中北大学大数据学院

实验报告

课 程 名 称 算 法 分 析 与 设 计 学 号 1807004716学 生 姓 名 王明凯教 师 井 超

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **专业** | **数据科学与**  **大数据技术** | **实验室名称** |  | **实验时间** |  |
| **实验名称**  AVL平衡树 | | | | | |
| **实验目的**  学习平衡平衡树的建立 | | | | | |
| **实验题目**  建立AVL平衡树 | | | | | |
| **实验源代码**  -----------------------------------节点------------------------------------------------  public class Listnode {  Listnode farther;//父节点  Listnode rightChild;//右孩子  Listnode leftChild;//左孩子  int KEY;  int heigh;  boolean flag;//这个变量时中序遍历的时候用的  public Listnode()  {  this.KEY=0;  this.heigh=0;  this.flag=false;  this.farther=null;  this.leftChild=null;  this.rightChild=null;  }  public Listnode(int key)  {  this.KEY=key;  this.heigh=0;  this.flag=false;  this.farther=null;  this.leftChild=null;  this.rightChild=null;  }  public Listnode(int key,Listnode father)  {  this.KEY=key;  this.heigh=0;  this.flag=false;  this.farther=father;  this.leftChild=null;  this.rightChild=null;  }  public Listnode(int key,Listnode father,Listnode LC,Listnode RC)  {  this.KEY=key;  this.heigh=0;  this.flag=false;  this.farther=father;  this.leftChild=LC;  this.rightChild=RC;  }  int getBalance()  {  int leftHigh=this.leftChild==null? -1:this.leftChild.heigh;  int rightHight=this.rightChild==null? -1:this.rightChild.heigh;  return leftHigh-rightHight;  }  boolean isLeftChildOf(Listnode par)  {  return this==par.leftChild;  }  boolean isRightChildOf(Listnode par)  {  return this==par.rightChild;  }  boolean isFatherOf(Listnode son)  {  return this==son.farther;  }  }  ------------------------------------------avl树----------------------------------------  import java.util.Stack;  public class AVL {  Listnode \_root;  private void rotateAt(Listnode v)  {  Listnode grand=v.farther.farther.farther;  Listnode newRoot=null;  if(v.isLeftChildOf(v.farther) && v.farther.isLeftChildOf(v.farther.farther))  newRoot = reShape34(v, v.farther, v.farther.farther, v.leftChild, v.rightChild, v.farther.rightChild, v.farther.farther.rightChild);  else if(v.isRightChildOf(v.farther) && v.farther.isRightChildOf(v.farther.farther))  newRoot = reShape34(v.farther.farther, v.farther, v, v.farther.farther.leftChild, v.farther.leftChild, v.leftChild, v.rightChild);  else if(v.isRightChildOf(v.farther) && v.farther.isLeftChildOf(v.farther.farther))  newRoot = reShape34(v.farther, v, v.farther.farther, v.farther.leftChild, v.leftChild, v.rightChild, v.farther.farther.rightChild);  else if(v.isLeftChildOf(v.farther) && v.farther.isRightChildOf(v.farther.farther))  newRoot = reShape34(v.farther.farther, v, v.farther, v.farther.farther.leftChild, v.leftChild, v.rightChild, v.farther.rightChild);  if(grand.KEY> newRoot.KEY)  grand.leftChild=newRoot;  else  grand.rightChild=newRoot;  newRoot.farther=grand;  }  private Listnode reShape34(Listnode a,Listnode b,Listnode c,Listnode T0,Listnode T1,Listnode T2,Listnode T3)  {  b.leftChild=a;  b.rightChild=c;  a.farther=b;  c.farther=b;  a.leftChild=T0; if(T0!=null) T0.farther=a;  a.rightChild=T1;if(T1!=null) T1.farther=a;  c.leftChild=T2; if(T2!=null) T2.farther=c;  c.rightChild=T3;if(T3!=null) T3.farther=c;  return b;  }  private void updateHight(Listnode v)  {  int leftHigh= v.leftChild==null? -1 :v.leftChild.heigh;  int rightHight=v.rightChild==null? -1:v.rightChild.heigh;  v.heigh=leftHigh>rightHight? leftHigh+1 : rightHight+1;  }  private Listnode findMax(Listnode v)  {  if(null==v)  return v;//这里可以变成抛异常  while(null != v.rightChild)  v=v.rightChild;  return v;  }  private Listnode findMin(Listnode v)  {  if(null==v)  return v;//这里可以变成抛异常  while(null != v.leftChild)  v=v.leftChild;  return v;  }  private void removeTools(Listnode v)//这个是删除的叶子节点的情况，其他的都可以转化为这种情况  {  Listnode x=v.farther;  if(x.KEY>v.KEY)  x.leftChild=null;//不指向他了，jvm虚拟机自动就清空了,c++还要删除，这里就不删除了  else  x.rightChild=null;  while(null!=x)  {  if(x.getBalance()>1||x.getBalance()<-1)  rotateAt(x);  updateHight(x);  x=x.farther;  }  }  public boolean isEmpty()  {  return \_root==null;  }  private void visitLastTools(Listnode v,Stack<Listnode> S)  {  while(null!=v)  {  S.push(v);  v=v.leftChild;  }  }  public void visitLast()  {  if(this.isEmpty())  return;  Stack<Listnode> S=new Stack<Listnode>();  visitLastTools(\_root, S);  while(!S.isEmpty())  {  Listnode v=S.pop();  if(!v.flag)  {  v.flag=true;  S.push(v);  v=v.rightChild;  visitLastTools(v, S);  }  else  System.out.println(v.KEY);  }  }  private void visitMiddleTools(Listnode v,Stack<Listnode> S)  {  while(null!=v)  {  S.push(v);  v=v.leftChild;  }  }  public void visitMiddle()  {  Stack<Listnode> S=new Stack<Listnode>();  visitMiddleTools(\_root, S);  while(!S.isEmpty())  {  Listnode v=S.pop();  System.out.println(v.KEY);  v=v.rightChild;  visitMiddleTools(v, S);  }  }  public Listnode search(int Key)  {  Listnode target=\_root;  Listnode father=null;  while(target!=null && target.KEY!=Key)  {  father=target;  target=target.KEY>Key? target.leftChild:target.rightChild;  }  return father;  }  public void insert(int Key)  {  Listnode father = search(Key);  Listnode node=new Listnode(Key,father);  if(null == father)  {  \_root=node;  return;  }  if(father.KEY>Key)  father.leftChild=node;  else  father.rightChild=node;  while(father!=null)  {  if(father.getBalance()>1||father.getBalance()<-1)  {  rotateAt(father);  break;//平衡之后，高度不变可以直接退出  }  updateHight(father);  father=father.farther;  }  }  public void remove(int Key)  {  if(isEmpty())  return;  Listnode father=search(Key);  Listnode node=father.KEY>Key? father.leftChild:father.rightChild;  if(node==null)//如果本来就不再树里边，那就不用操作，直接退出就可以;  return;  else//接下来是在树里边，这个有四种情况  {  if(null==node.leftChild&&null==node.rightChild)//删除叶子节点  removeTools(node);  else if(null==node.leftChild && null!=node.rightChild)//删除只有右子树的节点  {  Listnode targ=findMin(node.rightChild);  node.KEY=targ.KEY;  removeTools(targ);  }  else if(null!=node.leftChild && null==node.rightChild)//删除只有左子树的节点  {  Listnode targ=findMax(node.leftChild);  node.KEY=targ.KEY;  removeTools(targ);  }  else if(null!=node.leftChild && null!=node.rightChild)//删除有左子树，也有右子树的节点  {  Listnode targ=findMax(node.rightChild);  node.KEY=targ.KEY;  removeTools(targ);  }  }  }  }  -----------------------------------测试（插入中序打印）---------------------------------  public class Test {  public static void main(String[] args)  {  AVL avl=new AVL();  avl.insert(1);  avl.insert(7);  avl.insert(-4);  avl.insert(4);  avl.visitMiddle();  }  } | | | | | |
| **实验结果**（给出题目的运行结果（插入运行结果截图））    中序打印为由小到大的顺序 | | | | | |