软件工程课程设计（II）实验报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 林镕琛 | 学号 | | 181491117 |
| 班级 | 软件1班 | 时间 | | 2020-05-23 |
| 题目 | 二叉排序树的操作 | | | |
| 实验成绩 |  | 指导教师 | 杜岳峰 | |
| 实验内容：   1. 实验程序：   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Author：林镕琛  @Date： 2020-05-23  @Description：   二叉排序树的操作   给定数据：11，3,5,6,17,22  （1）创建一棵二叉排序树，标明左右孩子的情况  （2）二叉排序树上的查找操作  （3）二叉排序树上的删除操作    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <iostream>  #include<stdlib.h>  using namespace std;    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：创建树结点  @Parm: nodeElem 存放数据  @Parm：leftChild 左孩子结点指针  @Parm：rightChild 右孩子结点指针    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  typedef struct treeNode{   int nodeElem;   struct treeNode \*leftChild;   struct treeNode \*rightChild;  }treeNode;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：插入树结点  @Parm：nodeRoot 根节点指针  @Parm：insertNode 插入的那个结点  @return: 返回根节点指针  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  treeNode \*insertNodes(treeNode \*nodeRoot,treeNode \*insertNode){   treeNode \*nextPoint; // 下一个指针   treeNode \*nowPoint; // 当前指针   nextPoint = nodeRoot;   if (nodeRoot == NULL){ // 注意：此处一定不能忘记，否则无法创建   return insertNode;   }   while (nextPoint){   nowPoint = nextPoint; // 当前指针记住现在的状态   if (insertNode->nodeElem == nextPoint->nodeElem ){   return nodeRoot;   }else if (insertNode->nodeElem < nextPoint->nodeElem){   nextPoint = nextPoint->leftChild;   } else {   nextPoint = nextPoint->rightChild;   }   }   // 到达叶子结点，进行插入   if(nowPoint->nodeElem > insertNode->nodeElem){ //如果插入结点大于叶子结点   nowPoint->leftChild = insertNode;   }else {   nowPoint->rightChild = insertNode;   }   return nodeRoot;  }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：创建二叉排序树  @Parm：nodeRoot 根节点  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  treeNode \*treeCreate(treeNode \*nodeRoot){   int elem;   treeNode \*insertNode;   cout<<"请输入数据,以0结束:";   cin>>elem;   while (elem != 0){   insertNode = (treeNode \*)malloc(sizeof(treeNode));   insertNode->nodeElem = elem;   insertNode->leftChild = NULL;   insertNode->rightChild = NULL;   // 执行插入操作   nodeRoot = insertNodes(nodeRoot,insertNode);   cin>>elem;   }   return nodeRoot;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：二叉排序树上的显示操作  @Parm： nodeRoot 根节点    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void treePrint(treeNode \*nodeRoot){   // cout<<"请输入数据,以0结束:";   if(nodeRoot)   {   treePrint(nodeRoot->leftChild);   cout<<nodeRoot->nodeElem<<" ";   treePrint(nodeRoot->rightChild);       }  }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：二叉排序树上的查找操作  @Parm： nodeFind 要查找的结点    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int treeNodeFind(treeNode \*nodeRoot,int nodeFind){   while (nodeRoot){   if (nodeFind == nodeRoot->nodeElem){   cout<<nodeFind<<"在二叉树中";   return 0;   }else if (nodeFind < nodeRoot->nodeElem){   nodeRoot = nodeRoot->leftChild;   }else {   nodeRoot = nodeRoot->rightChild;   }   }   cout<<nodeFind<<"不在二叉树中";   return 1;  }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  @Function：二叉排序树上的删除操作  @Parm: elem 要删除的数  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int Delete(treeNode \*\*nodeDelete){   treeNode \*q,\*s;   if((\*nodeDelete)->rightChild == NULL){ // 右子树空则只需重接它的左子树   q = \*nodeDelete;   \*nodeDelete = (\*nodeDelete) ->leftChild;   free(q);   }else if((\*nodeDelete)->leftChild == NULL) // 左子树空则只需重接它的右子树   {   q = \*nodeDelete;   \*nodeDelete = (\*nodeDelete)->rightChild;   free(q);   }   // 左右子树均不空   else   {   q = \*nodeDelete;   s = (\*nodeDelete)->leftChild;   // 寻找删除结点的前驱   while (s->rightChild) {   q=s;   s=s->rightChild;   }   // 将被删结点前驱的值取代被删结点的值   (\*nodeDelete)->nodeElem = s->nodeElem;   if(q != (\*nodeDelete))   q->rightChild = s->leftChild; // 重接q的右子树   else   q->leftChild=s->leftChild; // 重接q的左子树   free(s);   }   return 1;  }    int treeNodeDelete(treeNode \*\*nodeRoot,int elem){ // 注意：此处必须要用二级指针，   // 否则删除成功也不能影响这个二叉排序树   if ((\*nodeRoot) == NULL){ //结点不存在   cout<<"结点不存在，退出删除操作"<<endl;   exit(1);   }else {   if (elem == (\*nodeRoot)->nodeElem)   return Delete(nodeRoot);   else if (elem < (\*nodeRoot)->nodeElem)   return treeNodeDelete(&(\*nodeRoot)->leftChild,elem);   else   return treeNodeDelete(&(\*nodeRoot)->rightChild,elem);   }  }    int main()  {   treeNode \*nodeRoot = NULL;   nodeRoot = treeCreate(nodeRoot);   cout<<"按中序遍历的方法访问：";   treePrint(nodeRoot);   //定义要查找和删除的点   int nodeFind,nodeDelete;   cout<<"\n输入查找的结点:";   cin>>nodeFind;   treeNodeFind(nodeRoot,nodeFind);   cout<<"\n输入要删除的结点：";   cin>>nodeDelete;   treeNodeDelete(&nodeRoot,nodeDelete);   cout<<"按中序遍历的方法访问：";   treePrint(nodeRoot);   return 0;  }   1. 实验运行结果： | | | | |