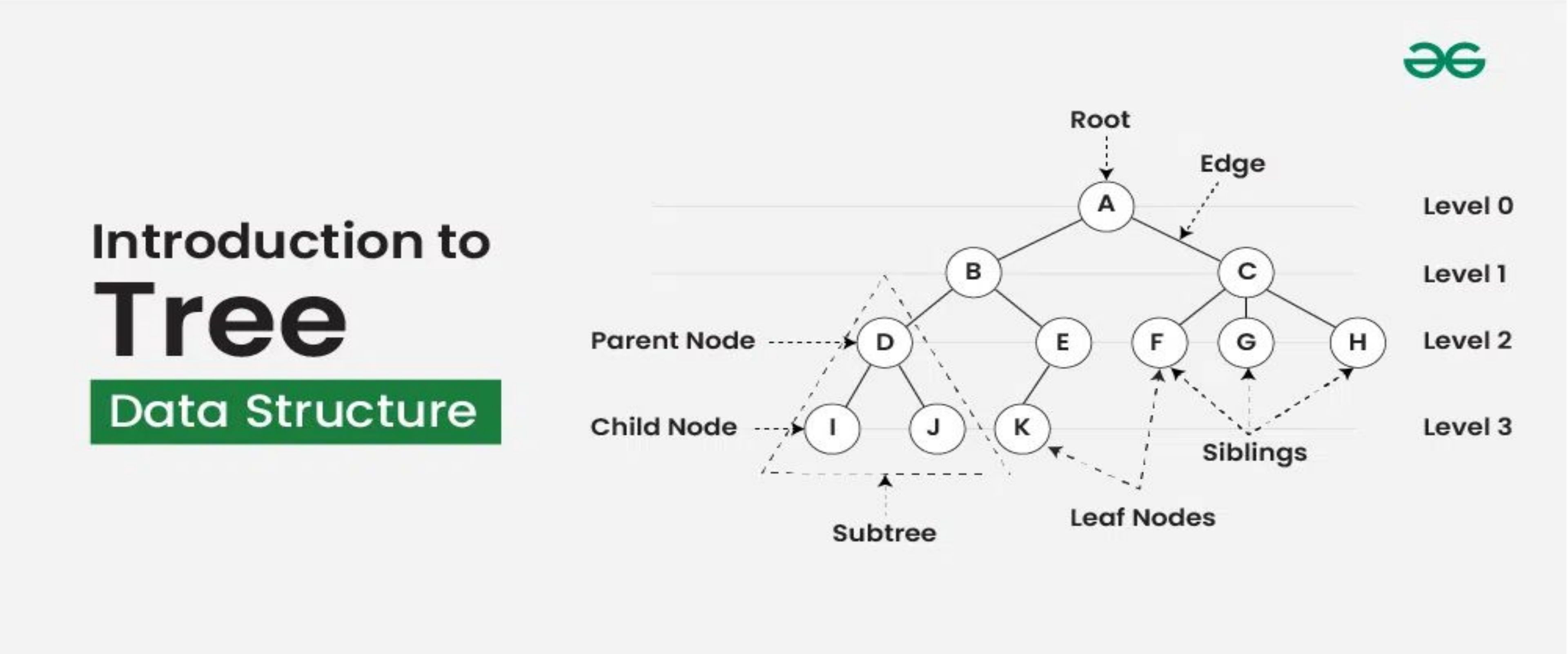
TREE -

- Concept
- Insert
- Search
- Delete
- Height
- Traversal

AVL Tree -

- Concept
- Balance factor
- Rolations
- Insert / delete

TREE



Introduction to Tree

Data Structure

TREE

- A tree is also one of the data structures that represent hierarchical data. Some properties-
  - Recursive data structure.
  - Non- linear Data structure.
  - No loop.
  - One-way (root to any node) traversal.
  - N nodes means (N-1) edges

TREE

➢ Some common keywords in Tree data structure-

- Parent Node
- Child Node
- Root Node
- Leaf Node or External Node
- Ancestor of a Node
- Descendant
- Sibling
- Level of a node
- Internal node
- Neighbour of a Node
- Subtree

Binary Tree

- Each node has up to two children, the left child node and the right child node. This structure is the foundation for more complex tree types like Binary Search Trees and AVL Trees.-
    - Searching: O(N).
    - Insertion: O(N).
    - Deletion: O(N).

Binary Search Trees (BSTs)

- A type of Binary Tree where for each node, the left child node has a lower value, and the right child node has a higher value..-
    - We will work with this -
    - Searching: O(h).
    - Insertion: O(h).
    - Deletion: O(h).
    - **h = height of the tree**

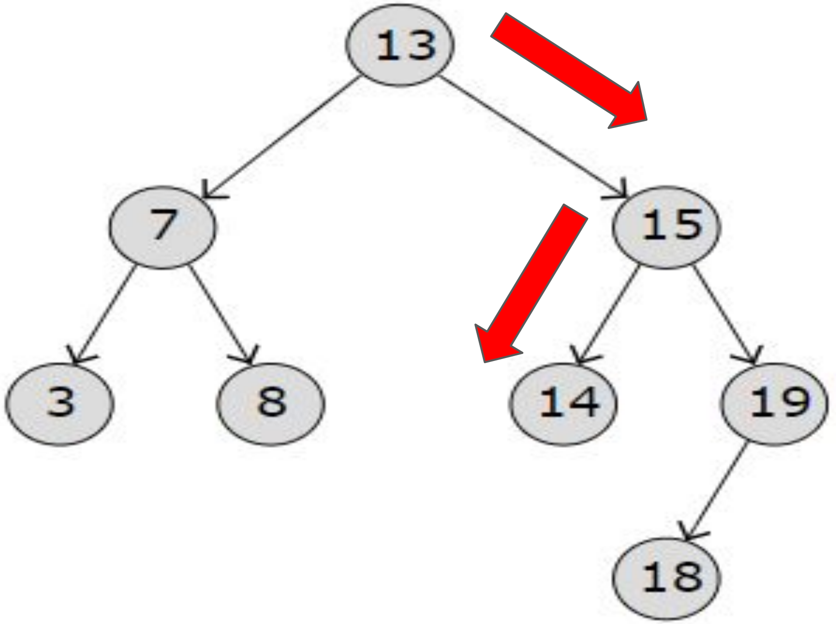
Searching

Giving a key, we need to search within the tree-

- Recursive Search
- If root is the key , we found it.
- If key less then the root, search the left subtree.
- If key more then the root, search the right subtree.
- If we reach NULL node, tree does not have the key.

Searching

Giving a key, we need to search ot within the tree, **Key - 14**



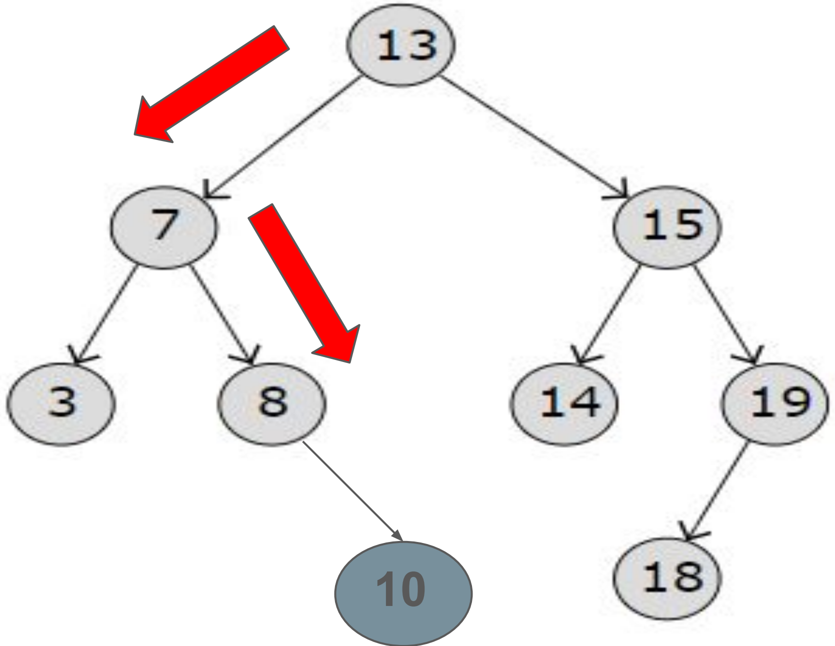
Inserting

Giving a key, we need to insert within the tree-

- Do Search
- If key is found, tree already has the node.
- If we reach NULL node, set this node value equal to key.

Inserting

Giving a key, we need to search ot within the tree, **Key - 10**



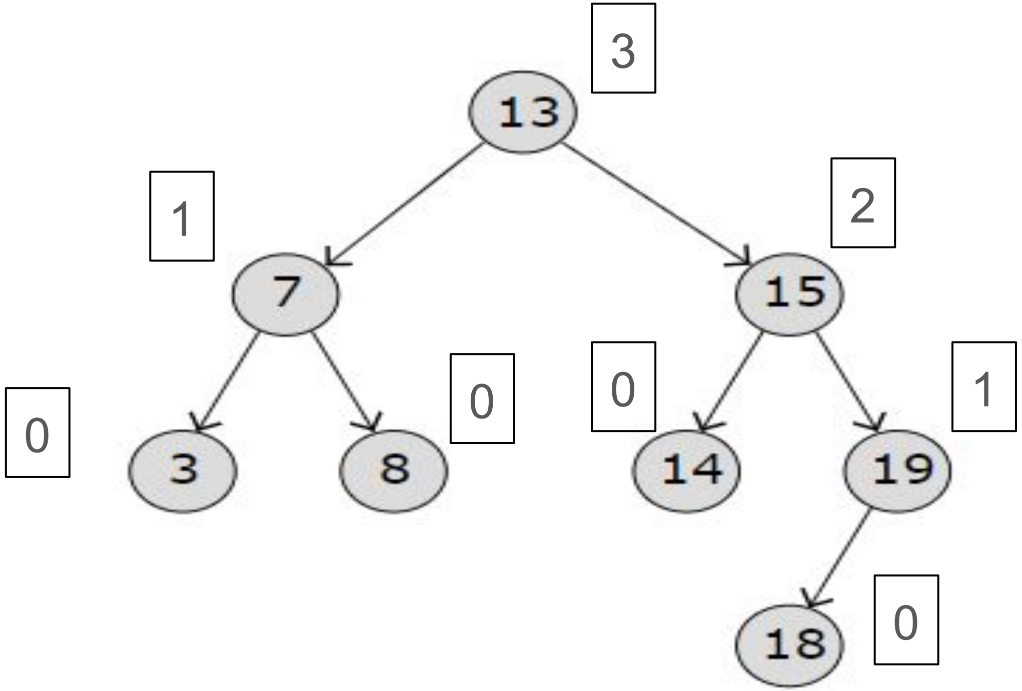
Height / Depth

We need to find height of the tree-

- The height of a tree is the length of the path from the root to the deepest node in the tree.
- A (rooted) tree with only a node (the root) has a height of zero.
- Height of leaf node is 0.
- Height of tree is height of the root.
- Height of a node is maximum of height of the children of the node and + 1.

Height / Depth

Giving a key, we need to search ot within the tree, **Height - 3**

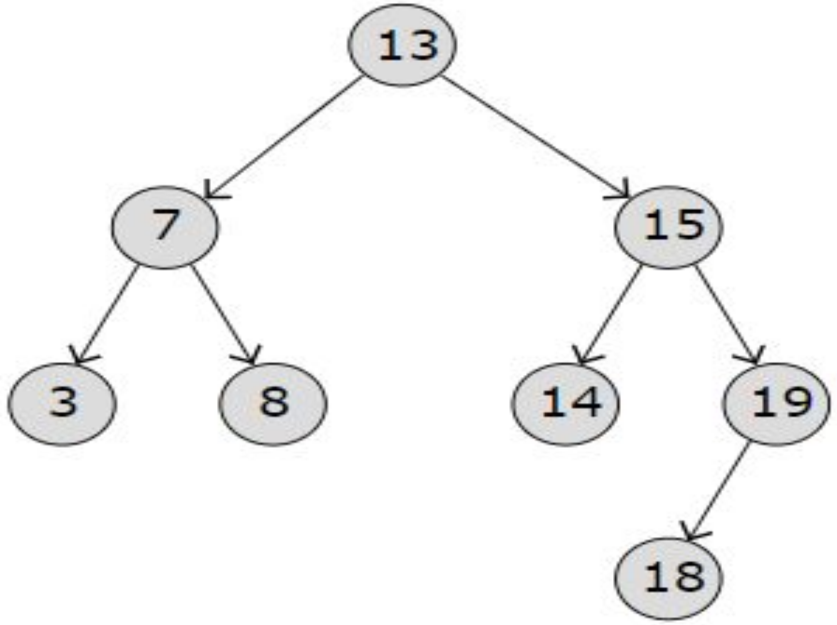


Inorder

Giving a tree, we need to find inorder list-

- Recursive inorder left
- Root
- Recursive inorder right

Inorder = 3 7 8 13 14 15 18 19
(inorder of BST is always sorted)

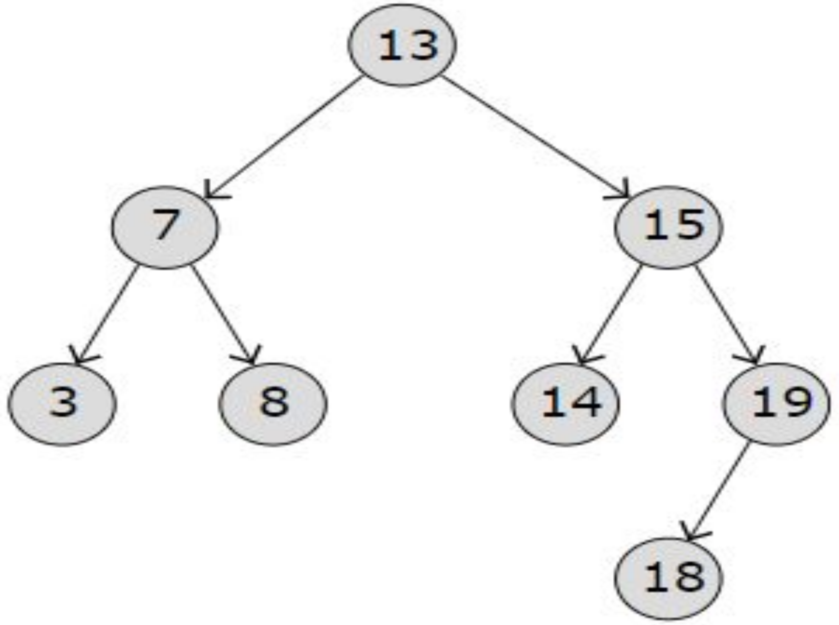


Preorder

Giving a tree, we need to find Preorder list-

- Root
- Recursive inorder left
- Recursive inorder right

Preorder = 13 7 3 8 15 14 19 18
(root is first in preorder)

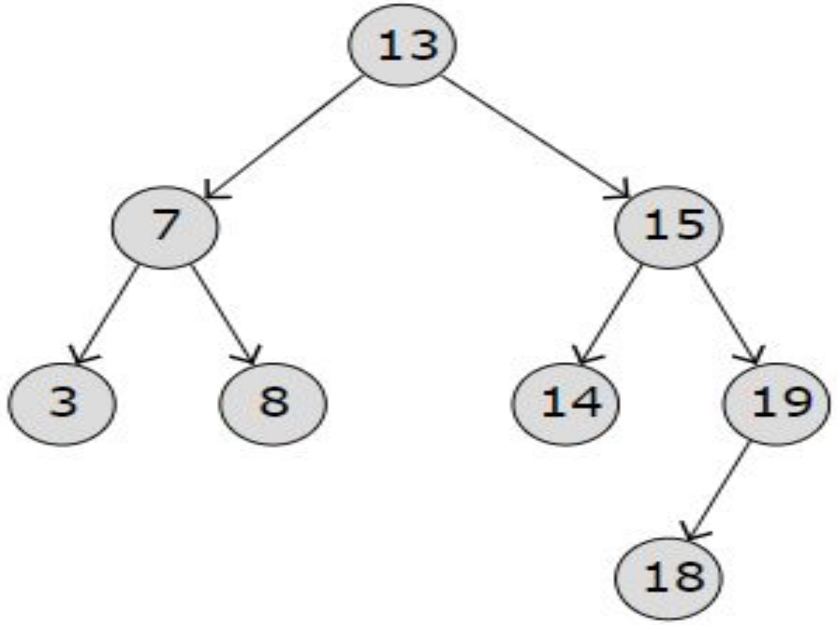


Postorder

Giving a tree, we need to find Postorder list-

- Recursive inorder left
- Recursive inorder right
- Root

Postorder = 3 8 7 14 18 19 15 13
(root is last in postorder)

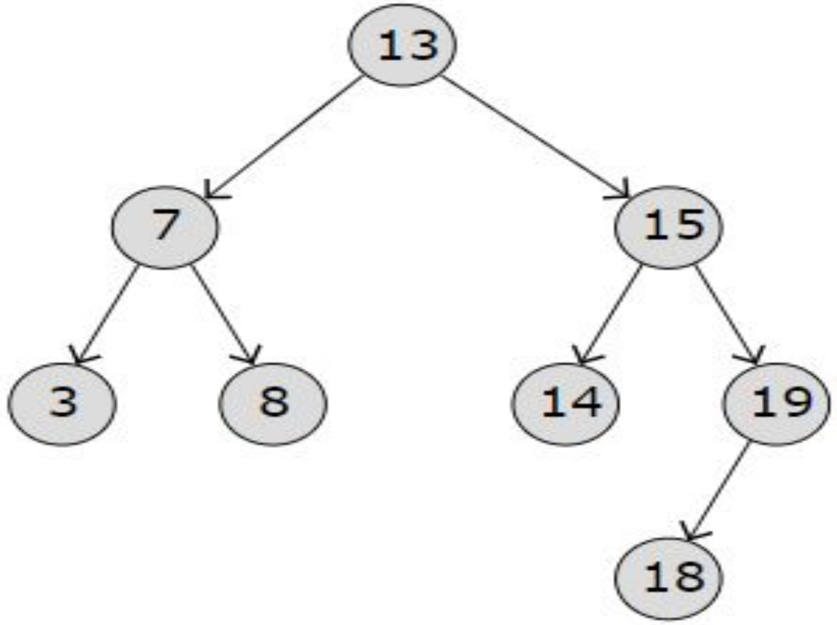


Level-order

Giving a tree, we need to find Level order list-

- Start with root
- Visit node level by level
- In a level, left to right

Level-order = 13 7 15 3 8 14 19 18



Deleting

Giving a key, we need to Delete within the tree-

- Do Search
- If key is not found, tree does not have the node to delete.
- If key is found, delete it.

Conditions -

- Node is leaf (no problem)
- Node has 1 child (replace child with the current node).
- Node has 2 child (?).

Deleting

Deleting a node with both children is not so simple. Here we have to delete the node is such a way, that the resulting tree follows the properties of a BST.

The trick is to find the inorder successor of the node. Copy contents of the inorder successor to the node, and delete the inorder successor [Recursive]. (inorder predecessor also works)

inorder successor = next node in Inorder list / smallest node in right subtree

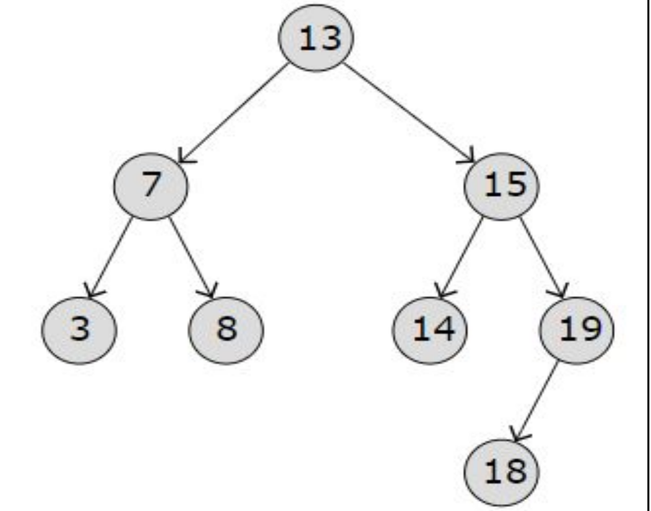
Deleting

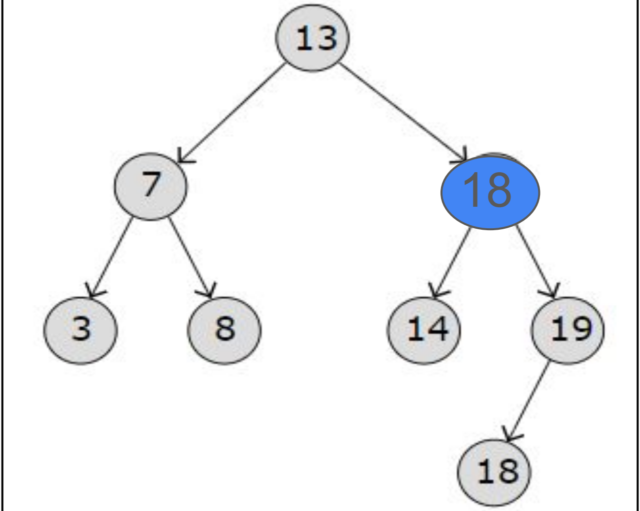
Giving a key, we need to delete the node within the tree, **Key - 15**

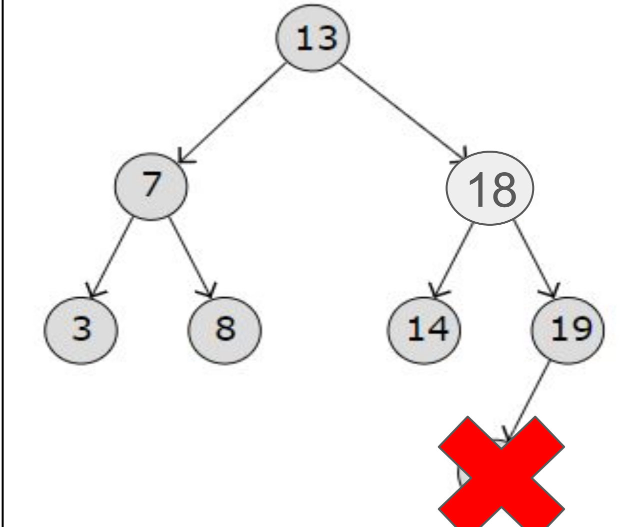
Delete key - 15
Inorder successor - 18

Node value - 15 → 18
Delete key - 18

Delete key - 18
18 is leaf node







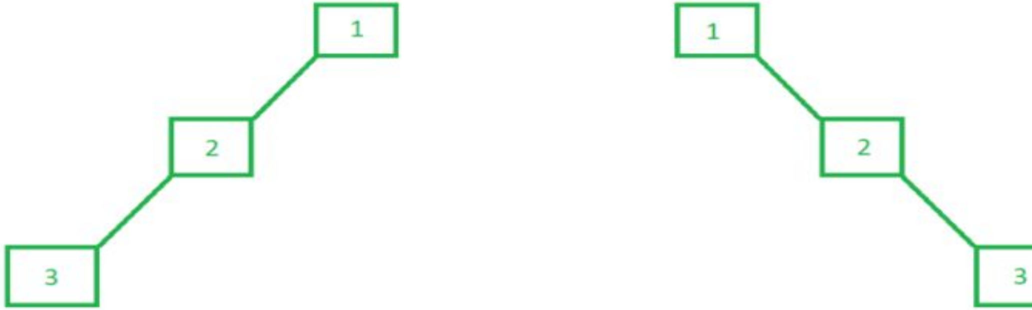
Question

Questions will be given-
Implement what we have discuss
Time - 2 hr.
Implement with menu driven format with functionality **(BST)**-

- Insert
- Search
- Delete
- Find_height
- 3 types of ordering
- Level order and zigzag order

Problem

- Complexity is O(h) . Worst case?
- N = h
- Time complexity becomes same as Binary Tree
- Solution ?
- Since each level has 2 nodes, we want to make height =
    - logN (base 2)
    -



AVL Tree

- A type of Binary Search Tree that self-balances so that for every node, the difference in height between the left and right subtrees is at most one. This balance is maintained through rotations when nodes are inserted or deleted.
- Balance of a node is determine by balance factor.

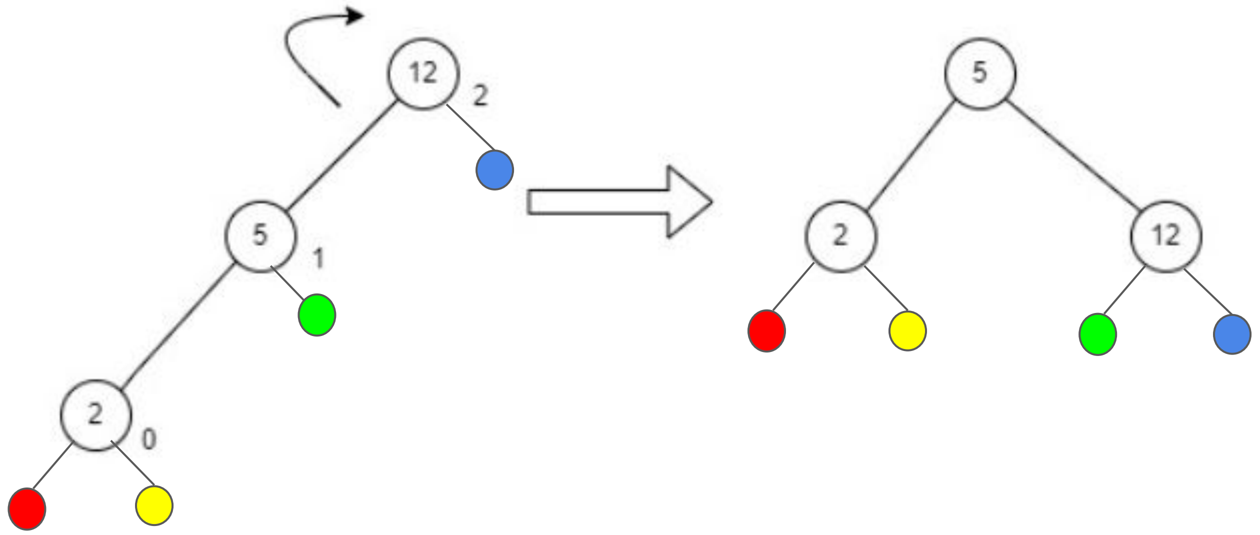
Balance Factor

- Balance Factor= Height of Left Subtree - Height of Right Subtree.
- Leaf node balance factor is 0.
- Valid balance factor that a AVL tree can have is { -1, 0, 1 }
- If there is invalid balance factor node , then make a rotation.

- **Note:** there can be multiple invalid node after an Insertion / deletion , but we will take the node which is closest to the point of Insertion / deletion.
- Rotation?
- 4 types of rotation

Type of Rotations

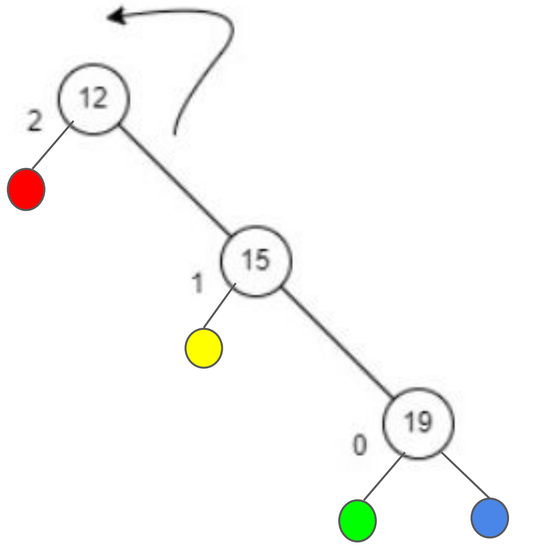
➢ Left - Left rotation

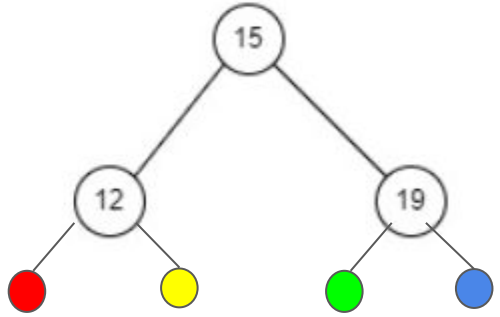


Note: Coloured Node indicate if they exist, what will be their position after rotation

Type of Rotations

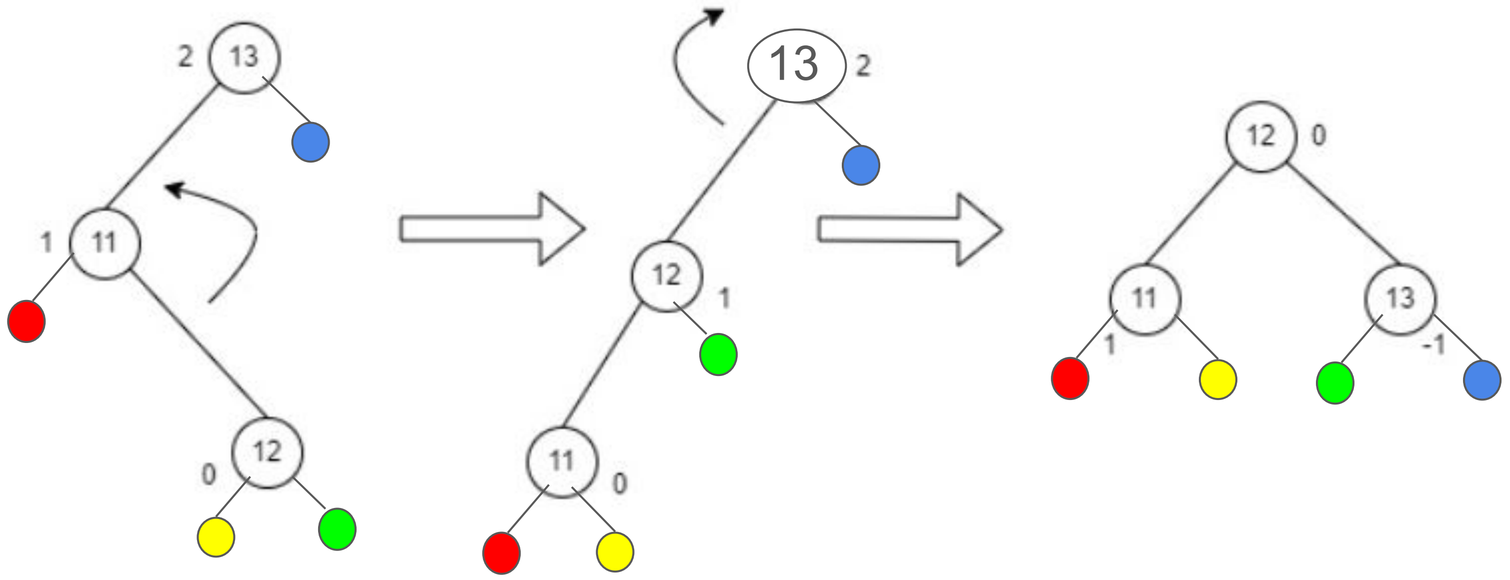
- Right - Right rotation





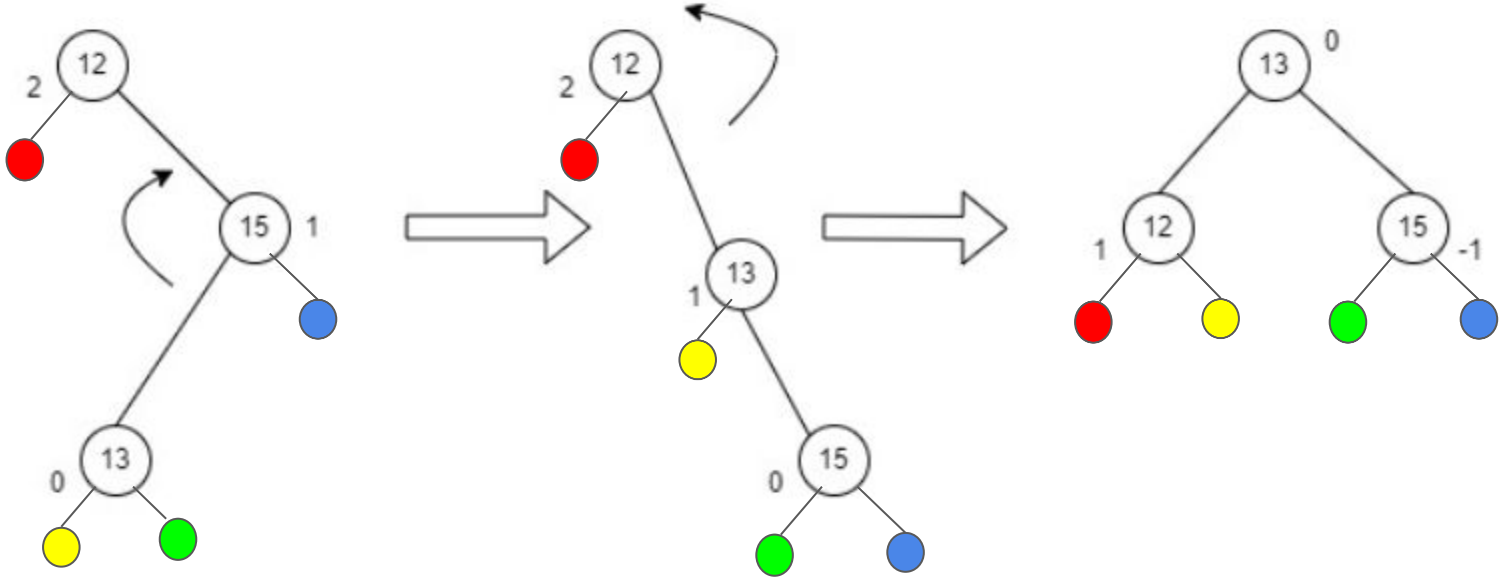
Type of Rotations

➢ Left - Right rotation



Type of Rotations

➢ Right - Left rotation



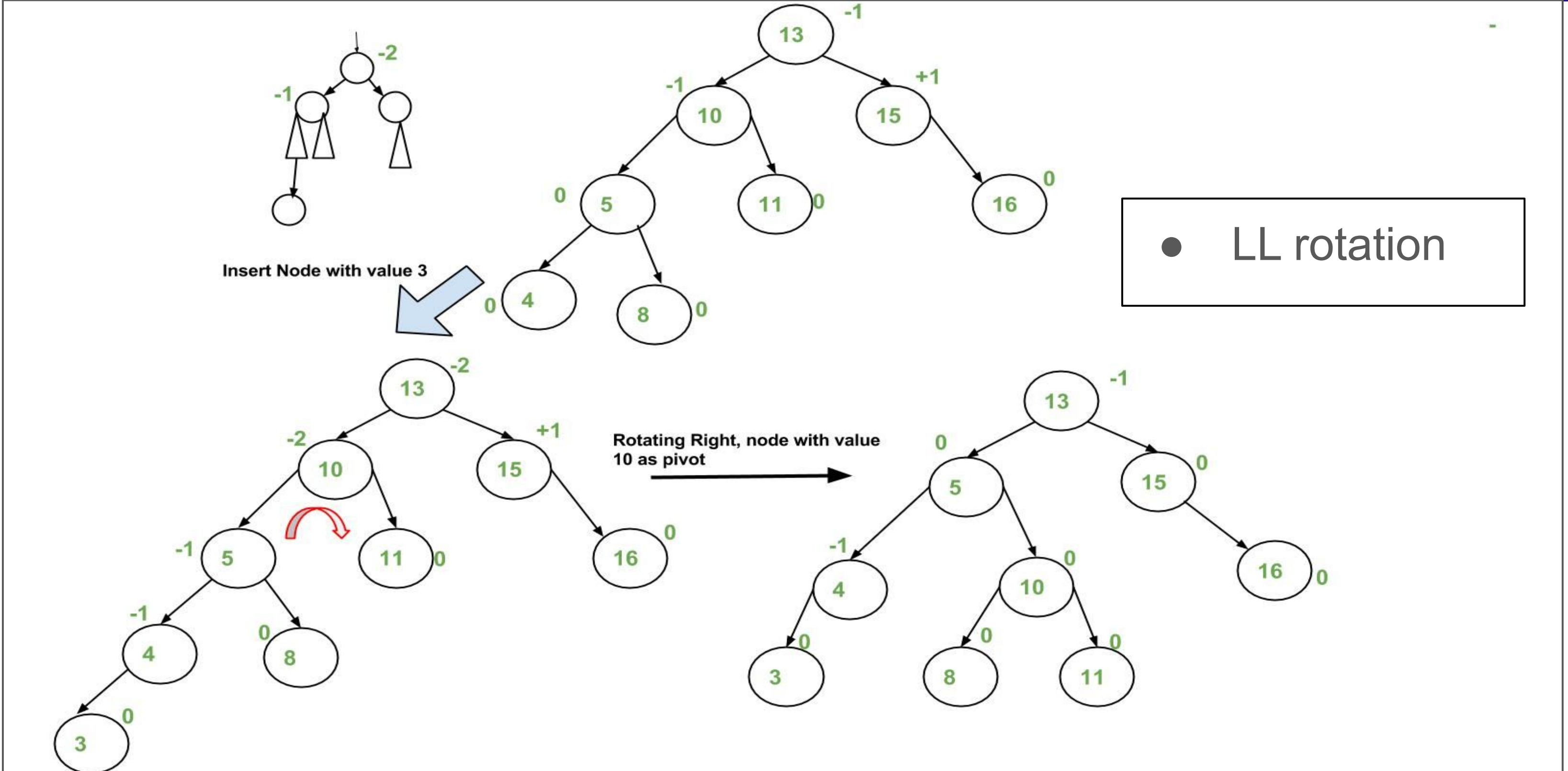
Insert

- Search is same as BST
- Insert is same as BST , after each insert check balance factor. If invalid, find rotation points and apply.

➢ Step 1: start journey from the inserted node towards the root
➢ Step 2: if found any invalid balance factor, stop and rotate
➢ That invalid node and previous 2 node in the journey path, will be the rotation points

- **Note:** there can be multiple invalid node after an Insertion / deletion , but we will take the node which is closest to the point of Insertion / deletion.

Insert



- LL rotation

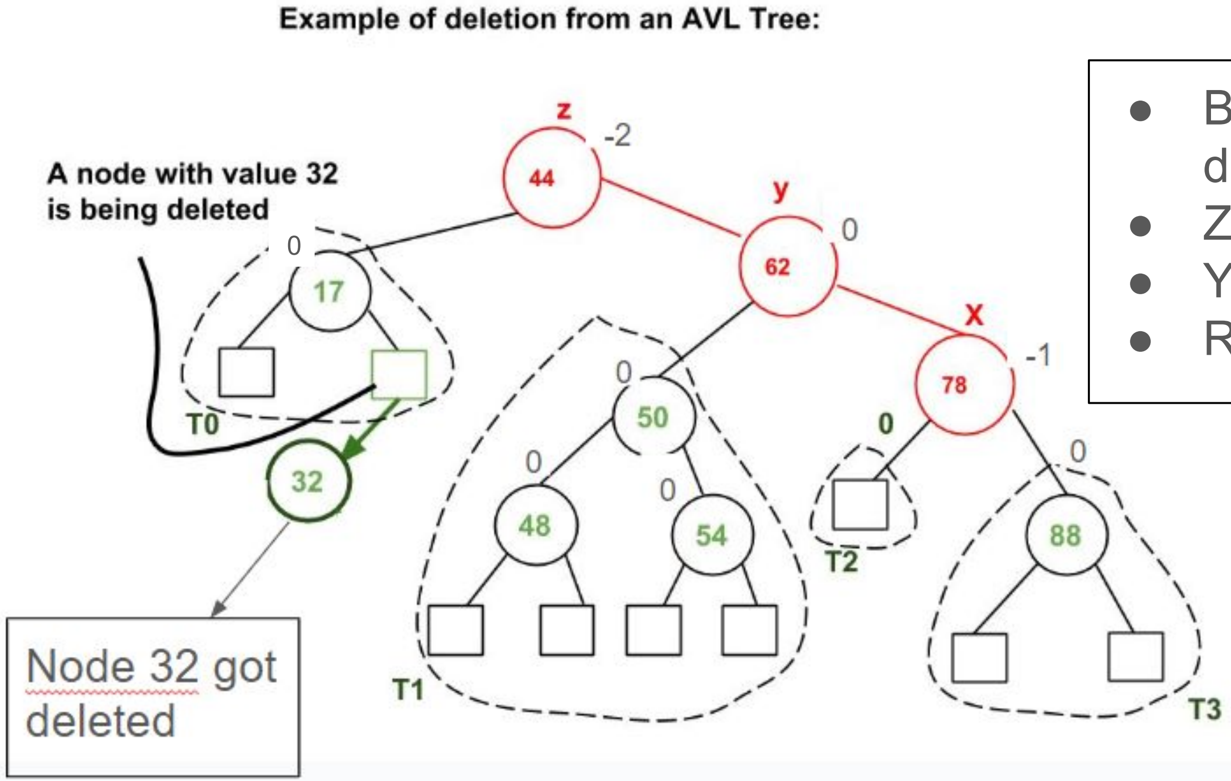
Delete

- Delete is same as BST. Now check the balance factor-

➢ Step 1: start journey from the deleted node position towards the root
➢ Step 2: if found any invalid balance factor, stop and rotate

➢ If balance factor is greater than 1, get the balance factor of left subtree. If balance factor of the left subtree is greater than or equal to 0, then it is Left Left rotation, else Left Right rotation.
➢ If balance factor is less than -1, get the balance factor of right subtree. If the balance factor of the right subtree is smaller than or equal to 0, then it is Right Right rotation, else Right Left rotation.

Delete

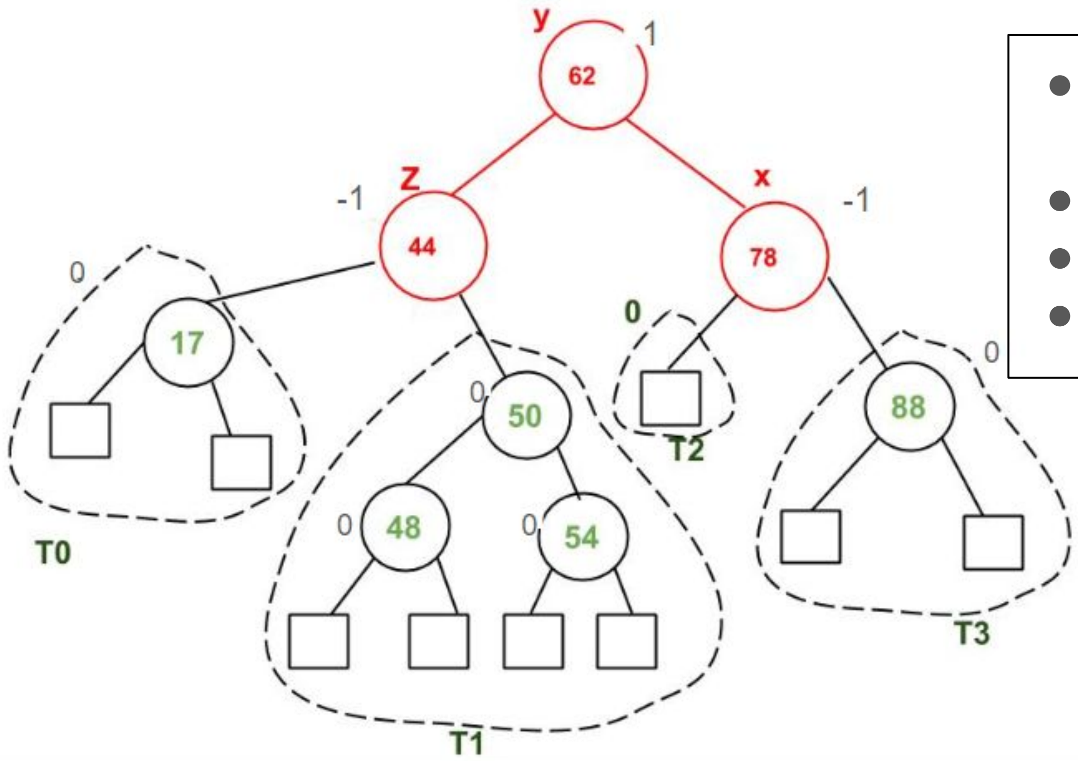


Example of deletion from an AVL Tree:
A node with value 32 is being deleted
z
-2
44
y
0
62
X
78
-1
0
17
T0
32
0
50
0
48
0
54
T1
0
T2
0
88
T3

Node 32 got deleted

- Balance factor after deletion
- Z invalid
- Y Balance factor >= 0
- RR rotation

Delete



- Balance factor after deletion
- Z invalid
- Y Balance factor >= 0
- RR rotation

Question

Questions will be given-
Implement AVL tree that we have discuss
Time - 2 hr.
Implement with menu driven format with functionality **(AVL)**-

- Insert
- Search
- Delete
- Find_height
- 3 types of ordering
- Level order and zigzag order

(**Note:** try to keep menu interface same as previous assignment, that way you can understand the difference between 2 approach)

Resources

- Tree - https://www.w3schools.com/dsa/dsa_theory_trees.php
- Insert, search, delete - https://www.javatpoint.com/binary-search-tree
- Preorder, Inorder, Postorder - https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/
- Level order - https://www.geeksforgeeks.org/level-order-tree-traversal/
- Rotations - https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm
- Insert AVL - https://www.geeksforgeeks.org/insertion-in-an-avl-tree/?ref=lbp
- Delete AVL - https://www.geeksforgeeks.org/deletion-in-an-avl-tree/?ref=lbp
- AVL tree - **well build it on your own**

Thank You!