public class BinaryTree {

Node root;

public void insert(int key, String name) {

//create a node

Node newNode = new Node(key, name);

if (root == null) {

root = newNode;

} else {

Node focusNode = root;

Node parent;

while (true) {

//root

parent = focusNode;

//left or right side?

if (key < focusNode.key) {

focusNode = focusNode.left;

//left side

if (focusNode == null) {

parent.left = newNode;

return;

}

//right side

} else {

focusNode = focusNode.right;

if (focusNode == null) {

parent.right = newNode;

return;

}

}

}

}

}

//1. left child and left side, root, right side and right side (from smallest to biggest)

// root

public void inorderTraverse(Node focusNode) {

if (focusNode != null) {

inorderTraverse(focusNode.left);

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

inorderTraverse(focusNode.right);

}

}

//1. root, left child and left side, right child and right side

public void preorderTraverse(Node focusNode) {

if (focusNode != null) {

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

preorderTraverse(focusNode.left);

preorderTraverse(focusNode.right);

}

}

//1.left child and left side, 2. right child and right side, 3. root

public void postorderTraverse(Node focusNode) {

if (focusNode != null) {

postorderTraverse(focusNode.left);

postorderTraverse(focusNode.right);

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

}

}

public Node search(int key) {

Node focusNode = root;

while (focusNode.key != key) {

//looking left

if (key < focusNode.key) {

focusNode = focusNode.left;

} else {

//looking right

focusNode = focusNode.right;

}

if (focusNode == null)

return null;

}

return focusNode;

}

public boolean remove(int key) {

// Start at the top of the tree

Node focusNode = root;

Node parent = root;

boolean isItALeftChild = true;

// While key that we're looking for isn't found

while (focusNode.key != key) {

parent = focusNode;

// Searching left and focusing on left or right child

if (key < focusNode.key) {

isItALeftChild = true;

focusNode = focusNode.left;

} else {

isItALeftChild = false;

focusNode = focusNode.right;

}

if (focusNode == null)

return false;

}

// If Node doesn't have children delete it

if (focusNode.left == null && focusNode.right == null) {

// If root delete it

if (focusNode == root)

root = null;

//if left child delete it

else if (isItALeftChild)

parent.left = null;

//if right child delete it

else

parent.right = null;

}

// If no right child

else if (focusNode.right == null) {

if (focusNode == root)

root = focusNode.left;

// If focus Node was on the left of parent

// move the focus Nodes left child up to the

// parent node

else if (isItALeftChild)

parent.left = focusNode.left;

// Vice versa for the right child

else

parent.right = focusNode.left;

}

// If no left child

else if (focusNode.left == null) {

if (focusNode == root)

root = focusNode.right;

// If focus Node was on the left of parent

// move the focus Nodes right child up to the

// parent node

else if (isItALeftChild)

parent.left = focusNode.right;

// Vice versa for the left child

else

parent.right = focusNode.right;

}

// If Two children - replacement needed

else {

Node replacement = getReplacementNode(focusNode);

// If the focusNode is root replace root

if (focusNode == root)

root = replacement;

// If the deleted node was a left child

else if (isItALeftChild)

parent.left = replacement;

// If the deleted node was a right child

else

parent.right = replacement;

replacement.left = focusNode.left;

}

return true;

}

public Node getReplacementNode(Node replacedNode) {

Node replacementParent = replacedNode;

Node replacement = replacedNode;

Node focusNode = replacedNode.right;

// While there are no more left children

while (focusNode != null) {

replacementParent = replacement;

replacement = focusNode;

focusNode = focusNode.left;

}

// If the replacement isn't the right child

// move the replacement into the parents

// leftChild slot and move the replaced nodes

// right child into the replacements rightChild

if (replacement != replacedNode.right) {

replacementParent.left = replacement.right;

replacement.right = replacedNode.right;

}

return replacement;

}

}

public class Node {

int key;

String name;

Node left;

Node right;

//constructor

Node(int key, String name) {

this.key = key;

this.name = name;

}

//print

public String toString() {

return name + " with the key " + key;

}

}

public class Test {

public static void main(String[] args) {

BinaryTree Tree = new BinaryTree();

Tree.insert(50, "CEO");

Tree.insert(25, "Vice President");

Tree.insert(15, "Project Manager");

Tree.insert(30, "Assistant");

Tree.insert(75, "Production Manager");

Tree.insert(85, "Salesman");

//Tree.remove(75);

Tree.inorderTraverse(Tree.root);

//Tree.preorderTraverse(Tree.root);

//Tree.postorderTraverse(Tree.root);

System.***out***.println("\nNode with the key 75");

System.***out***.println(Tree.search(75));

}

}