Christopher Aboshear

Data Structures 3319

Successfully completed “A” option

**Input\_File**

I McManus **295**-**1492**

I Figueroa **291**-**1864**

I Wimalasekara **295**-**1601**

I Parker **293**-**6122**

I Abo-Shear **295**-**1882**

I Syed **291**-**7890**

I Khadka **294**-**8075**

I Musco **584**-**3622**

FI Musco

FR Musco

FI Penton

FR Penton

TI Figueroa

I Bolen **294**-**1568**

I Skinner **294**-**1882**

I Krischke **295**-**6622**

TI McManus

TPI

D Parker

D McManus

D Figueroa

I Sumdin **294**-**1666**

I Bolen **295**-**1882**

TI Musco

TR

TPT

TOI

TOR

**Output**

Inserting McManus into tree

Inserting Figueroa into tree

Inserting Wimalasekara into tree

Inserting Parker into tree

Inserting Abo-Shear into tree

Inserting Syed into tree

Inserting Khadka into tree

Inserting Musco into tree

Searching for Musco iterative way

Musco found

Searching for Musco recursive way

Musco found

Searching for Penton iterative way

Penton not found

Searching for Penton recursive way

Penton not found

**Inorder Traversal of tree starting at Figueroa**

Khadka 294-8075

McManus 295-1492

Musco 584-3622

Parker 293-6122

Syed 291-7890

Wimalasekara 295-1601

Abo-Shear 295-1882

Inserting Bolen into tree

Inserting Skinner into tree

Inserting Krischke into tree

**Inorder Traversal of tree starting at McManus**

Musco 584-3622

Parker 293-6122

Skinner 294-1882

Syed 291-7890

Wimalasekara 295-1601

Abo-Shear 295-1882

Bolen 294-1568

Figueroa 291-1864

Khadka 294-8075

Krischke 295-6622

**Traversing Tree in Preorder Iterative**

McManus 295-1492

Figueroa 291-1864

Abo-Shear 295-1882

Bolen 294-1568

Khadka 294-8075

Krischke 295-6622

Wimalasekara 295-1601

Parker 293-6122

Musco 584-3622

Syed 291-7890

Skinner 294-1882

Deleting Parker from tree

Deleting McManus from tree

Deleting Figueroa from tree

Inserting Sumdin into tree

Inserting Bolen into tree

**Inorder Traversal of tree starting at Musco**

Skinner 294-1882

Sumdin 294-1666

Syed 291-7890

Wimalasekara 295-1601

Abo-Shear 295-1882

Bolen 294-1568

Khadka 294-8075

Krischke 295-6622

**Traversing Tree In Reverse InOrder**

Wimalasekara 295-1601

Syed 291-7890

Sumdin 294-1666

Skinner 294-1882

Musco 584-3622

Krischke 295-6622

Khadka 294-8075

Bolen 294-1568

Abo-Shear 295-1882

**Traversing Tree in Preorder using Threads**

Musco 584-3622

Khadka 294-8075

Abo-Shear 295-1882

Bolen 294-1568

Krischke 295-6622

Wimalasekara 295-1601

Skinner 294-1882

Syed 291-7890

Sumdin 294-1666

**Traversing Tree In PostOrder Iterative**

Bolen 294-1568

Abo-Shear 295-1882

Krischke 295-6622

Khadka 294-8075

Sumdin 294-1666

Syed 291-7890

Skinner 294-1882

Wimalasekara 295-1601

Musco 584-3622

**Traversing Tree In PostOrder Recursive**

Bolen 294-1568

Abo-Shear 295-1882

Krischke 295-6622

Khadka 294-8075

Sumdin 294-1666

Syed 291-7890

Skinner 294-1882

Wimalasekara 295-1601

Musco 584-3622

**BinarySearchTree.java**

**package** binaryTree;

**import** **java.util.LinkedList**;

**import** **java.util.Stack**;

**public** **class** **BinarySearchTree**<T **extends** Comparable<T>> {

**private** Node<T> root;

**public** Node<T> **getRoot**() {

**return** root;

}

**public** **BinarySearchTree**() {

root = **null**;

}

**public** **void** **insertBinarySearchTree**(T customer) {

Node<T> treeNode = **new** Node<T>(customer);

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

root = treeNode;

}**else** {

Node<T> temp = root;

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

**if**(temp.getInfo().compareTo(customer)>**0**) {

**if**(temp.isLeftTag()) {

temp = temp.getLeftLink();

}**else** {

temp.setLeftTag(**true**);

treeNode.setLeftLink(temp.getLeftLink());

temp.setLeftLink(treeNode);

treeNode.setRightTag(**false**);

treeNode.setRightLink(temp);

**break**;

}

}**else** **if**(temp.getInfo().compareTo(customer)<**0**) {

**if**(temp.isRightTag()) {

temp = temp.getRightLink();

}**else** {

temp.setRightTag(**true**);

treeNode.setRightLink(temp.getRightLink());

temp.setRightLink(treeNode);

treeNode.setLeftTag(**false**);

treeNode.setLeftLink(temp);

**break**;

}

}**else** {

**break**;

}

}

}

}

**public** Node<T> **findCustomerIterative**(T c) {

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

**return** **null**;

}**else** {

Node<T> temp = root;

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

**if**(temp.getInfo().compareTo(c)==**0**) {

**return** temp;

}**else** **if**(temp.getInfo().compareTo(c)>**0**) {

**if**(temp.isLeftTag()) {

temp = temp.getLeftLink();

}**else** {

**return** **null**;

}

}**else** {

**if**(temp.isRightTag()) {

temp = temp.getRightLink();

}**else** {

**return** **null**;

}

}

}

}

}

**public** Node<T> **findCustomerRecursive**(Node<T> temp,T c) {

**if**(temp==**null**) {

**return** **null**;

}**else** **if**(temp.getInfo().compareTo(c)==**0**) {

**return** temp;

}**else** **if**(temp.getInfo().compareTo(c)>**0**) {

**if**(temp.isLeftTag()) {

**return** **findCustomerRecursive**(temp.getLeftLink(),c);

}**else** {

**return** **null**;

}

}**else** {

**if**(temp.isRightTag()) {

**return** **findCustomerRecursive**(temp.getRightLink(),c);

}**else** {

**return** **null**;

}

}

}

**public** Node<T> **inOrderSuccessor**(Node<T> temp) {

Node<T> q = temp.getRightLink();

**if**(!temp.isRightTag()) {

**return** q;

}**else** {

**while**(q.isLeftTag()) {

q = q.getLeftLink();

}

}

**return** q;

}

**public** **void** **preOrderTraversalIterative**() {

Node<T> temp = root;

Stack<Node<T>> s = **new** Stack<>();

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

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

System.out.println(temp.print());

**if**(temp.isLeftTag() || temp.isRightTag()) {

s.push(temp);

**if**(temp.isLeftTag()) {

temp = temp.getLeftLink();

}**else** {

temp = **null**;

}

}**else** {

temp = **null**;

}

}**else** {

**if**(s.empty()) {

**break**;

}**else** {

temp = s.pop();

**if**(temp.isRightTag()) {

temp = temp.getRightLink();

}**else** {

temp = **null**;

}

}

}

}

}

**public** Node<T> **getFirstInOrder**() {

Node<T> prev,curr;

prev = curr = root;

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

prev = curr;

curr = inOrderPredecessor(curr);

**if**(curr.getInfo().compareTo(root.getInfo())==**0**) {

**break**;

}

}

**return** prev;

}

**private** Node<T> **inOrderPredecessor**(Node<T> curr) {

Node<T> q = curr.getLeftLink();

**if**(curr.isLeftTag()) {

**while**(q.isRightTag()) {

q = q.getRightLink();

}

}

**return** q;

}

**public** Node<T> **getLastInOrder**() {

Node<T> prev,curr;

prev = curr = root;

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

prev = curr;

curr = inOrderSuccessor(curr);

**if**(curr.getInfo().compareTo(root.getInfo())==**0**) {

**break**;

}

}

**return** prev;

}

**public** **void** **reverseInOrder**(Node<T> temp) {

**if**(!temp.isLeftTag() && !temp.isRightTag()) {

System.out.println(temp.print());

**return**;

}

**if**(temp.isRightTag()) {

reverseInOrder(temp.getRightLink());

}

System.out.println(temp.print());

**if**(temp.isLeftTag()) {

reverseInOrder(temp.getLeftLink());

}

}

**public** Node<T> **findParent**(T c) {

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

**return** **null**;

}**else** {

Node<T> curr = root;

Node<T> prev = root;

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

**if**(curr.getInfo().compareTo(c)==**0**) {

**return** prev;

}**else** **if**(curr.getInfo().compareTo(c)>**0**) {

**if**(curr.isLeftTag()) {

prev = curr;

curr = curr.getLeftLink();

}**else** {

**return** **null**;

}

}**else** {

**if**(curr.isRightTag()) {

prev = curr;

curr = curr.getRightLink();

}**else** {

**return** **null**;

}

}

}

}

}

**public** **void** **deleteRandomNode**(Node<T> q) {

**if**(!q.isLeftTag() && !q.isRightTag()) {

**if**(q.getRightLink().getLeftLink().getInfo().compareTo(q.getInfo())==**0**) {

q.getRightLink().setLeftTag(**false**);

q.getRightLink().setLeftLink(q.getLeftLink());

}**else** **if**(q.getLeftLink().getRightLink().getInfo().compareTo(q.getInfo())==**0**){

q.getLeftLink().setRightTag(**false**);

q.getLeftLink().setRightLink(q.getRightLink());

}

}**else** **if**(!q.isLeftTag() && q.isRightTag()) {

Node<T> parent = findParent(q.getInfo());

Node<T> predecessor = inOrderPredecessor(q);

Node<T> successor = inOrderSuccessor(q);

**if**(parent.getLeftLink().getInfo().compareTo(q.getInfo())==**0**) {

parent.setLeftLink(q.getRightLink());

}**else** {

parent.setRightLink(q.getRightLink());

}

**if**(successor.getLeftLink().getInfo().compareTo(q.getInfo())==**0**) {

successor.setLeftLink(predecessor);

}**else** {

successor.setRightLink(predecessor);

}

}**else** **if**(!q.isRightTag() && q.isLeftTag()) {

Node<T> parent = findParent(q.getInfo());

Node<T> predecessor = inOrderPredecessor(q);

Node<T> successor = inOrderSuccessor(q);

**if**(parent.getLeftLink().getInfo().compareTo(q.getInfo())==**0**) {

parent.setLeftLink(q.getLeftLink());

}**else** {

parent.setRightLink(q.getLeftLink());

}

**if**(successor.getLeftLink().getInfo().compareTo(q.getInfo())==**0**) {

successor.setLeftLink(predecessor);

}**else** {

successor.setRightLink(predecessor);

}

}**else** {

Node<T> temp = inOrderSuccessor(q);

T c = q.getInfo();

**if**(!temp.isRightTag() && !temp.isLeftTag()) {

q.setInfo(temp.getInfo());

temp.setInfo(c);

**if**(temp.getRightLink().getLeftLink().getInfo().compareTo(temp.getInfo())==**0**) {

temp.getRightLink().setLeftTag(**false**);

temp.getRightLink().setLeftLink(temp.getLeftLink());

}**else** **if**(temp.getLeftLink().getRightLink().getInfo().compareTo(temp.getInfo())==**0**){

temp.getLeftLink().setRightTag(**false**);

temp.getLeftLink().setRightLink(temp.getRightLink());

}

}**else** **if**(!temp.isLeftTag() && temp.isRightTag()) {

Node<T> parent = findParent(temp.getInfo());

Node<T> predecessor = inOrderPredecessor(temp);

Node<T> successor = inOrderSuccessor(temp);

**if**(parent.getLeftLink().getInfo().compareTo(temp.getInfo())==**0**) {

parent.setLeftLink(temp.getRightLink());

}**else** {

parent.setRightLink(temp.getRightLink());

}

**if**(successor.getLeftLink().getInfo().compareTo(temp.getInfo())==**0**) {

successor.setLeftLink(predecessor);

}**else** {

successor.setRightLink(predecessor);

}

q.setInfo(temp.getInfo());

temp.setInfo(c);

}**else** **if**(!temp.isRightTag() && temp.isLeftTag()) {

Node<T> parent = findParent(temp.getInfo());

Node<T> predecessor = inOrderPredecessor(temp);

Node<T> successor = inOrderSuccessor(temp);

**if**(parent.getLeftLink().getInfo().compareTo(temp.getInfo())==**0**) {

parent.setLeftLink(temp.getLeftLink());

}**else** {

parent.setRightLink(temp.getLeftLink());

}

**if**(successor.getLeftLink().getInfo().compareTo(temp.getInfo())==**0**) {

successor.setLeftLink(predecessor);

}**else** {

successor.setRightLink(predecessor);

}

q.setInfo(temp.getInfo());

temp.setInfo(c);

}

}

}

**public** Node<T> **preOrderSuccessor**(Node<T> p) {

Node<T> q;

**if**(p.isLeftTag()) {

q = p.getLeftLink();

}**else** {

q = p;

**while**(!q.isRightTag()) {

q = q.getRightLink();

}

q = q.getRightLink();

}

**return** q;

}

**public** **void** **preOrderThread**() {

Node<T> temp = root;

LinkedList<Node<T>> list = **new** LinkedList<>();

System.out.println(temp.print());

list.add(temp);

temp = preOrderSuccessor(temp);

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

System.out.println(temp.print());

temp = preOrderSuccessor(temp);

**if**(list.contains(temp)) {

**return**;

}**else** {

list.add(temp);

}

}

}

**public** Node<T> **postOrderPredecessor**(Node<T> p) {

Node<T> q = p;

**if**(p.isRightTag()) {

q = p.getRightLink();

}**else** {

**while**(!q.isLeftTag()) {

q = q.getLeftLink();

}

q = q.getLeftLink();

}

**return** q;

}

**public** **void** **postOrderIterative**() {

Node<T> temp = root;

Stack<Node<T>> s = **new** Stack<>();

LinkedList<Node<T>> ll = **new** LinkedList<>();

s.push(temp);

ll.add(temp);

temp = postOrderPredecessor(temp);

s.push(temp);

**boolean** shouldStop = **false**;

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

//System.out.println(temp.getInfo().getName() + " " + temp.getInfo().getPhoneNumber());

temp = postOrderPredecessor(temp);

**if**(ll.contains(temp)) {

shouldStop = **true**;

}

**if**(shouldStop) {

**break**;

}

ll.add(temp);

s.push(temp);

}

**int** a = s.size();

**for**(**int** i=**0**;i<a;i++) {

Node<T> t = s.pop();

System.out.println(t.print());

}

}

**public** **void** **postOrderRecursive**(Node<T> temp) {

**if**(!temp.isLeftTag() && !temp.isRightTag()) {

System.out.println(temp.print());

**return**;

}

**if**(temp.isLeftTag()) {

postOrderRecursive(temp.getLeftLink());

}

**if**(temp.isRightTag()) {

postOrderRecursive(temp.getRightLink());

}

System.out.println(temp.print());

}

}

**Driver.java**

**package** binaryTree;

**import** **java.io.File**;

**import** **java.io.FileNotFoundException**;

**import** **java.util.Scanner**;

**public** **class** **Driver** {

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

System.out.println("Please enter input file name: ");

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

String inputFileName = sc.nextLine();

File inputFile = **new** File("C:\\"+inputFileName);

Scanner fs = **new** Scanner(inputFile);

BinarySearchTree<Customer> binarySearchTree = **new** BinarySearchTree<>();

**while**(fs.hasNextLine()) {

String line = fs.nextLine();

//System.out.println("line=" + line);

**if**(line.charAt(**0**)=='I') {

String[] ar = line.split(" ");

String name = ar[**1**];

String phoneNumber = ar[**2**];

Customer c = **new** Customer(name,phoneNumber);

System.out.println("Inserting " + name + " into tree");

binarySearchTree.insertBinarySearchTree(c);

}**else** **if**(line.charAt(**0**)=='F') {

String[] ar = line.split(" ");

String name = ar[**1**];

**if**(line.charAt(**1**)=='I') {

Customer c = **new** Customer(name);

Node<Customer> temp = binarySearchTree.findCustomerIterative(c);

System.out.println("Searching for " + name + " iterative way");

**if**(temp==**null**) {

System.out.println(name + " not found");

}**else** {

System.out.println(name + " found");

}

}**else** {

Customer c = **new** Customer(name);

Node<Customer> temp = binarySearchTree.findCustomerRecursive(binarySearchTree.getRoot(), c);

System.out.println("Searching for " + name + " recursive way");

**if**(temp==**null**) {

System.out.println(name + " not found");

}**else** {

System.out.println(name + " found");

}

}

}**else** **if**(line.charAt(**0**)=='T') {

**if**(line.charAt(**1**)=='I') {

String[] ar = line.split(" ");

String name = ar[**1**];

Customer c = **new** Customer(name);

System.out.println("Inorder Traversal of tree starting at " + name);

Node<Customer> temp = binarySearchTree.findCustomerIterative(c);

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

Node<Customer> first = binarySearchTree.getFirstInOrder();

Node<Customer> last = binarySearchTree.getLastInOrder();

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

temp = binarySearchTree.inOrderSuccessor(temp);

**if**(last.getInfo().compareTo(temp.getInfo())==**0**) {

System.out.println(temp.getInfo().getName() + " " + temp.getInfo().getPhoneNumber());

**break**;

}

System.out.println(temp.getInfo().getName() + " " + temp.getInfo().getPhoneNumber());

//sc.nextLine();

}

System.out.println(first.getInfo().getName() + " " + first.getInfo().getPhoneNumber());

Customer c1 = **new** Customer(name);

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

first = binarySearchTree.inOrderSuccessor(first);

**if**(first.getInfo().compareTo(c1)==**0**) {

**break**;

}

System.out.println(first.getInfo().getName() + " " + first.getInfo().getPhoneNumber());

//sc.hasNextLine();

}

}

}**else** **if**(line.charAt(**1**)=='P'){

**if**(line.charAt(**2**)=='I') {

System.out.println("Traversing Tree in Preorder Iterative");

binarySearchTree.preOrderTraversalIterative();

}**else** {

System.out.println("Traversing Tree in Preorder using Threads");

binarySearchTree.preOrderThread();

}

}**else** **if**(line.charAt(**1**)=='R') {

System.out.println("Traversing Tree In Reverse InOrder");

binarySearchTree.reverseInOrder(binarySearchTree.getRoot());

}**else** {

**if**(line.charAt(**2**)=='I') {

System.out.println("Traversing Tree In PostOrder Iterative");

binarySearchTree.postOrderIterative();

}**else** {

System.out.println("Traversing Tree In PostOrder Recursive");

binarySearchTree.postOrderRecursive(binarySearchTree.getRoot());

}

}

}**else** **if**(line.charAt(**0**)=='D') {

String[] ar = line.split(" ");

String name = ar[**1**];

Customer c = **new** Customer(name);

System.out.println("Deleting " + name + " from tree");

Node<Customer> toDelete = binarySearchTree.findCustomerIterative(c);

binarySearchTree.deleteRandomNode(toDelete);

}

}

fs.close();

sc.close();

}

}

**Node.java**

**package** binaryTree;

**public** **class** **Node**<T **extends** Comparable<T>> {

**private** Node<T> leftLink, rightLink;

**private** **boolean** leftTag, rightTag;

**private** T info;

**public** **Node**() {

leftLink = rightLink = **this**;

leftTag = rightTag = **false**;

info = **null**;

}

**public** **Node**(T cust) {

leftLink = rightLink = **this**;

leftTag = rightTag = **false**;

info = cust;

}

**public** String **print**() {

**return** info.toString();

}

**public** Node<T> **getLeftLink**() {

**return** leftLink;

}

**public** **void** **setLeftLink**(Node<T> leftLink) {

**this**.leftLink = leftLink;

}

**public** Node<T> **getRightLink**() {

**return** rightLink;

}

**public** **void** **setRightLink**(Node<T> rightLink) {

**this**.rightLink = rightLink;

}

**public** **boolean** **isLeftTag**() {

**return** leftTag;

}

**public** **void** **setLeftTag**(**boolean** leftTag) {

**this**.leftTag = leftTag;

}

**public** **boolean** **isRightTag**() {

**return** rightTag;

}

**public** **void** **setRightTag**(**boolean** rightTag) {

**this**.rightTag = rightTag;

}

**public** T **getInfo**() {

**return** info;

}

**public** **void** **setInfo**(T info) {

**this**.info = info;

}

}

**Customer.java**

**package** binaryTree;

**public** **class** **Customer** **implements** Comparable<Customer>{

**private** String name, phoneNumber;

**public** **Customer**(String name, String phoneNumber) {

**this**.name = name;

**this**.phoneNumber = phoneNumber;

}

**public** **Customer**(String name) {

**this**.name = name;

}

**public** String **getName**() {

**return** name;

}

**public** **void** **setName**(String name) {

**this**.name = name;

}

**public** String **getPhoneNumber**() {

**return** phoneNumber;

}

**public** **void** **setPhoneNumber**(String phoneNumber) {

**this**.phoneNumber = phoneNumber;

}

**public** **int** **compareTo**(Customer c) {

**return** **this**.getName().compareTo(c.getName());

}

**public** String **toString**() {

**return** ""+getName() + " " + getPhoneNumber();

}

}