 **广东理工学院实验(训)报告**

**姓 名： 叶锦亮 学 号： 1812402301044**

**专业、班级： 18软工1班 课程名称： 数据结构与算法(Java)**

**指导教师： 翟允赛 实验地点：**

**实验时间： 2020.6.14 实验学时： 2学时**

**实验项目： 14 二叉排序树**  **成 绩：**

|  |
| --- |
| **实验目的：**  1.掌握二叉排序树的插入与遍历算法的基本思想及实现。  2.掌握二叉排序树的删除算法的思想及实现。  3.了解使用二叉排序树解决实际问题的方法。 |
| 实验内容和要求:  1、将该实验报告以“**14**+全部学号+姓名.doc”格式命名，完成后于本周周日24：00前交到班长处，班长再统一交给我，过期不候！！（以后实验都按照本次要求）。  2、实现二叉排序树的建立，在二叉排序树上实现对给定值进行查找操作。  编程实现如下功能：  假设每个姓氏有一个编码与其对应，下表是10个姓氏及其对应的编码。   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 编码 | 50 | 13 | 63 | 8 | 36 | 90 | 5 | 10 | 18 | 70 | | 姓氏 | Wang | Li | Zhang | Liu | Chen | Yang | Huang | Zhou | Wu | Zhou |   是根据上表中的数据设计一个基于二叉排序树的测试程序，要求完成以下操作：   1. 以上表中的编码为关键字建立一棵二叉排列树。 2. 以中序遍历形式显示该二叉排序树。 3. 分别查找关键字值为63和39对应的姓氏。 4. 删除关键字值为13的结点。 |
| **实验过程:**  **二叉排序树实现：**  **package serch;**  **import soft.KeyType;**  **import soft.RecordNode;**  **import tree.BinaryTree;**  **import tree.BinaryTreeNode;**  **public class BinarySerchTree {**  **private BinaryTree tree;**    **public BinaryTree getTree() {**  **return tree;**  **}**  **public void setTree(BinaryTree tree) {**  **this.tree = tree;**  **}**  **public BinarySerchTree() {**    **}**    **public BinarySerchTree(RecordNode[] rs) {**  **createBinarySerchTree(rs);**  **}**    **public void createBinarySerchTree(RecordNode[] rs) {**  **if(rs == null||rs[0] == null)return;**  **tree = new BinaryTree();**  **//用传入输出的第一个元素作为二叉查找树的根节点**  **tree.setRoot(new BinaryTreeNode(rs[0]));**    **//辅助变量**  **BinaryTreeNode currentTreeNode = tree.getRoot();**    **for(int i=1;i<rs.length;i++) {**  **if(rs[i]==null)return;**  **currentTreeNode = tree.getRoot();**  **while(currentTreeNode!=null) {**  **/\*1. 比较当前插入元素的关键字与当前指向节点的关键字的大小**  **\*2. 如果相等，则表示该元素再书中已经存在，跳过这个元素插入下一个元素**  **\* 否则根据比较的结果并且当前指向节点的左子树或右子树不为空则把当前指向节点切换到当前指向节点的左子树或右子树并且回到第一步**  **\* 如果当前指向节点的左子树或右子树为空则将当前插入元素插入到这个位置，然后进入到写一个插入元素的插入操作**  **\*/**  **int compareResut = ((RecordNode)currentTreeNode.getData()).key.compareTo((KeyType) rs[i].key);**  **if(compareResut==0) {**  **break;**  **}else if(compareResut<0){**  **if(currentTreeNode.getrChild()==null) {**  **currentTreeNode.setrChild(new BinaryTreeNode(rs[i]));**  **break;**  **}else {**  **currentTreeNode = currentTreeNode.getrChild();**  **}**    **}else if(compareResut>0){**  **if(currentTreeNode.getlChild()==null) {**  **currentTreeNode.setlChild(new BinaryTreeNode(rs[i]));**  **break;**  **}else {**  **currentTreeNode = currentTreeNode.getlChild();**  **}**  **}**  **}**  **}**  **}**    **public Object serch(Comparable key) {**  **if(key ==null)return null;**  **BinaryTreeNode btn = this.tree.getRoot();**  **while(btn!=null) {**  **int compareResult = ((RecordNode)btn.getData()).key.compareTo((KeyType) key);**  **if(compareResult==0) {**  **return btn.getData();**  **}else if(compareResult >0) {**  **btn = btn.getlChild();**  **}else {**  **btn = btn.getrChild();**  **}**  **}**  **return null;**  **}**  **public void insertNode(RecordNode node) throws Exception {**  **if(node.key==null)throw new Exception("试图插入不可比较对象。");**  **if(this.tree.getRoot()==null) {**  **tree.setRoot(new BinaryTreeNode(node));**  **return;**  **}**    **KeyType key = (KeyType) node.key;**  **BinaryTreeNode btn = this.tree.getRoot();**  **while(btn!=null) {**  **int compareResut = ((RecordNode)btn.getData()).key.compareTo(key);**  **if(compareResut == 0) {**  **return;**  **}else if(compareResut >0) {**  **if(btn.getlChild()==null) {**  **btn.setlChild(new BinaryTreeNode(node));**  **return;**  **}else {**  **btn = btn.getlChild();**  **}**  **}else {**  **if(btn.getrChild()==null) {**  **btn.setrChild(new BinaryTreeNode(node));**  **return;**  **}else {**  **btn = btn.getrChild();**  **}**  **}**  **}**  **}**    **public Object deleteNode(Comparable key) {**  **if(key == null)return null;**  **BinaryTreeNode currentNode = tree.getRoot();**  **BinaryTreeNode currentNode\_parent = null;**  **if(currentNode ==null) return null;**  **Object returnData = null;**  **//第一步,先在树中找到该节点。**  **while(currentNode!=null) {**  **int compareResult = ((RecordNode)currentNode.getData()).key.compareTo((KeyType) key);**  **if(compareResult==0) {**  **returnData = currentNode.getData();**  **boolean hasRChild = currentNode.getrChild()!=null;**  **boolean hasLChild = currentNode.getlChild()!=null;**  **if(hasLChild^hasRChild) { //仅有左子树或右子树**  **if(hasLChild) {**  **if(currentNode\_parent.getlChild().equals(currentNode)) {**  **currentNode\_parent.setlChild(currentNode.getlChild());**  **}else {**  **currentNode\_parent.setrChild(currentNode.getlChild());**  **}**  **}else {**  **if(currentNode\_parent.getlChild().equals(currentNode)) {**  **currentNode\_parent.setlChild(currentNode.getrChild());**  **}else {**  **currentNode\_parent.setrChild(currentNode.getrChild());**  **}**  **}**  **}else if(hasLChild&&hasRChild) { //有左右子树**  **//声明两个辅助变量来保存找到的前趋节点以及其父节点，方便后面对树结构的修改**  **BinaryTreeNode anteriorNode,anteriorNode\_parent;**  **anteriorNode\_parent = currentNode;**  **anteriorNode = currentNode.getlChild();**    **//寻找前趋节点**  **while(anteriorNode!=null) {**  **if(anteriorNode.getrChild()!=null) {**  **anteriorNode\_parent = anteriorNode;**  **anteriorNode = anteriorNode.getrChild();**  **}else {**  **break;**  **}**  **}**    **/\*先将前趋节点的左孩子吧前趋节点替换掉，这里要分两种情况**  **\* 1. 找到的前趋节点就是被删除节点的左孩子：**  **\* 这种情况要把前趋节点的父节点的左孩子替换成前趋节点的左孩子。**  **\* 2. 找到的前趋节点不是1说说的情况：**  **\* 这种情况直接把前趋节点的父节点的右孩子替换成前趋节点的左孩子就可以了**  **\*/**  **if(currentNode.getlChild().equals(anteriorNode)) {**  **anteriorNode\_parent.setlChild(anteriorNode.getlChild());**  **}else {**  **anteriorNode\_parent.setrChild(anteriorNode.getlChild());**  **}**    **//然后将前趋节点的左右孩子修改为被删除节点的左右孩子**  **anteriorNode.setlChild(currentNode.getlChild());**  **anteriorNode.setrChild(currentNode.getrChild());**    **//最后将被删除节点替换成前趋节点**  **if(currentNode\_parent.getlChild().equals(currentNode)) {**  **currentNode\_parent.setlChild(anteriorNode);**  **}else {**  **currentNode\_parent.setrChild(anteriorNode);**  **}**    **}else { //叶子结点**  **if(currentNode\_parent.getlChild().equals(currentNode)) {**  **currentNode\_parent.setlChild(null);**  **}else {**  **currentNode\_parent.setrChild(null);**  **}**  **}**  **break;**  **}else if(compareResult >0) {**  **currentNode\_parent = currentNode;**  **currentNode = currentNode.getlChild();**  **}else {**  **currentNode\_parent = currentNode;**  **currentNode = currentNode\_parent.getrChild();**  **}**  **}**  **return returnData;**  **}**    **public void inRootTraverse(BinaryTreeNode t) {**  **if (t != null) {**  **inRootTraverse(t.getlChild());**  **System.out.print((RecordNode)t.getData());**  **inRootTraverse(t.getrChild());**  **}**  **}**    **public void preRootTraverse(BinaryTreeNode t) {**  **if (t != null) {**  **System.out.print((RecordNode)t.getData());**  **preRootTraverse(t.getlChild());**  **preRootTraverse(t.getrChild());**  **}**  **}**    **public void postRootTraverse(BinaryTreeNode t) {**  **if (t != null) {**  **postRootTraverse(t.getlChild());**  **postRootTraverse(t.getrChild());**  **System.out.print((RecordNode)t.getData());**  **}**  **}**  **}**  **测试方法：**  @Test  **public** **void** test01() {  RecordNode[] rs = {  **new** RecordNode(50,**new** Person("Wang")),  **new** RecordNode(13,**new** Person("Li")),  **new** RecordNode(63,**new** Person("Zhang")),  **new** RecordNode(8,**new** Person("Liu")),  **new** RecordNode(36,**new** Person("Chen")),  **new** RecordNode(90,**new** Person("Yang")),  **new** RecordNode(5,**new** Person("Huang")),  **new** RecordNode(10,**new** Person("Zhao")),  **new** RecordNode(18,**new** Person("Wu")),  **new** RecordNode(70,**new** Person("Zhou"))  };    BinarySerchTree bst = **new** BinarySerchTree(rs);  System.***out***.println("中序遍历二叉排序树树：");  bst.inRootTraverse(bst.getTree().getRoot());  System.***out***.println();  Object serchResult = bst.serch(**new** KeyType(63));  System.***out***.print("查找关键字：63，");  **if**(serchResult == **null**) {  System.***out***.println("失败！");  }**else** {  System.***out***.println("成功！结果："+serchResult);  }  serchResult = bst.serch(**new** KeyType(39));  System.***out***.print("查找关键字：39，");  **if**(serchResult == **null**) {  System.***out***.println("失败！");  }**else** {  System.***out***.println("成功！结果："+serchResult);  }    serchResult = bst.deleteNode(**new** KeyType(13));  System.***out***.print("删除关键字：13，");  **if**(serchResult == **null**) {  System.***out***.println("失败！");  }**else** {  System.***out***.println("成功！结果："+serchResult);  }    System.***out***.println("删除后中序遍历：");  bst.inRootTraverse(bst.getTree().getRoot());  } |
| **实验结果：** |
| **实验总结**  **二叉排序树的核心其实还是折半查找，只要对折半查找有充足的理解二叉排序树也同样是可以理解的。** |