Tree:-

The tree is a non-linear data structure; in which we insert the data in a hierarchical structure.

In our project requirement, if any hierarchical data structure is required, it is highly recommended to use a tree data structure.

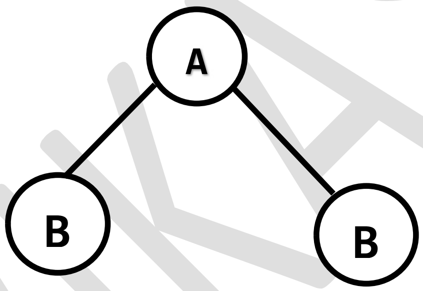
- Tree is a non-linear data structure.
- Tree is a collection of elements or nodes.
- In the tree hierarchy top of the node is marked as the root node and the remaining node is divided into multiple subtree below the root.

Uses of tree data structure:-

1. Banking Application.
2. File System.
3. Family hierarchy.

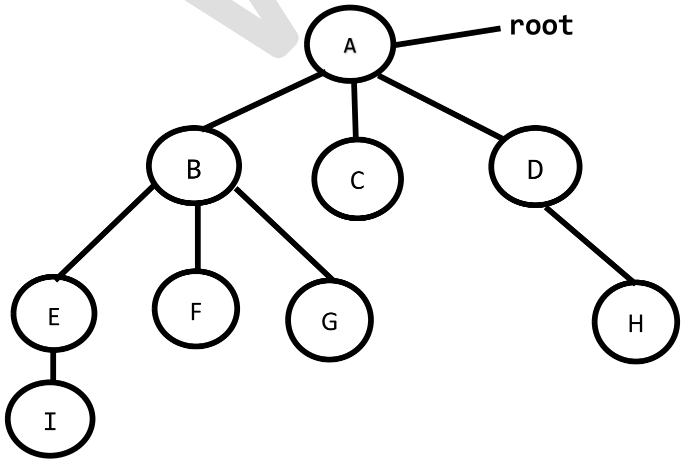
Basic Terminology in Tree Data Structure:-

Node:- Every individual element of a tree is known as node. Node storing a value and it connect or linked with other node.

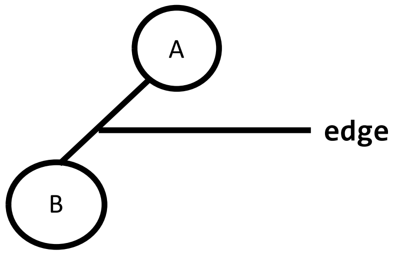


A B and C are node.

Root:-It is top of the node of tree hierarchy, they have no any parent.



Edge:- It is used to connect of two nodes.



Path:- It is sequence of nodes from source node till destination node.

Example:- Path between A to G is A-B-G or A to B and B to G.

Leaf node:-The nodes which have no any child called leaf node.

Example:-C I F G H is leaf node.

Height or Depth of Tree:-The number of edge on longest path between root and leaf.

Example:- 3 is height or depth of above tree, because A to I three edges are involved and it is longest path between root and leaf.

Level of tree:-In tree all nodes present in different level.

Root node level is 0(zero).

Direct child of root node level is 1(one).

Grand child of root node level is 2(two).

A----------------level 0.

B C D------------level 1.

E F G H----------level 2.

Parent node and Child node:-In the tree hierarchy two nodes are connected to each other, the top hierarchy kwon as the parent node, and the bottom hierarchy known as the child node.

Example:- A is a parent node of BCD and B is a parent of EGF.

Sibling node:-If more than one node has the same parent node then it is sibling.

BCD is sibling node.

EFG is sibling node.

Ancestor node: - A node that connected to all lower level node is called ancestor node.

Example:- E B A where BA is an ancestor of E and A is an ancestor of B.

Degree of node:-The degree of a node is the number of subtrees coming out of a node.

Example:-Degree of the B node is 3, degree of the C node is 0.

Degree of tree node:-The maximum degree of a node is called as the degree of the tree.

Example:-Degree of the above tree is 3.

Predecessors node:-When we display or traverse a tree and a node came before some other node then that node is known as the predecessor's node.

Example:-B is the predecessor of E.

Successor node:- When we display or traverse a tree and a node came next to some other node then that node is known as the successors' node.

Example:- E is the successor of B.

Types of Tree:-Tree are divided into different parts based on properties and characteristics.

1. Binary Tree.
2. Binary Search Tree.
3. AVL Tree.
4. B-Tree.
5. B+ Tree.
6. Red-Black Tree.
7. Segment.

Binary Tree:-

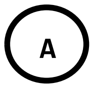
A binary Tree is one type of tree data structure, where each node has a maximum of two children and a minimum of zero children.

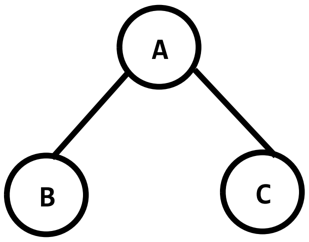
The child's name is the left child and the right child.

Properties of Binary Tree:-

1. Maximum number of nodes on level i of a binary tree is 2^i.
   Example:-Root level is 0 (zero), so the number of nodes at root is 2^0=1
2. There is only one path from root to any node.
3. Tree with 'N' nodes has exactly 'N-1' edges connecting these nodes.

Example:- All are binary tree except last.

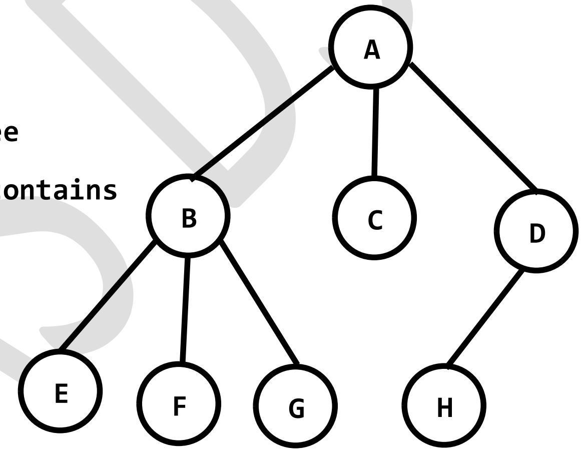




This is not binary tree

because A and B node contains

more than two node.



Example:-1 Build a Binary Tree.

```
package com.ds;
public class BT
{
static int index=-1;
class Node{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
Node buildTree(int[] nodes){
index++;
if(nodes[index]==-1)
return null;
Node node = new Node(nodes[index]);
node.left = buildTree(nodes);
node.right = buildTree(nodes);
return node;
}
public static void main(String[] args)
{
int[] nodes = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};
BT obj = new BT();
System.out.println(obj.buildTree(nodes).data);//1
//System.out.println(obj.buildTree(nodes).left.data);
//System.out.println(obj.buildTree(nodes).right.data);
}
}
```

Traverse data from Binary Tree:-

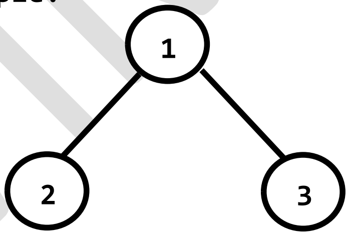
There are four ways to traverse the data from a binary Tree.

1. Pre Order:-In this we traverse the data in the following way ROOT_NODE----LEFT_NODE----RIGHT_NODE.

2. In Order:- In this we traverse the data in the following way LEFT_NODE----ROOT_NODE----RIGHT_NODE.

3. Post Order:- In this we traverse the data in the following way LEFT_NODE----RIGHT_NODE----ROOT_NODE.

4. Level Order:-In this we traverse the data based on the level of the tree.Example:-



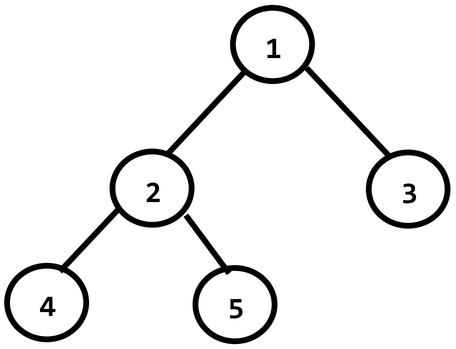
Pre Order:-1-2-3

In Order:-2-1-3

Post Order:-2-3-1

Level Order:-1

2 3



Pre Order:-1-2-4–5-3

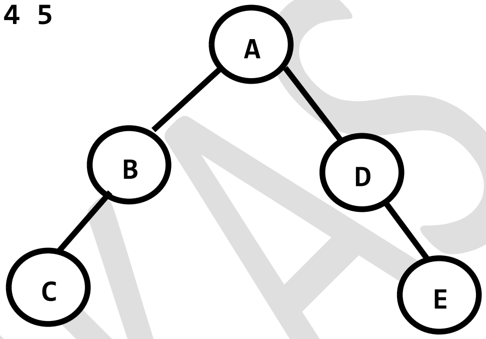
In Order:-4-2-5-1-3

Post Order:-4-5-2-3-1

Level Order:-1

2 3

4 5



Pre Order:- A-B-C-D-E

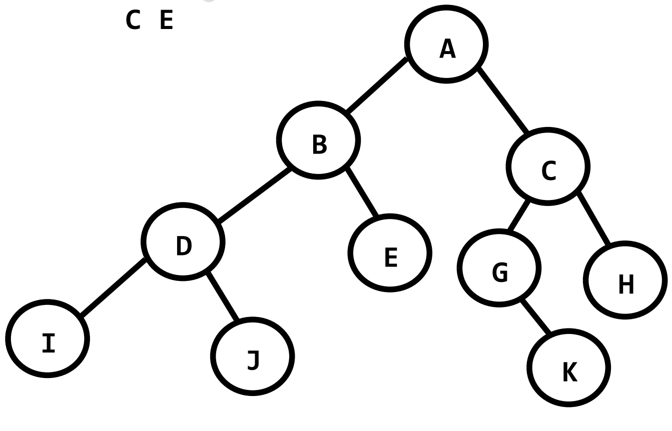
In Order:- C-B-A-D-E

Post Order:-C-B-E-D-A

Level Order: A

B D

C E



Pre Order:-A-B-D-I-J-E-C-G-K-H

In Order:-I-D-J-B-E-A-G-K-C-H

Post Order:-I-J-D-E-B-K-G-H-C-A

```
Level Order:-A
             B C
             D E G H
             I J K
```

Example:-Traverse the data from the Binary Tree.

Node.java

```java
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

BTTraverse.java

```java
package com.ds;
public class BTTraverse
{
Node root;
public BTTraverse()
{
root = null;
}

void preOrder(Node node)
{
if(node==null)
return;
System.out.print(node.data+" ");
preOrder(node.left);
preOrder(node.right);
}
void inOrder(Node node)
{
```

```
if(node==null)
return;
inOrder(node.left);
System.out.print(node.data+" ");
inOrder(node.right);
}
void postOrder(Node node)
{
if(node==null)
return;
postOrder(node.left);
postOrder(node.right);
System.out.print(node.data+" ");
}
public static void main(String[] args)
{
BTTraverse obj = new BTTraverse();
obj.root = new Node(1);
obj.root.left = new Node(2);
obj.root.right = new Node(3);
obj.root.left.left = new Node(4);
obj.root.left.right = new Node(5);
obj.root.right.right = new Node(6);
System.out.print("InOrder Traversal ===> ");
obj.inOrder(obj.root);
System.out.println();
System.out.print("PreOrder Traversal ===> ");
obj.preOrder(obj.root);
System.out.println();
System.out.print("PostOrder Traversal ===> ");
obj.postOrder(obj.root);
System.out.println();
}
}
```

Result:-

```
InOrder Traversal ===> 4 2 5 1 3 6
PreOrder Traversal ===> 1 2 4 5 3 6
PostOrder Traversal ===> 4 5 2 6 3 1
```

Example:-Traverse the data in Level Order.

Node.java

```
package com.ds;

public class Node
{
```

```
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

BTLavelOrder.java

```
package com.ds;
import java.util.LinkedList;
import java.util.Queue;
public class BTLavelOrder
{
Node root;
public BTLavelOrder()
{
root = null;
}
public static void levelOrderTraversal(Node node){
if(node==null)
return;
Queue<Node> q = new LinkedList<>();
q.add(node);
q.add(null);
while(!q.isEmpty()){
Node cur = q.remove();
if(cur==null){
System.out.println();
if(q.isEmpty())
break;
else
q.add(null);
}
else{
System.out.print(cur.data+" ");
if(cur.left!=null)
q.add(cur.left);
if(cur.right!=null)
q.add(cur.right);
}
}
}
public static void main(String[] args)
{
BTLavelOrder obj = new BTLavelOrder();
obj.root = new Node(1);
obj.root.left = new Node(2);
obj.root.right = new Node(3);
obj.root.left.left = new Node(4);
obj.root.left.right = new Node(5);
obj.root.right.right = new Node(6);
System.out.println("Level Order Traversal ");
levelOrderTraversal(obj.root);
}
}
```

Result:-

```
Level Order Traversal
1
2 3
4 5 6
```

Example:-In this application we get the number of nodes, the sum of all nodes, the height of the tree, and search for a particular node.

Node.java

```
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

NodeCount.java

```
package com.ds;
public class NodeCount
{
Node root;
public NodeCount()
{
root = null;
}
public static int countNodes(Node node)
{
if(node==null)
return 0;
int ln = countNodes(node.left);
int rn = countNodes(node.right);
return ln+rn+1;
}
public static int sumOfNodes(Node node){
if(node==null)
return 0;
int ls = sumOfNodes(node.left);
int rs = sumOfNodes(node.right);
return ls+rs+node.data;
}
public static int height(Node node){
if(node==null)
return 0;
int lh = height(node.left);
int rh = height(node.right);
return Math.max(lh,rh)+1;
}

public static boolean search(Node node,int data){
if(node==null)
return false;
if(node.data == data)
return true;
if(search(node.left,data))
return true;
if(search(node.right,data))
return true;
return false;
}

public static void main(String[] args)
{
NodeCount obj=new NodeCount();
obj.root = new Node(1);
obj.root.left = new Node(2);
obj.root.right = new Node(3);
obj.root.left.left = new Node(4);
obj.root.left.right = new Node(5);
obj.root.right.left = new Node(6);
obj.root.right.right = new Node(7);
System.out.println("Number of Node="+countNodes(obj.root));
System.out.println("Sum of Node="+sumOfNodes(obj.root));
System.out.println("Height of Tree="+height(obj.root));
System.out.println("7 is found ="+search(obj.root,7));
System.out.println("11 is found ="+search(obj.root,11));
}
}
```

Result:-

```
Number of Node=7
Sum of Node=28
Height of Tree=3
7 is found =true
11 is found =false
```

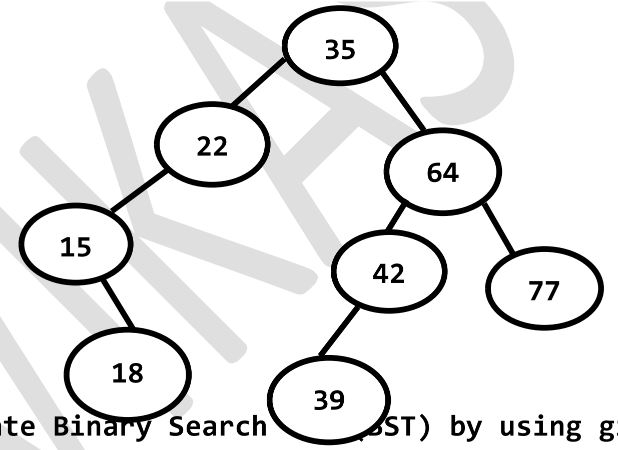
Binary Search Tree:-

- Binary Search Tree is one type of tree data structure where each node contains maximum two elements.
- It is extension of binary tree.
- In Binary Search Tree always left node is less than root, right node is greater than root.
- Binary Search Tree duplicate node we cannot insert.

Basic operations on a BST

- Create
- Insert
- Search
- Delete

Example:-



Example:-Create Binary Search (BST) by using given array in order form.

Node.java

```
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

BSTDemo.java

```
package com.ds;
public class BSTDemo
{
Node root;
public BSTDemo()
{
root = null;
}
static Node createBST(int[] nodes,int start,int end){
Node node = null;
if(start>end)
{
return null;
}
else
{
int mid = (start+end)/2;
node = new Node(nodes[mid]);
node.left = createBST(nodes,start,mid-1);
node.right = createBST(nodes,mid+1,end);
return node;
}
}
public static void main(String[] args)
{
BSTDemo obj = new BSTDemo();
int[] nodes = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
obj.root =createBST(nodes,0,nodes.length-1);
System.out.println(obj.root.data);//5
}
}
```

Example:-In this example we insert random data in a binary search tree.

Node.java

```
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
```

```
this.data = data;
this.left = null;
this.right = null;
}
}
```

BSTInsert.java

```
package com.ds;
public class BSTInsert
{
Node root;
BSTInsert()
{
root = null;
}
void insertNode(int data){
root = insertNode(root,data);
}
Node insertNode(Node node,int data){
if(node==null)
{
node = new Node(data);
}
else{
if(data<node.data)
{
node.left = insertNode(node.left,data);
}
else
{
node.right = insertNode(node.right,data);
}
}
return node;
}
void inOrder(Node node){
if(node==null)
{
return;
}
else
{
inOrder(node.left);
System.out.print(node.data+" ");
inOrder(node.right);
}
}
public static void main(String[] args)
{
BSTInsert obj = new BSTInsert();
obj.insertNode(2);
obj.insertNode(1);
obj.insertNode(3);
obj.insertNode(4);
obj.inOrder(obj.root);//1-2-3-4
}
}
```

Result:-1 2 3 4

Example:-In this example we perform the following operation

Search particular elements, find max node, find min node

Node.java

```
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

BSTOperation.java

```
com.ds;
public class BSTOperation
{
static Node root;
public BSTOperation()
{
root = null;
}
//Insert Operation
void insertNode(int data)
{
root = insertNode(root,data);
}
```

```
Node insertNode(Node node,int data)
{
if(node==null)
{
node = new Node(data);
}
else
{
if(data<node.data)
{
node.left = insertNode(node.left,data);
}
else
{
node.right = insertNode(node.right,data);
}
}
return node;
}
//Traverse
void inOrder(Node node)
{
if(node==null)
{
return;
}
else
{
inOrder(node.left);
System.out.print(node.data+" ");
inOrder(node.right);
}
}
//Search Operation
static boolean search(int value)
{
Node curr = root;
while(curr!=null)
{
if(value == curr.data)
{
return true;
}
else if(value<curr.data)
{
curr = curr.left;
}
else
{
```

```
curr = curr.right;
}
}
return false;
}
//Find Max Value
static Node findMaxNode(Node node)
{
if(node==null)
{
return null;
}
while(node.right!=null)
{
node = node.right;
}
return node;
}
//find Min Value
static Node findMinNode(Node node){
if(node==null)
{
return null;
}
while(node.left!=null)
{
node = node.left;
}
return node;
}
static boolean isBST(Node node)
{
if(node==null)
{
return true;
}
if(node.left!=null && findMaxNode(node.left).data>node.data)
{
return false;
}
if(node.right!=null && findMinNode(node.right).data<node.data)
{
return false;
}
return isBST(node.left) && isBST(node.right);
}
public static void main(String[] args)
{
BSTOperation obj = new BSTOperation();
```

```
obj.insertNode(6);
obj.insertNode(4);
obj.insertNode(2);
obj.insertNode(5);
obj.insertNode(1);
obj.insertNode(3);
obj.insertNode(8);
obj.insertNode(7);
obj.insertNode(9);
obj.insertNode(10);
obj.inOrder(obj.root);//1-2-3-4-5-6-7-8-9-10
System.out.println();
System.out.println("9 Is found = "+BSTOperation.search(9));//true
System.out.println("11 Is found = "+BSTOperation.search(11));//false
System.out.println("Max Value is = 
"+BSTOperation.findMaxNode(obj.root).data);//9
System.out.println("Min Value is = 
"+BSTOperation.findMinNode(obj.root).data);//9
System.out.println("Is it BST = "+BSTOperation.isBST(obj.root));//
}

}
```

Result:-

```
1 2 3 4 5 6 7 8 9 10
9 Is found = true
11 Is found = false
Max Value is = 10
Min Value is = 1
Is it BST = true
```

Delete operation perform on BTS:- When we delete the data from BTS then three cases will create.

Case 1: deleting a node that does not have any child nodes.

Case 2: Deleting a node which is having a single child.

Case 3: Deleting a node which is having two children.

Example:-

Node.java

```
package com.ds;
public class Node
{
int data;
Node left;
Node right;
Node(int data){
this.data = data;
this.left = null;
this.right = null;
}
}
```

BSTDelete.java

```
package com.ds;
public class BSTDelete
{
static Node root;
public BSTDelete()
{
root = null;
}
Node insertNode(Node node,int data){
if(node==null)
{
node = new Node(data);
}
else
{
if(data<node.data)
{
node.left = insertNode(node.left,data);
}
else
{
node.right = insertNode(node.right,data);
}
}
return node;
}
void inOrder(Node node){
if(node==null)
return;
else
{
inOrder(node.left);
System.out.print(node.data+" ");
inOrder(node.right);
}
}
static Node delete(Node node,int value){
if(node.data <value)
{
node.right = delete(node.right,value);
}
else if(node.data>value)
{
node.left = delete(node.left,value);
}
else
{
//case1: no child (leaf)
if(node.left==null && node.right==null)
{
return null;
}
//case2: one child
else if(node.left==null)
{
return node.right;
}
else if(node.right==null)
{
return node.left;
}
else
{
//case3: two children
Node is = findInOrderSuccessor(node.right);
node.data = is.data;
node.right = delete(node.right,is.data);
}
}
return node;
}
static Node findInOrderSuccessor(Node node)
{
while(node.left!=null)
{
node = node.left;
}
return node;
}
public static void main(String[] args)
{
BSTDelete obj = new BSTDelete();
obj.root = new Node(10);
obj.root.left = new Node(5);
obj.root.right = new Node(15);
obj.root.left.left = new Node(3);
obj.root.left.right = new Node(7);
obj.root.left.right.left = new Node(6);
obj.root.right.left = new Node(12);
obj.root.right.right = new Node(17);
obj.root.right.right.left = new Node(16);
obj.root.right.right.right = new Node(18);
obj.inOrder(obj.root);//3-5-6-7-10-12-15-16-17-18
```

```
System.out.println();
BSTDelete.delete(obj.root,17);
obj.inOrder(obj.root);//5-6-7-10-12-15-16-17-18
}
}
```

Result:-

```
3 5 6 7 10 12 15 16 17 18
3 5 6 7 10 12 15 16 18
```

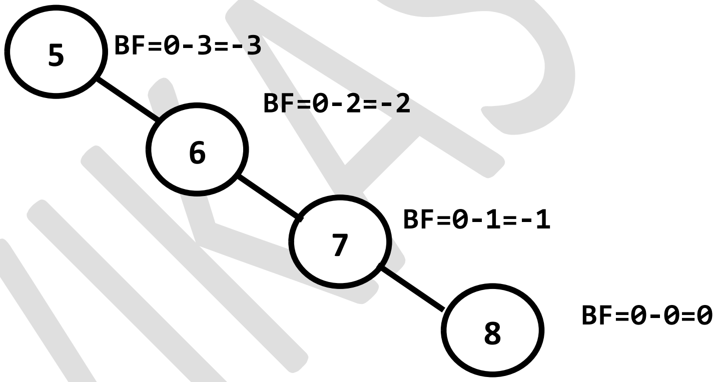
AVL Tree:-

- AVL tree is an extension of BST.
- AVL tree is the self-balance tree, which means every node has a balance factor and the range of balance factor -1 to 1 (-1, 0, and 1).
- In BST each node has three things left node reference, right node reference, and data.
  But in the AVL tree, each node has one extra information that balances the factor.

How to find the balance factor any node ::

Balance Factor=(Height of left sub-tree)-(Height of right sub-tree)

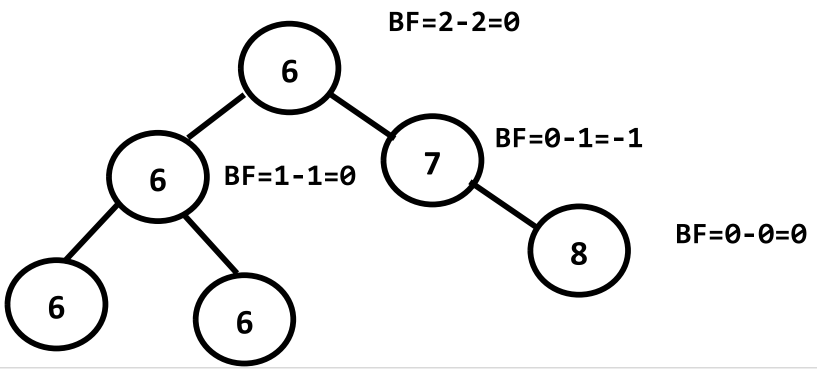
Example:-



This is unbalance BST because

Balance factor is more than -1 to 1.

Example:-2



BF=0-0=0   BF=0-0=0 This tree is balance and AVL because the balance factor is between -1 and 1.

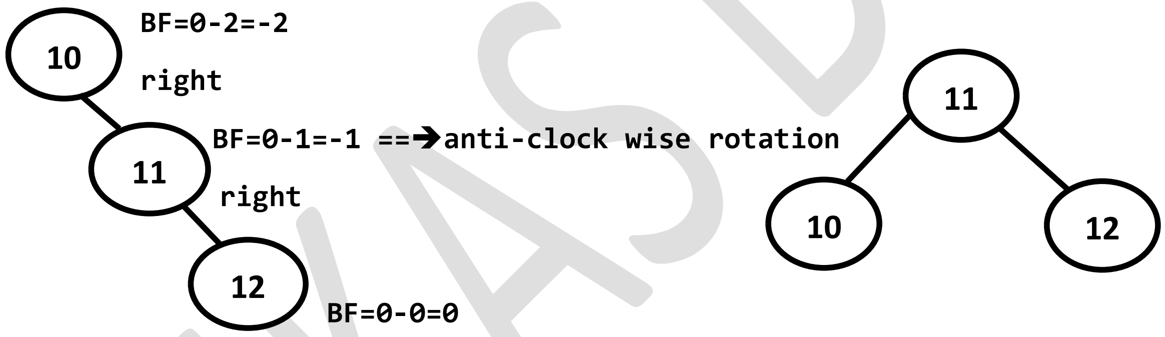
How to convert unbalanced BST to balance BST or AVL tree.

We perform some node rotation when we make a balanced tree from an un-balance tree.

Case:-1

When unbalanced BST is the right-right combination, then we perform anti-clockwise (left) rotation.

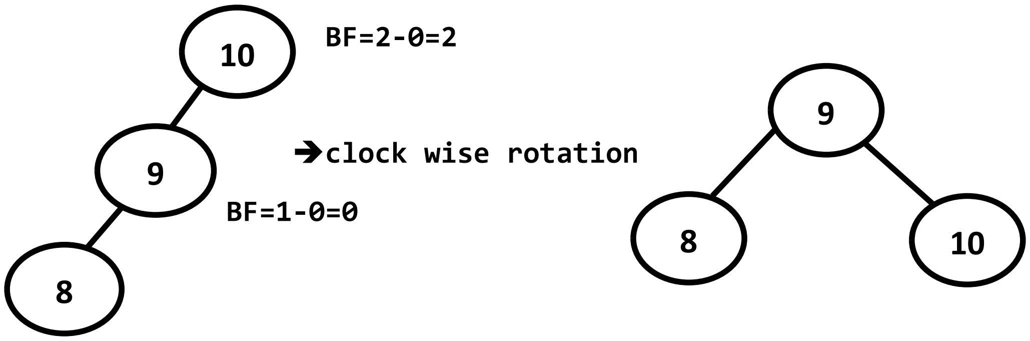
Example:-10,11,12



Case:-2

When unbalanced BST is the Left-Left combination, then we perform clockwise (Right) rotation.

Example:-10,9,8

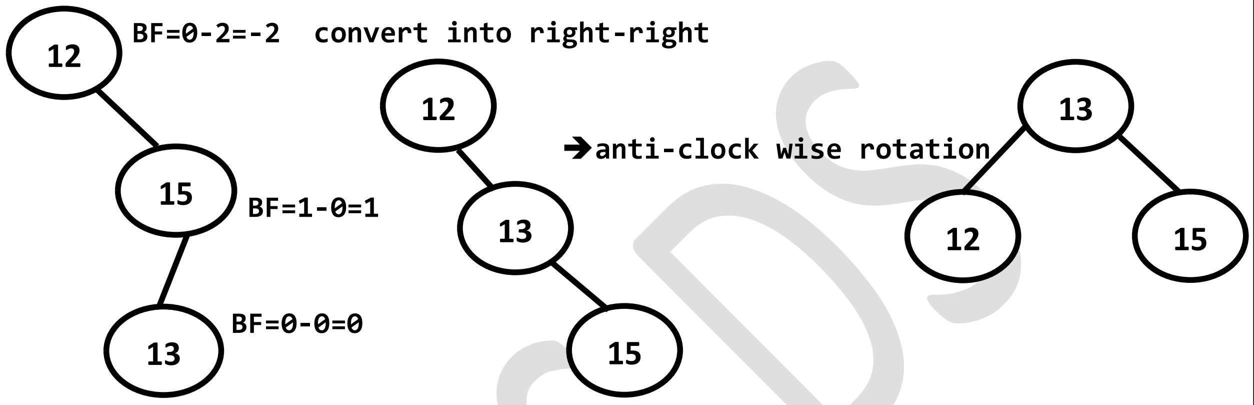


BF=0-0=0

Case:-3

When unbalanced BST is the Right-Left combination, then we make unbalance tree in Right-Right combination and perform anti-clockwise (Left) rotation.

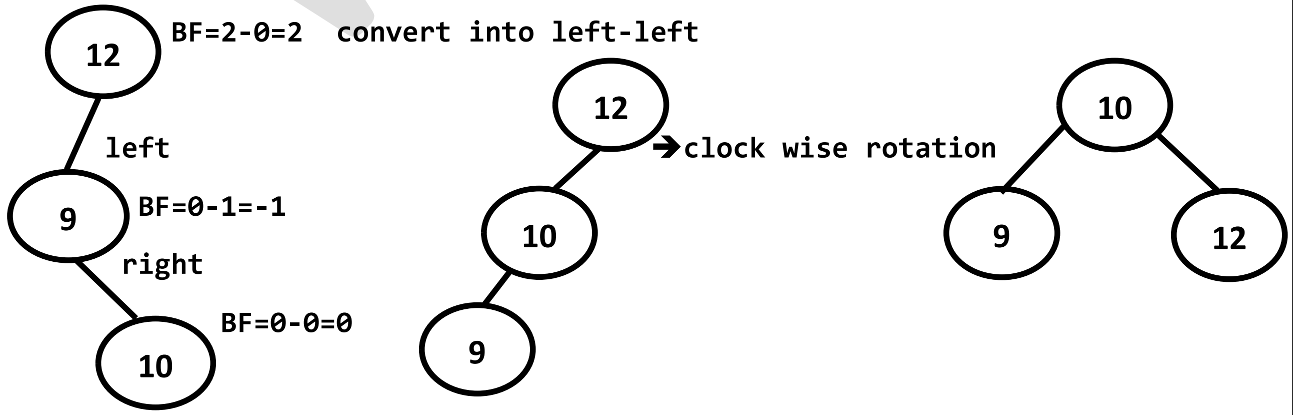
Example:-12,15,13



Case:-4

When unbalanced BST is the Left-Right combination, then we make unbalance tree in Left-Left combination and perform clockwise (right) rotation.

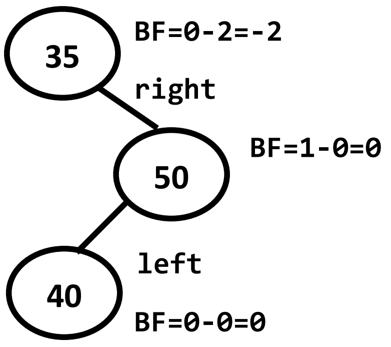
Example:-12,9,10



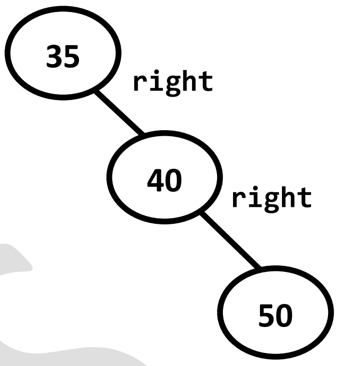
Example:- Create AVL tree(balance BST) by using given value.

35,50,40,25,30,60,78,20,28.

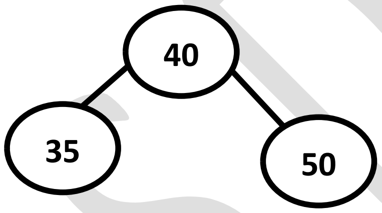
Step:-1



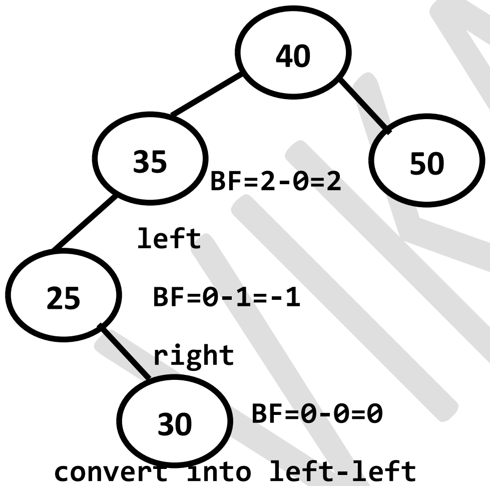
→convert into right-right



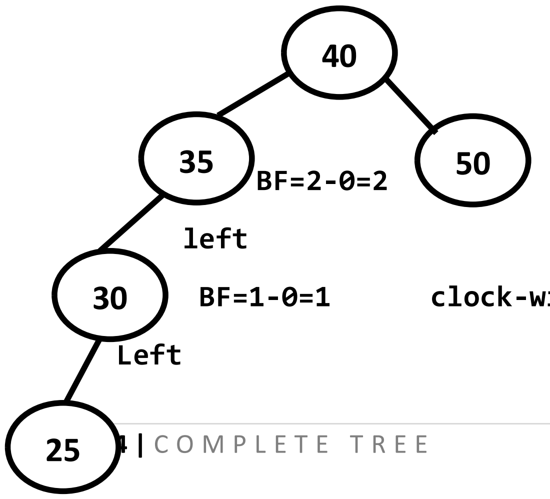
Anti-clockwise rotation:-



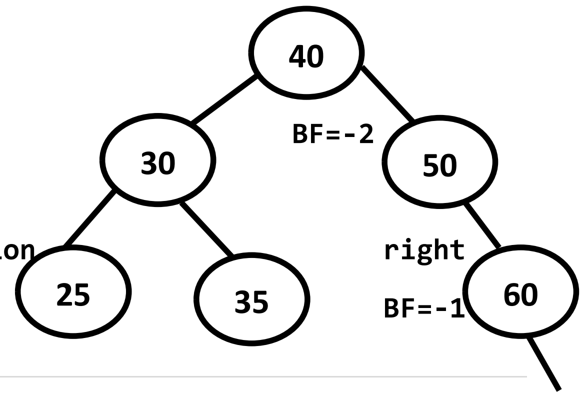
Step:-2



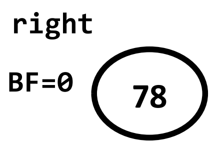
convert into left-left



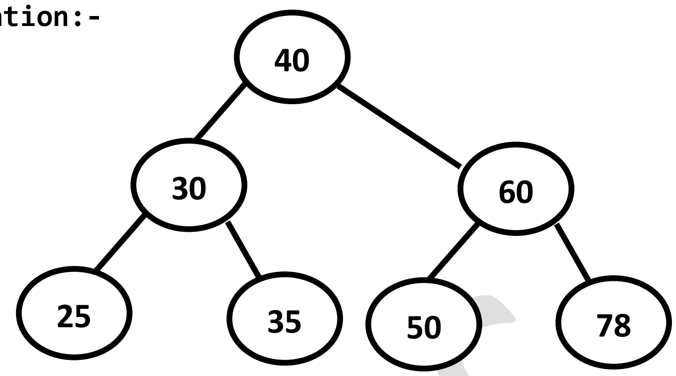
clock-wise rotation



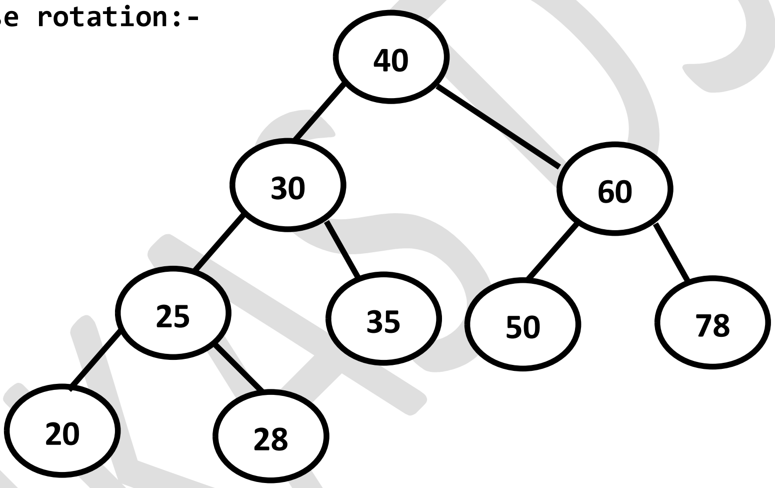
BF=0-0=0



Anti-clock wise rotation:-



Anti-clock wise rotation:-



Difference BST and AVL tree.

BST	AVL Tree
1. In BST not node not contain any balance factor	1.In AVL tree each node contain balance factor and range of balance factor is -1 to 1
2.Every BST is not AVL tree	2. Every AVL tree is BST
3.In BST each node contains thee field left subtree, data, and right subtree.	3.The AVL tree each node contains four fields left subtree, data, right subtree and balance factor.
4. It is not a balanced tree.	4. It is a balanced tree.

5. In BST insertion and deletion are very easy because no rotation is required.	5. In AVL tree insertion and deletion are very complex because multiple rotations are required to balance the tree.

Example:-In this program we perform multiple operations like

Height of tree, balance factor, insert, delete, in-order successors, pre-order, right rotation, left rotation.

Node.java

```
package com.ds;
public class Node
{
int data;
int ht;
Node left;
Node right;
Node(int data)
{
this.data = data;
this.left = null;
this.right = null;
this.ht = 1;
}
}
```

AVLTree.java

```
package com.ds;
public class AVLTree
{
static Node root;
AVLTree(){
root = null;
}
static int height(Node node){
if(node==null)
{
return 0;
}
else
{
return node.ht;
}
}
static int getBalance(Node node){//balance factor HL-HR
if(node==null)
```

```
{
return 0;
}
{
return height(node.left)-height(node.right);
}
}
static void insert(int value)
{
root = insert(root,value);
}
static Node insert(Node node,int value)
{
if(node==null)
{
return new Node(value);
}
if(value < node.data)
{
node.left = insert(node.left,value);
}
else if(value > node.data)
{
node.right = insert(node.right,value);
}
else
{
return node;//duplicate nodes
}
//balancing
node.ht = 1 + Math.max(height(node.left),height(node.right));
int bf = getBalance(node);
if(bf>1 && value < node.left.data)
{//LL case
return rightRotation(node);
}
if(bf<-1 && value > node.right.data){//RR case
return leftRotation(node);
}
if(bf>1 && value > node.left.data){//LR case
node.left = leftRotation(node.left);
return rightRotation(node);
}
if(bf<-1 && value<node.right.data){//RL case
node.right = rightRotation(node.right);
return leftRotation(node);
}
else
{
```

```
return node;
}
}
static void delete(int value)
{
root = delete(root,value);
}
static Node delete(Node node,int value){
if(node.data < value)
node.right = delete(node.right,value);
else if(node.data > value)
node.left = delete(node.left,value);
else{
if(node.left==null && node.right==null)//case1:no children
return null;
if(node.left==null)//case2: one child (right)
return node.right;
else if(node.right==null) //case2: one child (left)
return node.left;
Node is = findInOrderSuccessor(node.right);
node.data = is.data;
node.right = delete(node.right,is.data);
}
if(node==null)
return node;
node.ht = Math.max(height(node.left),height(node.right))+1;
int bf = getBalance(node);
if(bf>1 && getBalance(node.left)>=0)
return rightRotation(node);
if(bf>1 && getBalance(node.left)<0)
{
node.left = leftRotation(node.left);
return rightRotation(node);
}
if(bf<-1 && getBalance(node.right)<=0)
return leftRotation(node);
if(bf<-1 && getBalance(node.right)>0)
{
node.right = rightRotation(node.right);
return leftRotation(node);
}
return node;
}
static Node findInOrderSuccessor(Node node){
while(node.left!=null)
node = node.left;
return node;
}
static void preOrder(Node node){
```

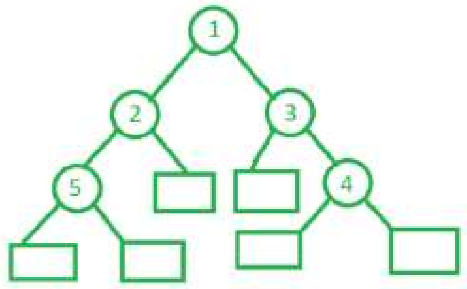
```
if(node==null)
return;
System.out.print(node.data+" ");
preOrder(node.left);
preOrder(node.right);
}
static Node rightRotation(Node y)
{
Node x = y.left;
Node T = x.right;
x.right = y;
y.left = T;
x.ht = 1+Math.max(height(x.left),height(x.right));
y.ht = 1+Math.max(height(y.left),height(y.right));
return x;
}
static Node leftRotation(Node x){
Node y = x.right;
Node T = y.left;
y.left = x;
x.right = T;
x.ht = 1+Math.max(height(x.left),height(x.right));
y.ht = 1+Math.max(height(y.left),height(y.right));
return y;
}

public static void main(String[] args)
{
AVLTree obj = new AVLTree();
AVLTree.insert(10);
AVLTree.insert(20);
AVLTree.insert(30);
AVLTree.insert(40);
AVLTree.insert(50);
AVLTree.insert(25);
AVLTree.preOrder(obj.root);
System.out.println();
AVLTree.delete(50);
AVLTree.preOrder(obj.root);
}
}
```

Binary extended Tree:-

A binary extended tree is one type of tree data structure in which all null(null reference) is replaced with a special node, which is known as the external node and non-null nodes are known as the internal node.

We used an Extended binary tree to make a binary tree, a complete binary tree.

Example:-



Where all ▭ node is an external node and all ◯ node is internal node.

It is useful for representation in algebraic expressions.

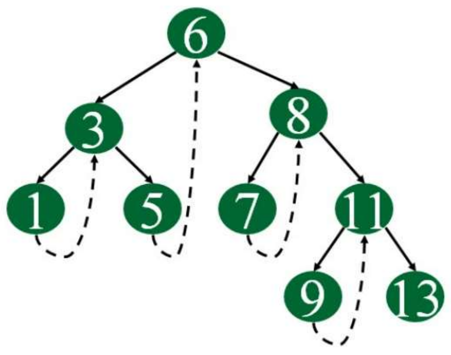
Threaded Binary Tree:-In binary tree, suppose we have n nodes then n+1 liked field (left, right reference) contains a null value, which is a waste of storage space. To overcome these problems null link replaces with a special link which is known as a thread and such type of binary tree with a thread is known as a threaded binary tree.

There are two types of threaded binary trees:-

- One-Way Threaded Binary Tree.
- Two-Way Threaded Binary Tree.

One-Way Threaded Binary Tree:-In this right null reference point to the in-order successors (If the Successors is available).

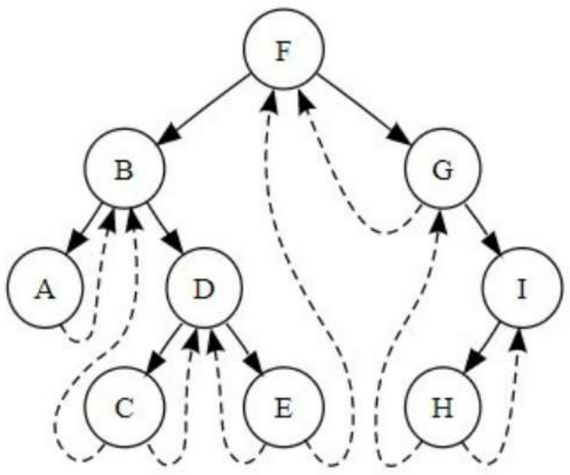
Example:-In-Order of given tree is:1,3,5,6,7,8,9,11,13



Single Threaded Binary Tree

Two-Way Threaded Binary Tree:-In this both left and right null reference points to the in-order predecessors and in-order successors (If predecessors and Successors are available).

Example:-In-order of given tree:A,B,C,D,E,F,G,H,I



Double Threaded Binary Tree

M-Way Search Tree:- M-Way tree is one type of Binary Tree.

In BST each node contains a maximum one element and two children.

But in the case of M-way Tree, each node contains how many maximum elements and children it depends on M-value.

Where m is the order of nodes.

Suppose M=3, then the node contains an element or key that is

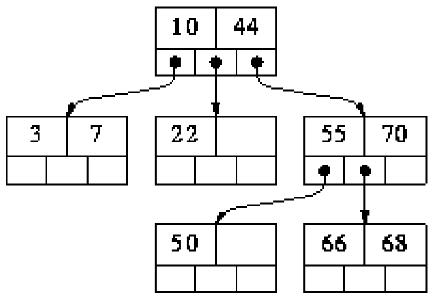
Key=M-1

=3-1

=2

And children=M

=3


10
44
3
7
22
55
70
50
66
68