<https://www.youtube.com/watch?v=jaUi-_S6ROI&list=PLsyeobzWxl7oRKwDi7wjrANsbhTX0IK0J&index=16>

Tree in Java

Types of Tree

1. Strict Binary Tree (either 2 or no children)
2. Full Binary Tree (all leaf nodes must be at same level)
3. Complete Binary Tree (all leaf nodes at level L or level L-1)

Height Binary Tree (Children route)

Depth Binary Tree (Parent route)

Height of root node / Height of Tree (From root to last leaf node)

Depth of root node = 0

Binary Tree

Binary Search Tree (Left Node < Parent < Right Node) (Ologn)

Complexity of BST : O(logn)

**Inserting Values in Binary Tree**

**InOrder/PreOrder/PostOrder Traversal**

**package** bst;

**public** **class** Node {

**int** data;

Node left;

Node right;

}

**package** bst;

**public** **class** BinarySearchTree {

**public** Node createNewNode(**int** k) {

Node newNode = **new** Node();

newNode.data = k;

newNode.left = **null**;

newNode.right = **null**;

**return** newNode;

}

**public** Node insert(Node node, **int** value) {

**if** (node == **null**) {

**return** createNewNode(value);

}

**if** (value < node.data) {

node.left = insert(node.left, value);

} **else** **if** (value > node.data) {

node.right = insert(node.right, value);

}

**return** node;

}

// leaf with 0,1,2 children

**public** Node delete(Node node, **int** value) {

**if** (node == **null**) {

**return** **null**;

}

// recursive call

**if** (value < node.data) {

node.left = delete(node.left, value);

} **else** **if** (value > node.data) {

node.right = delete(node.right, value);

} **else** {

**if** (node.left == **null** || node.right == **null**) {

Node tempNode = **null**;

tempNode = node.left == **null** ? node.right : node.left;

**if** (tempNode == **null**)

**return** **null**;

**else**

**return** tempNode;

} **else** {

Node successor = getSuccessor(node);

node.data = successor.data;

node.right = delete(node.right, value);

**return** node;

}

}

**return** node;

}

**public** Node getSuccessor(Node node) {

**if** (node == **null**)

**return** **null**;

Node tempNode = node.right;

**while** (tempNode != **null**) {

tempNode = tempNode.left;

}

**return** tempNode;

}

**public** **void** inOrder(Node node) {

**if** (node == **null**)

**return**;

inOrder(node.left);

System.***out***.print(node.data + " ");

inOrder(node.right);

}

**public** **void** preOrder(Node node) {

**if** (node == **null**)

**return**;

System.***out***.print(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

**public** **void** postOrder(Node node) {

**if** (node == **null**)

**return**;

postOrder(node.left);

postOrder(node.right);

System.***out***.print(node.data + " ");

}

}

**package** bst;

**public** **class** BinarySearchTreeTester {

**public** **static** **void** main(String[] args) {

BinarySearchTree binarySearchTree = **new** BinarySearchTree();

Node root = **null**;

// Insert element in tree

// 8,3,6,10,4,7,1,14,13

root = binarySearchTree.insert(root, 8);

root = binarySearchTree.insert(root, 3);

root = binarySearchTree.insert(root, 6);

root = binarySearchTree.insert(root, 10);

root = binarySearchTree.insert(root, 4);

root = binarySearchTree.insert(root, 7);

root = binarySearchTree.insert(root, 1);

root = binarySearchTree.insert(root, 14);

root = binarySearchTree.insert(root, 13);

// root = binarySearchTree.delete(root, 1);

// root = binarySearchTree.delete(root, 14);

System.***out***.println("\nIn order Traversal");

binarySearchTree.inOrder(root);// LR`R

System.***out***.println("\nPre order Traversal");

binarySearchTree.preOrder(root);// R`LR

System.***out***.println("\nPost order Traversal");

binarySearchTree.postOrder(root);// LRR`

}

}

// Binary Search Tree

// 8

// 3 10

// 1 6 14

// 4 7 13

**Check if a Value Exist in a Binary Tree**

**package** bst;

**public** **class** Node {

**int** data;

Node left;

Node right;

}

**package** bst;

**public** **class** BinarySearchTree {

**public** Node createNewNode(**int** k) {

Node newNode = **new** Node();

newNode.data = k;

newNode.left = **null**;

newNode.right = **null**;

**return** newNode;

}

**public** Node insert(Node node, **int** value) {

**if** (node == **null**) {

**return** createNewNode(value);

}

**if** (value < node.data) {

node.left = insert(node.left, value);

} **else** **if** (value > node.data) {

node.right = insert(node.right, value);

}

**return** node;

}

// leaf with 0,1,2 children

**public** Node delete(Node node, **int** value) {

**if** (node == **null**) {

**return** **null**;

}

// recursive call

**if** (value < node.data) {

node.left = delete(node.left, value);

} **else** **if** (value > node.data) {

node.right = delete(node.right, value);

} **else** {

**if** (node.left == **null** || node.right == **null**) {

Node tempNode = **null**;

tempNode = node.left == **null** ? node.right : node.left;

**if** (tempNode == **null**)

**return** **null**;

**else**

**return** tempNode;

} **else** {

Node successor = getSuccessor(node);

node.data = successor.data;

node.right = delete(node.right, value);

**return** node;

}

}

**return** node;

}

**public** Node getSuccessor(Node node) {

**if** (node == **null**)

**return** **null**;

Node tempNode = node.right;

**while** (tempNode != **null**) {

tempNode = tempNode.left;

}

**return** tempNode;

}

**public** **void** inOrder(Node node) {

**if** (node == **null**)

**return**;

inOrder(node.left);

System.***out***.print(node.data + " ");

inOrder(node.right);

}

**public** **void** preOrder(Node node) {

**if** (node == **null**)

**return**;

System.***out***.print(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

**public** **void** postOrder(Node node) {

**if** (node == **null**)

**return**;

postOrder(node.left);

postOrder(node.right);

System.***out***.print(node.data + " ");

}

**public** **boolean** ifNodePresent(Node node, **int** value) {

**boolean** isPresent = **false**;

**if** (node == **null**) {

**return** **false**;

}

**while** (node != **null**) {

**if** (value < node.data) {

node = node.left;

} **else** **if** (value > node.data) {

node = node.right;

} **else** {

isPresent = **true**;

**break**;

}

}

**return** isPresent;

}

}

**public** **class** BinarySearchTreeTester {

**public** **static** **void** main(String[] args) {

BinarySearchTree binarySearchTree = **new** BinarySearchTree();

Node root = **null**;

// Insert element in tree

// 8,3,6,10,4,7,1,14,13

root = binarySearchTree.insert(root, 8);

root = binarySearchTree.insert(root, 3);

root = binarySearchTree.insert(root, 6);

root = binarySearchTree.insert(root, 10);

root = binarySearchTree.insert(root, 4);

root = binarySearchTree.insert(root, 7);

root = binarySearchTree.insert(root, 1);

root = binarySearchTree.insert(root, 14);

root = binarySearchTree.insert(root, 13);

System.***out***.println();

System.***out***.println(binarySearchTree.ifNodePresent(**null**, 3));

System.***out***.println(binarySearchTree.ifNodePresent(root, 3));

System.***out***.println(binarySearchTree.ifNodePresent(root, 13));

System.***out***.println(binarySearchTree.ifNodePresent(root, 14));

System.***out***.println(binarySearchTree.ifNodePresent(root, 15));

System.***out***.println(binarySearchTree.ifNodePresent(root, 30));

}

}

// Binary Search Tree

// 8

// 3 10

// 1 6 14

// 4 7 13

**Get Parent Node of a given value in Binary Search Tree**

**package** bst;

**public** **class** Node {

**int** data;

Node left;

Node right;

}

**package** bst;

**public** **class** BinarySearchTree {

**public** Node createNewNode(**int** k) {

Node newNode = **new** Node();

newNode.data = k;

newNode.left = **null**;

newNode.right = **null**;

**return** newNode;

}

**public** Node insert(Node node, **int** value) {

**if** (node == **null**) {

**return** createNewNode(value);

}

**if** (value < node.data) {

node.left = insert(node.left, value);

} **else** **if** (value > node.data) {

node.right = insert(node.right, value);

}

**return** node;

}

// leaf with 0,1,2 children

**public** Node delete(Node node, **int** value) {

**if** (node == **null**) {

**return** **null**;

}

// recursive call

**if** (value < node.data) {

node.left = delete(node.left, value);

} **else** **if** (value > node.data) {

node.right = delete(node.right, value);

} **else** {

**if** (node.left == **null** || node.right == **null**) {

Node tempNode = **null**;

tempNode = node.left == **null** ? node.right : node.left;

**if** (tempNode == **null**)

**return** **null**;

**else**

**return** tempNode;

} **else** {

Node successor = getSuccessor(node);

node.data = successor.data;

node.right = delete(node.right, value);

**return** node;

}

}

**return** node;

}

**public** Node getSuccessor(Node node) {

**if** (node == **null**)

**return** **null**;

Node tempNode = node.right;

**while** (tempNode != **null**) {

tempNode = tempNode.left;

}

**return** tempNode;

}

**public** **void** inOrder(Node node) {

**if** (node == **null**)

**return**;

inOrder(node.left);

System.***out***.print(node.data + " ");

inOrder(node.right);

}

**public** **void** preOrder(Node node) {

**if** (node == **null**)

**return**;

System.***out***.print(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

**public** **void** postOrder(Node node) {

**if** (node == **null**)

**return**;

postOrder(node.left);

postOrder(node.right);

System.***out***.print(node.data + " ");

}

**public** **boolean** ifNodePresent(Node node, **int** value) {

**boolean** isPresent = **false**;

**if** (node == **null**) {

**return** **false**;

}

**while** (node != **null**) {

**if** (value < node.data) {

node = node.left;

} **else** **if** (value > node.data) {

node = node.right;

} **else** {

isPresent = **true**;

**break**;

}

}

**return** isPresent;

}

**public** Node getParentNode(Node node, **int** value) {

**if** (node == **null**)

**return** **null**;

Node getParent = **null**;

**while** (node != **null**) {

**if** (value < node.data) {

getParent = node;

node = node.left;

} **else** **if** (value > node.data) {

getParent = node;

node = node.right;

} **else** {

**break**;

}

}

**if** (node != **null**)

**return** getParent;

**else**

**return** **null**;

}

}

**package** bst;

**public** **class** BinarySearchTreeTester {

**public** **static** **void** main(String[] args) {

BinarySearchTree binarySearchTree = **new** BinarySearchTree();

Node root = **null**;

// Insert element in tree

// 8,3,6,10,4,7,1,14,13

root = binarySearchTree.insert(root, 8);

root = binarySearchTree.insert(root, 3);

root = binarySearchTree.insert(root, 6);

root = binarySearchTree.insert(root, 10);

root = binarySearchTree.insert(root, 4);

root = binarySearchTree.insert(root, 7);

root = binarySearchTree.insert(root, 1);

root = binarySearchTree.insert(root, 14);

root = binarySearchTree.insert(root, 13);

Node parentNode = binarySearchTree.getParentNode(root, 990);

**if** (parentNode != **null**) {

System.***out***.println(parentNode.data);

} **else** {

System.***out***.println("Parent does not exist");

}

}

}

// Binary Search Tree

// 8

// 3 10

// 1 6 14

// 4 7 13

**Get Sibling Node of a given value in Binary Search Tree**

**package** bst;

**public** **class** Node {

**int** data;

Node left;

Node right;

}

**package** bst;

**public** **class** BinarySearchTree {

**public** Node createNewNode(**int** k) {

Node newNode = **new** Node();

newNode.data = k;

newNode.left = **null**;

newNode.right = **null**;

**return** newNode;

}

**public** Node insert(Node node, **int** value) {

**if** (node == **null**) {

**return** createNewNode(value);

}

**if** (value < node.data) {

node.left = insert(node.left, value);

} **else** **if** (value > node.data) {

node.right = insert(node.right, value);

}

**return** node;

}

// leaf with 0,1,2 children

**public** Node delete(Node node, **int** value) {

**if** (node == **null**) {

**return** **null**;

}

// recursive call

**if** (value < node.data) {

node.left = delete(node.left, value);

} **else** **if** (value > node.data) {

node.right = delete(node.right, value);

} **else** {

**if** (node.left == **null** || node.right == **null**) {

Node tempNode = **null**;

tempNode = node.left == **null** ? node.right : node.left;

**if** (tempNode == **null**)

**return** **null**;

**else**

**return** tempNode;

} **else** {

Node successor = getSuccessor(node);

node.data = successor.data;

node.right = delete(node.right, value);

**return** node;

}

}

**return** node;

}

**public** Node getSuccessor(Node node) {

**if** (node == **null**)

**return** **null**;

Node tempNode = node.right;

**while** (tempNode != **null**) {

tempNode = tempNode.left;

}

**return** tempNode;

}

**public** **void** inOrder(Node node) {

**if** (node == **null**)

**return**;

inOrder(node.left);

System.***out***.print(node.data + " ");

inOrder(node.right);

}

**public** **void** preOrder(Node node) {

**if** (node == **null**)

**return**;

System.***out***.print(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

**public** **void** postOrder(Node node) {

**if** (node == **null**)

**return**;

postOrder(node.left);

postOrder(node.right);

System.***out***.print(node.data + " ");

}

**public** **boolean** ifNodePresent(Node node, **int** value) {

**boolean** isPresent = **false**;

**if** (node == **null**) {

**return** **false**;

}

**while** (node != **null**) {

**if** (value < node.data) {

node = node.left;

} **else** **if** (value > node.data) {

node = node.right;

} **else** {

isPresent = **true**;

**break**;

}

}

**return** isPresent;

}

**public** Node getParentNode(Node node, **int** value) {

**if** (node == **null**)

**return** **null**;

Node getParent = **null**;

**while** (node != **null**) {

**if** (value < node.data) {

getParent = node;

node = node.left;

} **else** **if** (value > node.data) {

getParent = node;

node = node.right;

} **else** {

**break**;

}

}

**if** (node != **null**)

**return** getParent;

**else**

**return** **null**;

}

**public** Node getSiblingNode(Node node, **int** value) {

**if** (node == **null**) {

**return** **null**;

}

Node parentNode = **null**;

**while** (node != **null**) {

**if** (value < node.data) {

parentNode = node;

node = node.left;

} **else** **if** (value > node.data) {

parentNode = node;

node = node.right;

} **else** {

**break**;

}

}

**if** (parentNode.left!=**null** && value == parentNode.left.data)

**return** parentNode.right;

**else** **if** (parentNode.right!=**null** && value == parentNode.right.data)

**return** parentNode.left;

**else**

**return** **null**;

}

}

**package** bst;

**public** **class** BinarySearchTreeTester {

**public** **static** **void** main(String[] args) {

BinarySearchTree binarySearchTree = **new** BinarySearchTree();

Node root = **null**;

// Insert element in tree

// 8,3,6,10,4,7,1,14,13

root = binarySearchTree.insert(root, 8);

root = binarySearchTree.insert(root, 3);

root = binarySearchTree.insert(root, 6);

root = binarySearchTree.insert(root, 10);

root = binarySearchTree.insert(root, 4);

root = binarySearchTree.insert(root, 7);

root = binarySearchTree.insert(root, 1);

root = binarySearchTree.insert(root, 14);

root = binarySearchTree.insert(root, 13);

Node siblingNode = binarySearchTree.getSiblingNode(root, 14);

**if** (siblingNode != **null**) {

System.***out***.println(siblingNode.data);

} **else** {

System.***out***.println("Sibling Node does not exist");

}

}

}

// Binary Search Tree

// 8

// 3 10

// 1 6 14

// 4 7 13