《数据结构与算法》实验报告6

——树与森林

吉林大学数学学院2020级计算1班 姓名 陈文宇 学号10200115

日期 2023/5/28

一、实验题目

实现二叉树的定义，使用二叉链表存储二叉树。

实现创建二叉树。

使用递归算法，和非递归算对二叉树进行先序、中序和后序遍历。

实现复制一颗二叉树。

使用二叉树结构存储和求解表达式。

实现对线索二叉树的定义，实现普通二叉树的线索化。

实现对线索二叉树的中序遍历；

二、算法简述

本章较为重要的算法是递归遍历，使用栈的非递归遍历，使用二叉树结构求解表达式的值。

递归遍历算法，即对非空树访问，调用本函数来遍历左子树和右子树，访问根节点，具体顺序看需求。

非递归遍历算法，将某子树入栈，并定义其状态为 Travel，即不迫切访问，根节点入栈，并定义其状态为Visit，即访问，将另一子树入栈，并定义其状态为Visit，即迫切访问。每次遇到可访问对象就访问。

二叉树求解表达式值，将表达式的操作符对应值写入二叉树，操作数在数组中的位序写入二叉树，递归地，遍历左子树求第一操作数，遍历右子树求第二操作数，调用根结点操作符，识别操作符并进行运算，返回结果。

三、程序代码

|  |
| --- |
| Tree\_Basic\_Opration.cpp |
| //陈文宇  //10200115  #include<iostream>  #include<stack>  using namespace std;  const int LISTINIT\_SIZE=100;  const int LISTINCREMENT=10;  const bool TRUE=1;  const bool FALSE=0;  const int PLUS=-1;  const int MINUS=-2;  const int ASTERISK=-3;  const int SLANT=-4;    //二叉树定义  typedef int TElemType;  typedef struct BiTNode{  TElemType data;  struct BiTNode \*lchild,\*rchild;  }BiTNode,\*BiTree;  typedef enum {Travel =1,Visit =0} TaskType;  //Traval =1：工作状态为遍历  //Visit =0：工作状态为访问  typedef struct{  BiTree ptr; //指向二叉树结点的指针  TaskType task; //任务的性质  }SElemType; //栈元素的类型定义  void CreatebiTree(BiTree &T);  void BiTreeDepth(BiTree T,int h,int &depth);  BiTNode \*GetTreeNode(TElemType iten,BiTNode \*lptr,BiTNode \*rptr);  BiTNode \*CopyTree(BiTree T);  double value(BiTree T,float \*opnd);  void Preorder(BiTree T, void (\*visit)(BiTree T));  void Inorder(BiTree T, void (\*visit)(BiTree T));  void Nextorder(BiTree T, void (\*visit)(BiTree T));  void Preorder\_iter(BiTree T, void (\*visit)(BiTree));  void Inorder\_iter(BiTree T, void (\*visit)(BiTree));  void Nextorder\_iter(BiTree T, void (\*visit)(BiTree));  void visit(BiTree T);  int main(){    BiTree T,S,V;  /\*  //----------------------------------------------------  cout<<"请按先序遍历的顺序创建二叉树，若其节点的左孩子或右孩子不存在则使用\*代替！如：（AB#DE###C##）"<<endl;  CreatebiTree(T);  cout<<"先序遍历( 递归 )的结果为:";  Preorder(T, visit);  cout<<endl;  cout<<"中序遍历( 递归 )的结果为:";  Inorder(T, visit);  cout<<endl;  cout<<"后序遍历( 递归 )的结果为:";  Nextorder(T, visit);  cout<<endl;    cout<<"先序遍历(非递归)的结果为:";  Preorder\_iter(T, visit);  cout<<endl;  cout<<"中序遍历(非递归)的结果为:";  Inorder\_iter(T, visit);  cout<<endl;  cout<<"后序遍历(非递归)的结果为:";  Nextorder\_iter(T, visit);  cout<<endl;  cout<<endl;  //----------------------------------------------------  //树的深度  int h=1,depth=0;  BiTreeDepth(T,h,depth);  cout<<"树的深度为："<<depth<<endl;  cout<<endl;  //----------------------------------------------------  //复制一棵二叉树  S=CopyTree(T);  cout<<"复制二叉树的结果为:";  Preorder(S, visit);  cout<<endl;  cout<<endl;  \*/  //----------------------------------------------------  //表达式求值  CreatebiTree(V);  cout<<"先序遍历( 递归 )的结果为:";  Preorder(V, visit);  cout<<endl;  float opnd[7]={0,3.4,4.7,29.2,61.5,3.14,1.41};  cout<<value(V,opnd)<<endl;    }  //二叉链表创建二叉树  void CreatebiTree(BiTree &T){  TElemType ch;  cin>>ch;  if(ch==0)T=NULL;  else{  T=new BiTNode;  T->data=ch;  CreatebiTree(T->lchild);  CreatebiTree(T->rchild);  }    }//CreatebiTree  //求二叉树的深度  void BiTreeDepth(BiTree T,int h,int &depth){  //h的初值为 1，depth的初值为 0  //h 指向结点所在的层,depth是深度  if(T){  if(h>depth) depth=h;  BiTreeDepth(T->lchild,h+1,depth);  BiTreeDepth(T->rchild,h+1,depth);  }  }//BiTreeDepth  //复制一棵二叉树  //复制左右子树，组装为树  BiTNode \*GetTreeNode(TElemType item,BiTNode \*lptr,BiTNode \*rptr){  BiTNode \*T;  T=new BiTNode;  T->data=item;  T->lchild=lptr;  T->rchild=rptr;  }  BiTNode \*CopyTree(BiTree T){  BiTree newlptr,newrptr,newnode;    if(!T) return NULL;  if(T->lchild) newlptr=CopyTree(T->lchild);  else newlptr=NULL;  if(T->rchild) newrptr=CopyTree(T->rchild);  else newrptr=NULL;  newnode=GetTreeNode(T->data,newlptr,newrptr);  return newnode;  }  //表达式求值  double value(BiTree T,float \*opnd){  double Lv,Rv,v;  if(!T) return 0;  if(T->data>=0) return opnd[T->data];  Lv=value(T->lchild,opnd);  Rv=value(T->rchild,opnd);  switch(T->data){  case PLUS: {v=Lv+Rv; break;}  case MINUS: {v=Lv-Rv; break;}  case ASTERISK: {v=Lv\*Rv; break;}  case SLANT: {v=Lv/Rv; break;}  default: cout<<"不合法的运算"<<endl;  }//switch  return v;  }  //先序遍历 （递归）  void Preorder(BiTree T, void (\*visit)(BiTree)){  if(T){  visit(T);  Preorder(T->lchild,visit);  Preorder(T->rchild,visit);  }  }  //中序变历 （递归）  void Inorder(BiTree T, void (\*visit)(BiTree)){  if(T){  Inorder(T->lchild,visit);  visit(T);  Inorder(T->rchild,visit);  }  }  //后序变历 （递归）  void Nextorder(BiTree T, void (\*visit)(BiTree)){  if(T){  Nextorder(T->lchild,visit);  Nextorder(T->rchild,visit);  visit(T);  }  }  //先序变历 （非递归）  void Preorder\_iter(BiTree T, void (\*visit)(BiTree)){  stack<SElemType> s;  SElemType e;  BiTree p;    e.task=Travel; e.ptr=T;  if(T) s.push(e);  while(!s.empty()){  e=s.top();  s.pop();  if(e.task==Visit) visit(e.ptr);  else{  if(e.ptr){ //处理非空树的遍历任务  p=e.ptr;  e.ptr=p->rchild; e.task=Travel;s.push(e); //最不迫切任务  e.ptr=p->lchild; e.task=Travel;s.push(e); //迫切任务进栈  e.ptr=p; e.task=Visit; s.push(e); //处理访问任务的工作状态和结点指针进栈  }//if  }//else  }//while    }//Preorder\_iter  //中序变历 （非递归）  void Inorder\_iter(BiTree T, void (\*visit)(BiTree)){  stack<SElemType> s;  SElemType e;  BiTree p;    e.task=Travel; e.ptr=T;  if(T) s.push(e);  while(!s.empty()){  e=s.top();  s.pop();  if(e.task==Visit) visit(e.ptr);  else{  if(e.ptr){ //处理非空树的遍历任务  p=e.ptr;  e.ptr=p->rchild; e.task=Travel;s.push(e); //最不迫切任务  e.ptr=p; e.task=Visit; s.push(e); //处理访问任务的工作状态和结点指针进栈  e.ptr=p->lchild; e.task=Travel;s.push(e); //迫切任务进栈  }//if  }//else  }//while    }//Inorder\_iter  //后序变历 （非递归）  void Nextorder\_iter(BiTree T, void (\*visit)(BiTree)){  stack<SElemType> s;  SElemType e;  BiTree p;    e.task=Travel; e.ptr=T;  if(T) s.push(e);  while(!s.empty()){  e=s.top();  s.pop();  if(e.task==Visit) visit(e.ptr);  else{  if(e.ptr){ //处理非空树的遍历任务  p=e.ptr;  e.ptr=p; e.task=Visit; s.push(e); //处理访问任务的工作状态和结点指针进栈  e.ptr=p->rchild; e.task=Travel; s.push(e); //最不迫切任务  e.ptr=p->lchild; e.task=Travel; s.push(e); //迫切任务进栈  }//if  }//else  }//while    }//Nextorder\_iter  //访问函数  void visit(BiTree T){  if(T) cout<<T->data<<" ";  }  bool treeEmpty(BiTree T){  if(T==NULL){  cout<<"该二叉树为空"<<endl;  return TRUE;  }  else{  cout<<"该二叉树不为空"<<endl;  return FALSE;  }  }  //全线索链表的中序遍历算法  void Inorder(BiThrTree H,void(\*visit)(BiTree)){  BiThrNode \*p;  p=H->succ;  while(p!=H){  visit(p);  p=p->succ;  }  }  //中序遍历线索化  void InorderThreading(BiThrTree &H,BiThrTree T){  BiThrNode pre;  H=new BiThrNode;  H->lchild=T; H->rchild=NULL;  if(!T){  InThreading(T,pre);  pre->succ=H;  H->pred=pre;  }  }  void InThreading(BiThrTree p,BiThrTree &T){  BiThrNode pre;  if(p){  InThreading(p->lchild,pre);  pre->succ=p; p->pred=pre;  pre=p;  InThreading(p->rchild,pre);  }  } |

四、实验结果



