山东大学软件学院

数据结构、算法与应用课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301249 | 姓名： 王帅 | | 班级： 4班 |
| 实验题目：实验六 堆和搜索树 | | | |
| 实验学时：4 | | 实验日期： 2019.10.10 | |
| 实验目的：  掌握堆和搜索树的基本概念，插入、删除方法。 | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  Microsoft Visual C++ | | | |
| 实验步骤与内容：   * 1. 输入一系列不为零的正整数（最多不超过 20 个），遇到 0 代表输入结束（不包含 0）。   2. 根据上面输入的数据序列，用初始化方法创建最大堆（不要用节点依次插入的办法创建最大堆），然后输出最大堆的层次序列。   3. 输出用堆排序后的排序结果。   4. 根据上面输入的数据，创建二叉搜索树（关键字不允许重复，如遇重复，则不重复插入该关键字），输出二叉搜索树的前序序列、中序序列（分行输出）。   实验代码如下：  #include<iostream>  using namespace std;  /\*  1、输入一系列不为零的正整数（最多不超过 20 个），遇到 0 代表输入结 束（不包含 0）。  2、根据上面输入的数据序列，用初始化方法创建最大堆（不要用节点依次插入的办法创建最大堆），然后输出最大堆的层次序列。  3、输出用堆排序后的排序结果。  4、根据上面输入的数据，创建二叉搜索树（关键字不允许重复，如遇重，则不重复插入该关键字），输出二叉搜索树的前序序列、中序序列（分行输出）。  \*/  //定义节点结构  class Maxheap  {  public:  Maxheap()  {  heap = new int[25];  size = 0;  }    void initialize()  {  for(int i = size/2; i>=1; i--) //i为heapsize/2，即为最后一个元素的父节点  {  heap[0] = heap[i]; //heap[0]存放临时元素i，堆中元素从heap[1]开始存放  int son = 2\*i; //i的左孩子    while(son <= size) //检查当前子树的每一个节点，保证其为最大堆  {  if(son < size && heap[son] < heap[son+1]) //如果有两个儿子，得到值比较大的儿子  son++;    if(heap[0] >= heap[son]) //父节点大于等于子节点，此子树已是最大堆，退出while，进行下一个子树的初始化  break;  else  {  heap[son/2] = heap[son]; //父节点小于子节点， 原子节点提到父节点，son为子节点的子节点，继续和原父节点比较  son = 2\*son;  }  }    heap[son/2] = heap[0]; //找到交换的son，i赋给其父节点  }  }    int size; //元素个数  int \*heap; //存放元素的数组  };  struct treeNode //二叉树的基本结构  {  int element;  treeNode \*leftchild,\*rightchild;    treeNode()  {  leftchild = rightchild = NULL;  }  treeNode(int e)  {  element=e;  leftchild = rightchild = NULL;  }  treeNode(int e,treeNode \*lc,treeNode \*rc)  {  element=e;  leftchild = lc;  rightchild = rc;  }  };  class tree  {  public:  tree() {root = NULL; size = 0;}  int treesize() {return size;}    //插入一个节点  void insert(int e)  {  if(size==0)  {  root = new treeNode(e);  size++;  }    else  {  treeNode \*temp=NULL; //temp用来遍历  treeNode \*lastnode=NULL; //lastnode用来存最后一个节点，e插到这个节点后边  temp=root; //从根节点开始遍历（此时根节点不空）    while(temp!=NULL) //temp空时，e应插到temp的父节点后边  {  lastnode=temp;  if(e==temp->element) //重复元素，不予插入  {  break;  }  if(e>temp->element)  {  temp=temp->rightchild;  }  else  {  temp=temp->leftchild;  }  }    if(e>lastnode->element)  {  treeNode \*tn = new treeNode();  lastnode->rightchild=tn;  tn->element=e;  size++;  }  if(e<lastnode->element)  {  treeNode \*tn = new treeNode();  lastnode->leftchild=tn;  tn->element=e;  size++;  }    }  }    //访问节点\*x，仅输出element域  void visit(treeNode \*x)  {  if(formark==0)  {  cout<<x->element;  }  else  {  cout<<","<<x->element;  }  }    //前序遍历  void preorder() {formark=0; preorder(root);}  void preorder(treeNode \*t)  {  if(t!=NULL)  {  visit(t);  formark++;  preorder(t->leftchild);  preorder(t->rightchild);  }  }    //中序遍历  void inorder() {formark=0; inorder(root);}  void inorder(treeNode \*t)  {  if(t!=NULL)  {  inorder(t->leftchild);  visit(t);  formark++;  inorder(t->rightchild);  }  }    private:  treeNode \*root;  int size;  int formark; //方便逗号的输出  };  int main()  {  cout<<"Input"<<endl;    int n;  int i=1;  Maxheap h;    //1、输入一系列不为零的正整数（最多不超过 20 个），遇到 0 代表输入结 束（不包含 0）  cin>>n;  while(n!=0)  {  h.heap[i]=n;  i++;  cin>>n;  }  h.size = i-1;    int a[h.size]; //数组a保存输入的初始数据，为第四问做准备  for(int j=0;j<h.size;j++)  {  a[j]=h.heap[j+1];  }    cout<<"Output"<<endl;    //2、根据上面输入的数据序列，用初始化方法创建最大堆（不要用节点依次插入的办法创建最大堆），然后输出最大堆的层次序列。  h.initialize();  for(int j=1;j<h.size;j++)  {  cout<<h.heap[j]<<",";  }  cout<<h.heap[h.size]<<endl; //此时数组中的内容即为层次遍历序列    //3、输出用堆排序后的排序结果。  /\*  堆排序：  1、创建一个最大堆H；  2、把堆首和堆尾元素互换；  3、把堆的大小减1，重新构造一个最大堆；  4、重复步骤2、3，直到堆的大小减少为1。  \*/  Maxheap h1;  h1.size=h.size;  int temp;  for(int j=1;j<=h.size;j++) //把原来堆赋给新堆，此最已是最大堆  {  h1.heap[j]=h.heap[j];  }  while(h1.size!=1) //每次把最大值放在最后边  {  temp=h1.heap[h1.size];  h1.heap[h1.size]=h1.heap[1];  h1.heap[1]=temp;    (h1.size)--;    h1.initialize();  }  for(int j=1;j<h.size;j++)  {  cout<<h1.heap[j]<<",";  }  cout<<h1.heap[h.size]<<endl;    //4、根据上面输入的数据，创建二叉搜索树（关键字不允许重复，如遇重，则不重复插入该关键字），输出二叉搜索树的前序序列、中序序列（分行输出）。  tree bst;    for(int j=0;j<h.size;j++)  {  bst.insert(a[j]);  }  bst.preorder();  cout<<endl;  bst.inorder();  cout<<endl;    cout<<"End";    return 0;  } | | | |
| 结论分析与体会：  堆和树的关系密不可分，需要结合起来对比学习，印象会更加深刻。 | | | |