**Practical 15**

**Name: Shantanu Sethi**

**Roll no. 163**

**Aim: To implement different operation on BST**

**Objectives:**

1. To implement Insert, search ,trees traversal ( Inorder,preorder,postorder) , count, largest and smallest operation on BST

**Theory:**

For every node X, all the keys in its left subtree are smaller than the key value in X, and all the keys in its right subtree are larger than the key value in X

**Binary Tree Traversal**

1.**Pre order**

* Visit the node.
* Traverse the left subtree in preorder
* Traverse the right subtree in preorder

Algorithm preOrder (root <tree ptr>)

Pre: root points to root of the tree

Return : Traverses the tree in preorder

1.if (root <>NULL)

process(root->data)

preOrder(root->left)

preOrder(root->right)

2.return

**2.In order:**

* Traverse the left subtree in inorder.
* Visit the node.
* Traverse the right subtree in inorder.

Algorithm inOrder (root <tree ptr>)

Pre: root points to root of the tree

Return : Traverses the tree in inorder

1.if (root <>NULL)

inOrder(root->left)

process(root->data)

inOrder(root->right)

2.return

**3.Post order**

* Traverse the left subtree in postorder.
* Traverse the right subtree in postorder.
* Visit the node.

Algorithm postOrder (root <tree ptr>)

Pre: root points to root of the tree

Return : Traverses the tree in postorder

1.if (root <>NULL)

postOrder(root->left)

postOrder(root->right)

process(root->data)

2.return

**Insert in BST:**

Algorithm recInsert (root <tree ptr>, new<pointer>)

Pre: root points to root of the tree

Return : new node inserted in tree

1.if (root == NULL)

root = new

root->left = NULL

root->right = NULL

return

2.else

if ( new->data < root->data)

.recInsert (root->left, new)

else

recInsert (root->right, new)

**Search in BST:**

Algorithm recSearch (root <tree ptr>, target<keyType>)

Pre: root points to root of the tree

Return : Node address where key is found, NULL if not found

1.if (root == NULL)

return NULL

2.if (target < root->key)

return recSearch (root->left, target)

3.else if (target > root->key)

return recSearch (root->right, target)

4.else

return root

**TREE-MINIMUM (x)**

while x->left <> NULL do

x = x->left

return x

**TREE-MAXIMUM (x)**

while x->right<> NULL do

x = x->right

return x

**Program:**

**package** abhinav;

**import** java.util.Scanner;

**public** **class** BST {

**class** Node {

**int** data;

Node left, right;

**public** Node(**int** data) {

**this**.data = data;

left = right = **null**;

}

}

**private** Node root;

**public** BST() {

root = **null**;

}

**public** **void** insert(**int** data) {

root = insertRec(root, data);

}

**private** Node insertRec(Node root, **int** data) {

**if** (root == **null**) {

root = **new** Node(data);

**return** root;

}

**if** (data < root.data) {

root.left = insertRec(root.left, data);

} **else** **if** (data > root.data) {

root.right = insertRec(root.right, data);

}

**return** root;

}

**public** **void** inorder() {

inorderRec(root);

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

}

**private** **void** inorderRec(Node root) {

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

inorderRec(root.left);

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

inorderRec(root.right);

}

}

**public** **void** preorder() {

preorderRec(root);

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

}

**private** **void** preorderRec(Node root) {

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

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

preorderRec(root.left);

preorderRec(root.right);

}

}

**public** **void** postorder() {

postorderRec(root);

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

}

**private** **void** postorderRec(Node root) {

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

postorderRec(root.left);

postorderRec(root.right);

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

}

}

**public** **void** findLargest() {

**if** (root == **null**) {

System.***out***.println("Tree is empty");

**return**;

}

Node largest = root;

**while** (largest.right != **null**) {

largest = largest.right;

}

System.***out***.println("Largest node is: " + largest.data);

}

**public** **void** findSmallest() {

**if** (root == **null**) {

System.***out***.println("Tree is empty");

**return**;

}

Node smallest = root;

**while** (smallest.left != **null**) {

smallest = smallest.left;

}

System.***out***.println("Smallest node is: " + smallest.data);

}

**public** **int** countNodes() {

**return** countNodesRec(root);

}

**private** **int** countNodesRec(Node root) {

**if** (root == **null**) {

**return** 0;

}

**return** 1 + countNodesRec(root.left) + countNodesRec(root.right);

}

**public** **void** search(**int** value) {

**if** (searchRec(root, value)) {

System.***out***.println("Element " + value + " is found in the tree.");

} **else** {

System.***out***.println("Element " + value + " is not found in the tree.");

}

}

**private** **boolean** searchRec(Node root, **int** value) {

**if** (root == **null**) {

**return** **false**;

}

**if** (root.data == value) {

**return** **true**;

}

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

**return** searchRec(root.left, value);

} **else** {

**return** searchRec(root.right, value);

}

}

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

System.***out***.println("184-ABHINAV SINGH");

BST tree = **new** BST();

Scanner sc = **new** Scanner(System.***in***);

**while** (**true**) {

System.***out***.println("1: Insert 2: Inorder 3: Preorder 4: Postorder 5: Find Largest 6: Find Smallest 7: Count Nodes 8: Search 9: Exit");

System.***out***.print("Enter your choice: ");

**int** choice = sc.nextInt();

**switch** (choice) {

**case** 1:

System.***out***.print("Enter value to insert: ");

**int** value = sc.nextInt();

tree.insert(value);

**break**;

**case** 2:

System.***out***.print("Inorder traversal: ");

tree.inorder();

**break**;

**case** 3:

System.***out***.print("Preorder traversal: ");

tree.preorder();

**break**;

**case** 4:

System.***out***.print("Postorder traversal: ");

tree.postorder();

**break**;

**case** 5:

tree.findLargest();

**break**;

**case** 6:

tree.findSmallest();

**break**;

**case** 7:

System.***out***.println("Total number of nodes: " + tree.countNodes());

**break**;

**case** 8:

System.***out***.print("Enter value to search: ");

**int** searchValue = sc.nextInt();

tree.search(searchValue);

**break**;

**case** 9:

System.***out***.println("Exiting program.");

sc.close();

System.*exit*(0);

**default**:

System.***out***.println("Invalid choice. Please try again.");

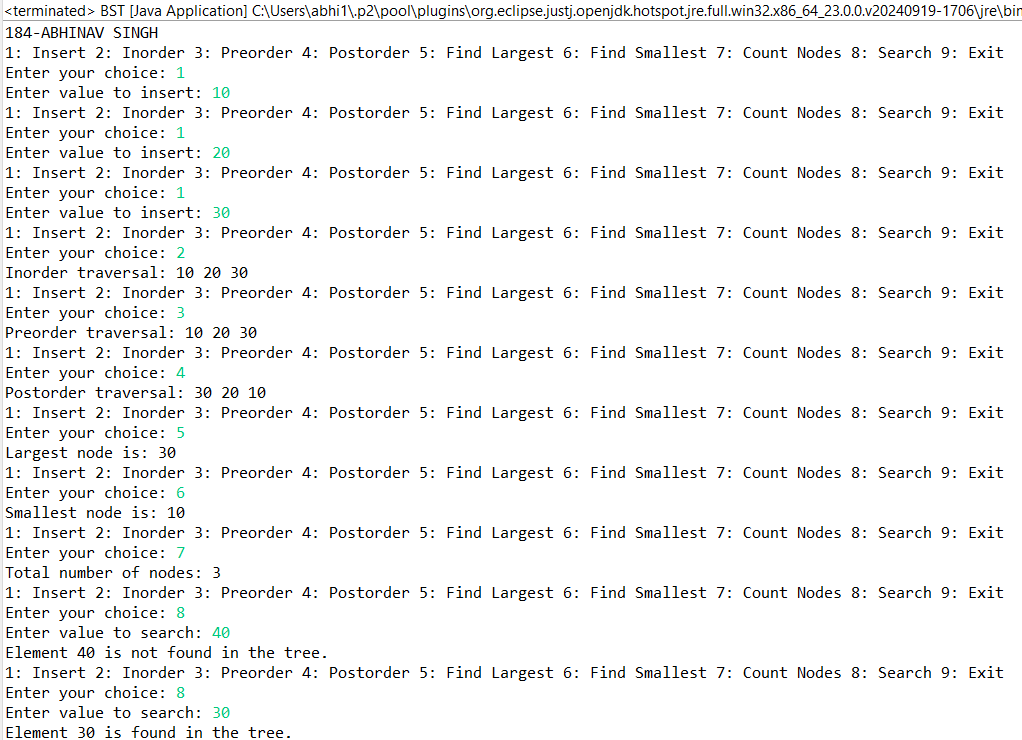
}

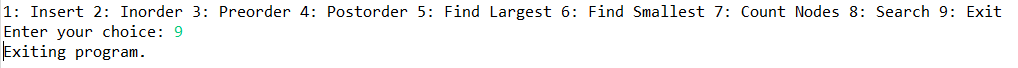
}

}

}

**OutPut:**





**Conclusion**: Successfully implemented different operations on BST.