学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 李煦 | **学号** | 2021902007 |
| **学院** | 信息工程学院 | **专业** | 电子信息类 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 1. **学习内容** 2. **树结构** 3. **二分搜索树** 4. **二分搜索树的添加和查询** 5. **二分搜索树的层序遍历和前序遍历** 6. **删除二分查找树中的最大元素和最小元素** 7. **学习收获（思考）** 8. **学习了树结构** 9. **学习二分查找树的一些操作** |
| 本周练习过的代码（例） |
| （本周练习写过的代码，如实填写，疑似抄袭会单独测试）  *代码1：*  *#include<stdio.h>*  *int main()*  *{*  *printf("HelloWorld\n");*  *return 0;*  *}*  *代码2：*  …….. |
| 本周练习过的代码 |
| package com.fx.binarySearchTree; import com.fx.printer.BinaryTreeInfo; import java.util.Stack; public class BinarySearchTree<E> implements BinaryTreeInfo {     private static class Node<E> {         E element;         Node<E> left;         Node<E> right;         Node<E> parent;          public Node(E element, Node<E> parent) {             this.element = element;             this.parent = parent;         }     }      private int size;     private Node<E> root;     private Comparator<E> comparator;      public BinarySearchTree() {      }      public BinarySearchTree(Comparator<E> comparator) {         this.comparator = comparator;     }      public void add(E element) {         elementNotNullCheck(element);            if (root == null) {             root = new Node<>(element, null);         }               Node<E> node = root;          Node<E> parent = root;        int cmp = 0;         while (node != null) ；             cmp = compare(element, node.element);                parent = node;             if (cmp > 0) {                 node = node.right;             } else if (cmp < 0) {                 node = node.left;             } else {                 return;             }         }         Node<E> newNode = new Node<>(element, parent);         if (cmp > 0) {             parent.right = newNode;         } else {             parent.left = newNode;         }     }      public void remove(E element) {      }      public boolean contains(E e1, E e2) {         return true;     }        private int compare(E e1, E e2) {         if (comparator != null) {             return comparator.compare(e1, e2);         }                  return ((java.lang.Comparable<E>) e1).compareTo(e2);     }       public int size() {         return size;     }        public boolean isEmpty() {         return size == 0;     }          public void clear() {      }         private void elementNotNullCheck(E element) {         if (element == null) {             throw new IllegalArgumentException("element must not be null");         }     } }  public class Test {   private FigureNode root;       public static void main(String[] args) {     Test test = new Test();     test.createBinaryTree();     test.preOrderSeach(5);     System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");     test.preOrderSeach(8);   }     private void createBinaryTree() {     FigureNode figureNode = new FigureNode(1,"如来佛祖");     FigureNode figureNode2 = new FigureNode(2,"观音菩萨");     FigureNode figureNode3 = new FigureNode(3,"牛魔王");     FigureNode figureNode4 = new FigureNode(4,"孙悟空");     FigureNode figureNode5 = new FigureNode(5,"猪八戒");     FigureNode figureNode6 = new FigureNode(6,"红孩儿");     FigureNode figureNode7 = new FigureNode(7,"铁扇公主");         root = figureNode;     root.setLeft(figureNode2);      root.setRight(figureNode3);     figureNode2.setLeft(figureNode4)；     figureNode2.setRight(figureNode5)；     figureNode3.setLeft(figureNode6);     figureNode3.setRight(figureNode7);     private void preOrderSeach(int no) {     if (root != null) {       System.out.println("前序遍历查找");       FigureNode result = preOrderSeach(root, no);       if (result == null) {         System.out.println("没有序号为" + no + "的人物");       } else {         System.out.println("找到了序号为" + no + "的人物，他（她）是"             + result.getName());       }     } else {       System.out.println("这是一棵空的二叉树");     }   }     private FigureNode preOrderSeach(FigureNode node, int no) {     if (node.getNo() == no) {// 比较当前节点       return node;     }         FigureNode res = null;      if (node.getLeft() != null) {       res = preOrderSeach(node.getLeft(), no);     }      if (res != null) {       return res;     }       if (node.getRight() != null) {        res = preOrderSeach(node.getRight(), no);     }     return res;   }  public class LevelOrder {     public static List<List<Integer>> levelOrder(TreeNode root){             List<List<Integer>> resLists = new ArrayList<>();             Queue<TreeNode> queue = new LinkedList<>();         if (root == null){             return null;         }         queue.add(root);         while(!queue.isEmpty()){             int queueSize = queue.size();                      ArrayList<Integer> temp = new ArrayList<Integer>();             for (int i = 0; i <queueSize ; i++) {                             TreeNode treeNode = queue.poll();                 temp.add(treeNode.val);                           if (treeNode.left!=null){                     queue.add(treeNode.left);                 }                           if (treeNode.right!=null){                     queue.add(treeNode.right);                 }             }                          resLists.add(temp);         }         return resLists;     } }  class TreeNode {     int val;    TreeNode left;      TreeNode right;        public TreeNode(int val) {         this.val = val;     } }  public class TestBinarySearchTree {     private static class TreeNode {         public int val;         public TreeNode left;         public TreeNode right;           public TreeNode(int val) {             this.val = val;         }     }       private TreeNode root;          public boolean contains(int val) {         if(root == null) {             return false;         }         TreeNode cur = root;         while(cur != null) {             if(cur.val == val) {                 return true;             }else if(cur.val > val) {                 cur = cur.left;             }else {                 cur = cur.right;             }         }         return false;     }         public boolean insert(int val) {         if(root == null) {             root = new TreeNode(val);             return true;         }         TreeNode cur = root;         TreeNode parent = null;         while(cur != null) {             if(cur.val < val) {                 parent = cur;                 cur = cur.right;             }else if(cur.val > val) {                 parent = cur;                 cur = cur.left;             }else {                 return false;//不能插入相同的数字             }         }         if(parent.val < val) {             parent.right = new TreeNode(val);         }else {             parent.left = new TreeNode(val);         }         return true;     }       /\*\*      \* 二叉查找树的删除      \* @param val      \* @return      \*/     public boolean remove(int val) {         if(root == null) {             return false;         }         TreeNode cur = root;         TreeNode parent = null;         while(cur != null) {             if(cur.val < val) {                 parent = cur;                 cur = cur.right;             }else if(cur.val > val) {                 parent = cur;                 cur = cur.left;             }else {                 removeNode(parent,cur);                 return true;             }         }         return false;     }       public void removeNode(TreeNode parent, TreeNode cur) {         if(cur.left == null) {             if(cur == root) {                 root = cur.right;             }else if(parent.right == cur) {                 parent.right = cur.right;             }else {                 parent.left = cur.right;             }         }else if(cur.right == null) {             if(cur == root) {                 cur = cur.left;             }else if(parent.left == cur) {                 parent.left = cur.left;             }else {                 parent.right = cur.left;             }         }else {             TreeNode target = cur.right;             TreeNode targetParent = cur;             while(target != null) {                 targetParent = target;                 target = target.left;             }             cur.val = target.val;             if(target == targetParent.left) {                 targetParent.left = target.right;             }else {                 targetParent.right = target.right;             }         }     }      public void inorder(TreeNode root) {         if(root == null) {             return ;         }         inorder(root.left);         System.out.print(root.val + " ");         inorder(root.right);     }           public void findMax(TreeNode root) {         if(root == null) {             System.out.println("二叉查找树为空");             return ;         }         while(root.right != null) {             root = root.right;         }         System.out.println("最大值为：" + root.val);     }           public void findMin(TreeNode root) {         if(root == null) {             System.out.println("二叉查找树为空");             return ;         }         while(root.left != null) {             root = root.left;         }         System.out.println("最小值为：" + root.val);     } } |

1. 该表信息将会作为你报名申请的重要依据，请认真仔细填写。
2. 培训班有严格的制度，请认真阅读规则并结合自身情况填写该表。
3. 完成该表填写后以“xxx学习情况”命名，及时上传作业。

最后希望大家能够加入我们，一起努力，共同进步！