**Lab 7 Report**

**CS303L Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

* Insert elements in a Binary Search Tree
* Search an element in a Binary Search Tree
* In-Order-Traversal of the BST to print elements in a sorted order
* Manage a Binary Search Tree through both insertions and deletions

**In-class Assignment:**

1. A Binary Search Tree written in Java is provided (see below).

a. Implement a method Insert (T data) in BinarySearchTree.java to insert a node having value of key = data in a Binary Search Tree.

b. Implement In-Order-Traversal () in BinarySearchTree.java that prints the elements of the BST in an in-order format. See the algorithm presented below. If your insertion of elements in the BST is correct, then the In-Order-Traversal output should print the elements in a sorted order.

1. Write a driver program to test the BinarySearchTree class.

**Source code for In-class Assignment:**

package Lab7;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class BinaryTree<T extends Comparable<T>> {

public static void main(String[] args) {

int[] array = {5, 2, 3, 7, 8, 6, 10};

BinaryTree<Integer> tree = new BinaryTree<Integer>(null);

for(int i = 0; i <= array.length -1; i++){

BinaryTreeNode<Integer> node = new BinaryTreeNode<Integer>(array[i]);

tree.insert(tree, node);

}

tree.inOrderTreeWalk(tree.root);

}

protected BinaryTreeNode<T> root; // the root of the tree private

private BinaryTreeNode<T> cursor; // the current node

/\*\*  \* Constructor for initializing a tree with node  \* being set as the root of the tree. \* @param node  \*/

public BinaryTree(BinaryTreeNode<T> node) {

root = node;

}

/\*\* \* Moves the cursor to the root. \*/

public void toRoot() {

cursor = root;

}

/\*\* \* Returns the cursor node. \* @return cursor \*/

public BinaryTreeNode<T> getCursor() {

return cursor;

}

/\*\*

\* \* Sets the root to the provided node. \* ONLY USE IN THE DELETE METHOD

\* \* @param node

\*/

public void setRoot(BinaryTreeNode<T> node) {

root = node;

}

/\*\*

\* \* Checks if the tree node has a left child node \* @return true if left

\* child exists else false

\*/

public boolean hasLeftChild() {

return cursor.getLeft() != null;

}

/\*\*

\* \* Checks if the tree node has a right child node \* @return true if right

\* child exists else false

\*/

public boolean hasRightChild() {

return cursor.getRight() != null;

}

/\*\* \* Move the cursor to the left child \*/

public void toLeftChild() {

cursor = cursor.getLeft();

}

/\*\* \* Move the cursor to the right child \*/

public void toRightChild() {

cursor = cursor.getRight();

}

/\*\* \* @return height of the tree \*/

public int height() {

if (root != null) {

return root.height();

} else {

return 0;

}

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*\*/

public String toString() {

if (root != null) {

return root.toStringPreOrder(".");

} else {

return "";

}

}

public void insert(BinaryTree<T> T, BinaryTreeNode<T> z){

BinaryTreeNode<T> y = null;

BinaryTreeNode<T> x = T.root;

while(x != null) {

y = x;

if (z.getData().compareTo(x.getData()) < 0) {

x = x.getLeft();

}

else x = x.getRight();

}

if(y == null) {

T.root = z;

}

else if(z.getData().compareTo(y.getData()) < 0) {

y.setLeft(z);

}

else y.setRight(z);

}

public void inOrderTreeWalk(BinaryTreeNode<T> x) {

if(x != null) {

inOrderTreeWalk(x.getLeft());

System.out.println(x.getData().toString());

inOrderTreeWalk(x.getRight());

}

}

**Output for In-class Assignment:**

2

3

5

6

7

8

10

**Homework Assignment:**

1. Implement Search (T data) in BinarySearchTree.java to search for a node having value of key = data in a Binary Search Tree.
2. Implement Delete (T data) in BinarySearchTree.java to search for a node having value of key = data in a Binary Search Tree.
3. Exercise your Binary Search Tree by selectively inserting and removing items, and using In-Order-Traversal () at each step to show the tree’s contents at each step. At the maximum, your tree should contain at least ten objects.

**Source code for Homework Assignment:**

package Lab7;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class BinaryTree<T extends Comparable<T>> {

public static void main(String[] args) throws FileNotFoundException {

int[] array = {2, 5, 7, 3, 8, 6, 10, 1, 4, 9};

BinaryTree<Integer> tree = new BinaryTree<Integer>(null);

BinaryTreeNode<Integer> node1 = new BinaryTreeNode<Integer>(array[0]);

System.out.println("Inserted new node to binary tree.");

tree.insert(tree, node1);

tree.inOrderTreeWalk(tree.root);

BinaryTreeNode<Integer> node2 = new BinaryTreeNode<Integer>(array[1]);

System.out.println("\n" + "Inserted new node to binary tree.");

tree.insert(tree, node2);

tree.inOrderTreeWalk(tree.root);

BinaryTreeNode<Integer> node3 = new BinaryTreeNode<Integer>(array[2]);

System.out.println("\n" + "Inserted new node to binary tree.");

tree.insert(tree, node3);

tree.inOrderTreeWalk(tree.root);

tree.delete(tree, node1);

System.out.println("\n" + "Deleted first node from binary tree.");

tree.inOrderTreeWalk(tree.root);

System.out.println("\n" + "Searching for " + node3.getData() );

boolean s = tree.treeSearch(7, node3);

if(s == true){

System.out.println("The node you searched for was found.");

}

else System.out.println("The node you searched for was not found.");

}

protected BinaryTreeNode<T> root; // the root of the tree private

private BinaryTreeNode<T> cursor; // the current node

/\*\*  \* Constructor for initializing a tree with node  \* being set as the root of the tree. \* @param node  \*/

public BinaryTree(BinaryTreeNode<T> node) {

root = node;

}

/\*\* \* Moves the cursor to the root. \*/

public void toRoot() {

cursor = root;

}

/\*\* \* Returns the cursor node. \* @return cursor \*/

public BinaryTreeNode<T> getCursor() {

return cursor;

}

/\*\*

\* \* Sets the root to the provided node. \* ONLY USE IN THE DELETE METHOD

\* \* @param node

\*/

public void setRoot(BinaryTreeNode<T> node) {

root = node;

}

/\*\*

\* \* Checks if the tree node has a left child node \* @return true if left

\* child exists else false

\*/

public boolean hasLeftChild() {

return cursor.getLeft() != null;

}

/\*\*

\* \* Checks if the tree node has a right child node \* @return true if right

\* child exists else false

\*/

public boolean hasRightChild() {

return cursor.getRight() != null;

}

/\*\* \* Move the cursor to the left child \*/

public void toLeftChild() {

cursor = cursor.getLeft();

}

/\*\* \* Move the cursor to the right child \*/

public void toRightChild() {

cursor = cursor.getRight();

}

/\*\* \* @return height of the tree \*/

public int height() {

if (root != null) {

return root.height();

} else {

return 0;

}

}

public String toString() {

if (root != null) {

return root.toStringPreOrder(".");

} else {

return "";

}

}

public void insert(BinaryTree<T> T, BinaryTreeNode<T> z){

BinaryTreeNode<T> y = null;

BinaryTreeNode<T> x = T.root;

while(x != null) {

y = x;

if (z.getData().compareTo(x.getData()) < 0) {

x = x.getLeft();

}

else x = x.getRight();

}

if(y == null) {

T.root = z;

}

else if(z.getData().compareTo(y.getData()) < 0) {

y.setLeft(z);

}

else y.setRight(z);

}

public void inOrderTreeWalk(BinaryTreeNode<T> x) {

if(x != null) {

inOrderTreeWalk(x.getLeft());

System.out.println(x.getData().toString());

inOrderTreeWalk(x.getRight());

}

}

public boolean treeSearch(T k, BinaryTreeNode<T> x) {

while(x != null) {

if (k.compareTo(x.getData()) == 0) {

return true;

}

if(k.compareTo(x.getData()) < 0) {

x.setLeft(x);

}

else

x.setRight(x);

}

return false;

}

public void delete(BinaryTree<T> T,BinaryTreeNode<T> z) {

BinaryTreeNode<T> y = null;

if(z.getLeft() == null) {

transplant(T, z, z.getRight());

}

else if(z.getRight() == null) {

transplant(T, z, z.getLeft());

}

else {

y = treeMinimum(z.getRight());

if(y.getParent() != z){

transplant(T,y, y.getRight());

y.setRight(z.getRight());

y.getLeft().getParent().setParent(y);

}

transplant(T, z, y);

y.setLeft(z.getLeft());

y.getRight().getParent().setParent(y);

}

}

public BinaryTreeNode<T> treeMinimum(BinaryTreeNode<T> x) {

while(x.getRight() != null){

x = x.getRight();

}

return x;

}

public void transplant(BinaryTree<T> T,BinaryTreeNode<T> u, BinaryTreeNode<T> v) {

if(u.getParent() == null){

T.root = v;

}

else if(u == u.getParent().getLeft()) {

u.getParent().setLeft(v);

}

else {

u.getParent().setRight(v);;

}

if(v != null) {

v.setParent(u.getParent());

}

}

}

**Output for Homework Assignment:**

Inserted new node to binary tree.

2

Inserted new node to binary tree.

2

5

Inserted new node to binary tree.

2

5

7

Deleted first node from binary tree.

5

7

Searching for 7

The node you searched for was found.