学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 杜金瑞 | **学号** | 2020905073 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术 |

|  |
| --- |
| 学习情况简述 |
| C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\DCEFB260BDF55B99E1BB843A5D51196C.jpg  C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\34AA9F4ACFE9A19E1108FC58E1230A51.jpg  C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\4DEAEFA1E7A8065098AF00F382BCAEEE.jpg  C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\280BBD69B9071EEA1AE07A90145C306E.jpg |
| 本周练习过的代码（例） |
| import java.util.ArrayList;  public class AVLTree <K extends Comparable <K>,V>{   public class Node{   public K key;  public V value;   public Node left,right;  public int height;   public Node(K key,V value){  this.key=key;  this.value=value;  left=null;  right=null;  height=1;  }  }   private Node root;  private int size;   public AVLTree(){  root=null;  size=0;  }   public int getSize(){  return size;  }   public boolean isEmpty(){  return size==0;  }   //判断二叉树是否是平衡二叉树  public boolean isBST(){  ArrayList<K> keys=new ArrayList<>();  inOrder(root,keys);  for(int i=0;i<keys.size();i++){  if(keys.get(i-1).compareTo(keys.get(i))>0)  return false;  }  return true;  }  private void inOrder(Node node,ArrayList<K> keys){  if(node==null)  return;   inOrder(node.left,keys);  keys.add(node.key);  inOrder(node.right,keys);  }   //获得节点Node的高度  private int getHeight(Node node){  if(node==null)  return 0;  return node.height;  }   //获得节点Node的平衡因子  private int getBalanceFactor(Node node){  if(node==null)  return 0;  return getBalanceFactor(node.left)-getBalanceFactor(node.right);  }   //判断是否是平衡二叉树  public boolean isBalanced(){  return isBalanced(root);  }   private boolean isBalanced(Node node){  if(node==null)  return true;   int balancedFactor=getBalanceFactor(node);  if(Math.*abs*(balancedFactor)>1)  return false;  return isBalanced(node.left)&&isBalanced(node.right);   }  //向二分搜索树中添加新的元素（key,value）  public void add(K key,V value){   root=add(root,key,value);  }   private Node getnode(Node node,K key){   if(node==null)  return null;   if(node.key.compareTo(key)==0)  return node;  else if(node.key.compareTo(key)<0)  return getnode(node.right,key);  else  return getnode(node.left,key);   }   //右旋转  private Node rightRotate(Node y){  Node x=y.left;  Node T3=x.right;   x.right=y;  y.left=T3;   y.height=Math.*max*(getHeight(y.left),getHeight(y.right))+1;  x.height=Math.*max*(getHeight(x.left),getHeight(x.right))+1;   return x;  }   //左旋转  private Node leftRotate(Node y){  Node x=y.right;  Node T3=x.left;   x.left=y;  y.right=T3;   y.height=Math.*max*(getHeight(y.left),getHeight(y.right))+1;  x.height=Math.*max*(getHeight(x.left),getHeight(x.right))+1;   return x;  }  //以node为根节点的二分搜索树中添加，返回插入新节点后二分搜索树的根  private Node add(Node node,K key,V value){   if(node==null){  size++;  return new Node(key,value);  }  if(key.compareTo(node.key)<0)  node.left=add(node.left,key,value);  else if(key.compareTo(node.key)>0)  node.right=add(node.right,key,value);  else node.value=value;   //更新Height  node.height=1+Math.*max*(getHeight(node.left),getHeight(node.right));   //计算平衡因子  int balanceFactor=getBalanceFactor(node);   //平衡维护  //LL  if(balanceFactor>1&&getBalanceFactor(node.left)>=0)  return rightRotate(node);  //RR  if(balanceFactor<-1&&getBalanceFactor(node.right)<=0)  return leftRotate(node);  //LR  if(balanceFactor>1&&getBalanceFactor(node.left)<0){  node.left=leftRotate(node.left);  return rightRotate(node);  }  //RL  if(balanceFactor<-1&&getBalanceFactor(node.right)>0){  node.right=rightRotate(node.right);  return leftRotate(node);  }   return node;  }   public Node minimum(Node node){  if(node.left==null)  return node;   return minimum(node.left);  }    public V removee(K key){  Node node = getnode(root,key);  if(node!=null){  root=removee(root,key);  return node.value;  }  return null;  }  public Node removee(Node node,K key){  if(node==null)  return null;   Node retNode;  if(key.compareTo(node.key)<0) {  node.left = removee(node.left, key);  retNode= node;  }  else if(key.compareTo(node.key)>0) {  node.right = removee(node.right, key);  retNode= node;  }  else{ //e.compareTo(node.e)==0  //待删除节点右子树为空  if(node.right==null){  Node left=node.left;  node.left=null;  size--;  retNode= left;  }   //待删除节点左子树为空  else if(node.left==null){  Node right = node.right;  node.right=null;  size--;  retNode= right;  }   //待删除节点左右子树都不为空  else {  Node successor = minimum(node.right);  successor.right = removee(node.right, successor.key);  successor.left = node.left;   node.left = node.right = null;  retNode = successor;  }  }   if(retNode==null)  return null;  //更新Height  retNode.height=1+Math.*max*(getHeight(retNode.left),getHeight(retNode.right));   //计算平衡因子  int balanceFactor=getBalanceFactor(retNode);   //平衡维护  //LL  if(balanceFactor>1&&getBalanceFactor(retNode.left)>=0)  return rightRotate(retNode);  //RR  if(balanceFactor<-1&&getBalanceFactor(retNode.right)<=0)  return leftRotate(retNode);  //LR  if(balanceFactor>1&&getBalanceFactor(retNode.left)<0){  retNode.left=leftRotate(retNode.left);  return rightRotate(retNode);  }  //RL  if(balanceFactor<-1&&getBalanceFactor(retNode.right)>0){  retNode.right=rightRotate(retNode.right);  return leftRotate(retNode);  }   return retNode;  } }  import java.util.LinkedList; import java.util.Queue; import java.util.Stack;  public class RBTree <K extends Comparable <K>,V>{   //静态的不可以修改的  private static final boolean *RED*=true;  private static final boolean *BLACK*=true;   public class Node{   public K key;  public V value;  public Node left,right;  public boolean color;   public Node(K key,V value){  this.key=key;  this.value=value;  left=null;  right=null;  this.color=*RED*;  }  }   private Node root;  private int size;   public RBTree(){  root=null;  size=0;  }   public int getSize(){  return size;  }   public boolean isEmpty(){  return size==0;  }   public boolean isRed(Node node){  if(node==null)  return *BLACK*;  return node.color;  }   //添加元素后两节点情形  private Node leftRotate(Node node){  Node x=node.right;   node.right=x.left;  x.left=node;  x.color= node.color;  node.color=*RED*;   return x;  }   private Node rightRotate(Node node){  Node x=node.left;   node.left=x.right;  x.right=node;  x.color=node.color;  node.color=*RED*;   return x;  }  //add后三节点情形  private void flipColors(Node node){   node.color=*RED*;  node.left.color=*BLACK*;  node.right.color=*BLACK*;  }   public void addd(K key,V value){   root =addd(root,key,value);  root.color=*BLACK*;  }  private Node addd(Node node,K key,V value ){   if(node==null){  size++;  return new Node(key,value);  }   if(key.compareTo(node.key)<0)  node.left=addd(node.left,key,value);  else if(key.compareTo(node.key)>0)  node.right=addd(node.right,key,value);  else  node.value=value;   if(isRed(node.right)&&!isRed(node.left))//右节点为红色，左节点为黑色或无左节点  node=leftRotate(node);   if(isRed(node.left)&&isRed(node.left.left))//左节点为红色且左节点的左节点为红色  node=rightRotate(node);   if(isRed(node.left)&&isRed(node.right))//左节点右节点都为红色  flipColors(node);   return node;  }   } |