**Algoma University**

**School of Computer Science and Technology**

**COSC 2007 Data Structures II (Lab)**

**Assignment 6**

**Date of Lab: Feb26, 2024**

Create a binary search tree (BST) to hold integer data and perform the following operations on it:

(a) Insertion of an integer into BST (no duplicates) -d

(b) Deletion of a given integer -d

(c) Find a given integer in BST -d

(d) Find minimum element in BST -d

(d) Display the integers in ascending order -d

Trace the tree using the diagrammatic representation of the tree.

class Node {

int key;

Node left, right;

public Node(int item) {

key = item;

left = right = null;

}

}

class BinarySearchTree {

Node root;

BinarySearchTree() {

root = null;

}

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

if (root == null) {

root = new Node(key);

return root;

}

if (key < root.key) {

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

} else if (key > root.key) {

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

}

return root;

}

void deleteKey(int key) {

root = deleteRec(root, key);

}

Node deleteRec(Node root, int key) {

if (root == null) return root;

if (key < root.key) {

root.left = deleteRec(root.left, key);

} else if (key > root.key) {

root.right = deleteRec(root.right, key);

} else {

if (root.left == null)

return root.right;

else if (root.right == null)

return root.left;

root.key = minValue(root.right);

root.right = deleteRec(root.right, root.key);

}

return root;

}

int minValue(Node root) {

int minValue = root.key;

while (root.left != null) {

minValue = root.left.key;

root = root.left;

}

return minValue;

}

boolean search(int key) {

root = searchRec(root, key);

return root != null;

}

Node searchRec(Node root, int key) {

if (root == null || root.key == key) return root;

if (root.key > key)

return searchRec(root.left, key);

return searchRec(root.right, key);

}

void inorder() {

inorderRec(root);

}

void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

System.out.print(root.key + " ");

inorderRec(root.right);

}

}

void printTree(Node root, int space) {

if (root == null) return;

space += 10;

printTree(root.right, space);

System.out.println();

for (int i = 10; i < space; i++) {

System.out.print(" ");

}

System.out.println(root.key + "\n");

printTree(root.left, space);

}

void printTree() {

printTree(root, 0);

}

}

public class Main {

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

bst.insert(50);

bst.insert(30);

bst.insert(20);

bst.insert(40);

bst.insert(70);

bst.insert(60);

bst.insert(80);

System.out.println("Inorder traversal (ascending order):");

bst.inorder();

System.out.println("\n");

bst.deleteKey(20);

System.out.println("Inorder traversal after deleting 20:");

bst.inorder();

System.out.println("\n");

if (bst.search(30)) {

System.out.println("Element 30 found in the BST.");

} else {

System.out.println("Element 30 not found in the BST.");

}

System.out.println("Minimum element in BST: " + bst.minValue(bst.root));

System.out.println("\nDiagrammatic Representation of the BST:");

bst.printTree();

}

}