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：实验五 二叉树操作 | | | |
| 实验学时：4 | | 实验日期： 2019.10.10 | |
| 实验目的：  掌握二叉树的基本概念，二叉树的存储结构使用链表。 | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  Microsoft Visual C++ | | | |
| 实验步骤与内容：   * 1. 输入一个完全二叉树的层次遍历字符串，创建这个二叉树，输出这个二叉树的前序遍历字符串、中序遍历字符串、后序遍历字符串、结点数目、二叉树高度(上述每一个结果独立一行显示)。   2. 输入二叉树前序序列和中序序列(各元素各不相同)，创建这个二叉树，输出该二叉树的后序序列、层次遍历。   实验代码如下：  #include <iostream>  #include<queue>  #include <cstdio>  using namespace std;  /\*  1、输入一个完全二叉树的层次遍历字符串，创建这个二叉树，输出这个二叉树的  前序遍历字符串、中序遍历字符串、后序遍历字符串、结点数目、二叉树高度(上述每一个结果独立一行显示)。    2、输入二叉树前序序列和中序序列(各元素各不相同)，创建这个二叉树，输出该二叉树的后序序列、层次遍历。  \*/  struct treeNode  {  char element;  treeNode \*leftchild,\*rightchild;    treeNode()  {  leftchild = rightchild = NULL;  }  treeNode(char e)  {  element=e;  leftchild = rightchild = NULL;  }  treeNode(char e,treeNode \*lc,treeNode \*rc)  {  element=e;  leftchild = lc;  rightchild = rc;  }  };  class tree  {  public:  tree() {root = NULL; size = 0;}  int treesize() {return 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);  }  }    //后序遍历  void postorder() {formark=0; postorder(root);}  void postorder(treeNode \*t)  {  if(t!=NULL)  {  postorder(t->leftchild);  postorder(t->rightchild);  visit(t);  formark++;  }  }    //层次遍历  void levelorder() {formark=0; levelorder(root);}  void levelorder(treeNode \*t)  {  /\*queue<treeNode\*>q;  while(t!=NULL)  {  visit(t);  formark++;    //将t的孩子插入队列  if (t->leftchild != NULL) q.push(t->leftchild);  if (t->rightchild != NULL) q.push(t->rightchild);    //提取下一个要访问的节点  try{t = q.front();}  catch (queueEmpty) {return;}  q.pop();  }\*/    if (!t)  return;  treeNode \*temp;  queue<treeNode\*>q;  q.push(t);  while (!q.empty())  {  temp = q.front();  visit(temp);  formark++;  if (temp->leftchild)  q.push(temp->leftchild);  if (temp->rightchild)  q.push(temp->rightchild);  q.pop();  }  }    //确定树的高度  int treeheight()  {  int result;  result = treeheight(root);  return result;  }  int treeheight(treeNode \*t)  {  if(t == NULL) return 0;  int hl = treeheight(t->leftchild);  int hr = treeheight(t->rightchild);  if(hl>hr) return ++hl;  else return ++hr;  }    //用层遍历建立二叉树  void levelcreat()  {  int flag=0; //用来判断左右子树 左0右1  string str;  cin>>str;  int len = str.size();  size = str.size();  queue <treeNode\*> q;    root = new treeNode();  root->element = str.at(0);  root->leftchild = NULL;  root->rightchild = NULL;  q.push(root);    for(int i=1;i<len;i++)  {  treeNode \* tempNode = new treeNode();  tempNode->element = str.at(i);  tempNode->leftchild = NULL;  tempNode->rightchild = NULL;  if(flag==0)  {  q.front()->leftchild = tempNode;  q.push(tempNode);  flag=1;  }  else  {  q.front()->rightchild = tempNode;  q.push(tempNode);  q.pop();  flag=0;  }  }    }    //用前序遍历和中序遍历创建二叉树    /\*1.前序遍历即按照根节点、左子树、右子树的顺序遍历，所以前序遍历的第一个元素肯定是根节点。  2.得知根节点后，有两种情况，只有根节点或者有子树(判断前序遍历结果的长度)  3.在中序遍历中找到根节点root的位置，root的左边就是左子树遍历结果，右边是右子树遍历结果  4.在第三步中知道了左右子树的长度，再回到前序遍历中，删除第一个根节点，截取对应长度即可获得左右子树的前序遍历结果。  这时知道了左右子树的前序遍历和中序遍历结果，通过递归找出每个根节点和相应的左右子树，即可重建二叉树。\*/  /\*void pre\_increat()  {  string str1,str2,prestr,instr,leftstr,rightstr;  cin>>str1; //前序遍历结果  cin>>str2; //中序遍历结果  char temproot; //每一次递归的根节点  int n=0;    prestr = str1;  temproot = prestr.at(0);  if(prestr.size()>1) //有子树  {  instr = str2;  while(instr.at(n)!=temproot) {n++;}  if(n>0)  {  leftstr = instr.substr(0,n-1);  }    if(n<instr.size()-1)  {  rightstr = instr.substr(n+1,instr.size()-1);  }  }    }\*/  void pre\_increat()  {  string s1,s2;  cin>>s1; cin>>s2;  size = s1.size();  pre\_increat(s1,s2,size,root);  }    void pre\_increat(string preOrder, string inOrder, int length, treeNode\* &head){  if(length<1) return ;  char data = preOrder[0];  treeNode\* tempPoint = new treeNode(data);  head = tempPoint;  int indexOfInOrder = (int)inOrder.find(data);  string leftInOrder = inOrder.substr(0, indexOfInOrder);  string rightInOrder = inOrder.substr(indexOfInOrder+1);  string leftPreOrder = preOrder.substr(1, indexOfInOrder);  string rightPreOrder = preOrder.substr(indexOfInOrder+1);  pre\_increat(leftPreOrder, leftInOrder, indexOfInOrder, head->leftchild);  pre\_increat(rightPreOrder, rightInOrder, length-indexOfInOrder-1, head->rightchild);  }    private:  treeNode \*root;  int size;  int formark = 0; //方便逗号的输出  };  int main()  {  /\*  输入一个完全二叉树的层次遍历字符串，创建这个二叉树，输出这个二叉树的  前序遍历字符串、中序遍历字符串、后序遍历字符串、结点数目、二叉树高度(上述每一个结果独立一行显示)。  \*/  cout<<"Input1"<<endl;  tree t1;  t1.levelcreat();  cout<<"Output1"<<endl;  t1.preorder();  cout<<endl;  t1.inorder();  cout<<endl;  t1.postorder();  cout<<endl;  cout<<t1.treesize()<<endl;  cout<<t1.treeheight()<<endl;    /\*  输入二叉树前序序列和中序序列(各元素各不相同)，创建这个二叉树，输出该二叉树的后序序列、层次遍历。  \*/  cout<<"Input2"<<endl;  tree t2;  t2.pre\_increat();  cout<<"Output"<<endl;  t2.postorder();  cout<<endl;  t2.levelorder();  cout<<endl;  cout<<"End";    return 0;  } | | | |
| 结论分析与体会：  二叉树是最重要的一种树结构，为后边的堆打基础，需要熟练掌握。 | | | |