**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

**一、调试成功程序及说明**

**1、**

**题目：**编程实现书P75 ADT BinaryTree 基本操作20个，用二叉链表结构实现；

**算法思想：**利用二叉链表实现基本操作，结合书本与之前所学的单链表知识，最后由switch菜单一一运行。

**运行结果：**运行出现包含20个基本操作的菜单，初始化和创建之后便可以实现各个命令操作的要求。

**结果分析：正确**

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

struct BiTreeQueue

{

BiTree t;

struct BiTreeQueue \*next;

};

//1.构造空二叉树

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

//2.销毁二叉树

Status DestroyBiTree(BiTree &T)

{

if(T==NULL)

return OK;

DestroyBiTree(T->lchild);

DestroyBiTree(T->rchild);

free(T);

return OK;

}

//3.用先序遍历创建二叉树

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

//4.清空二叉树

Status ClearBiTree(BiTree &T)

{

if(!T)

return ERROR;

DestroyBiTree(T->lchild);

DestroyBiTree(T->rchild);

free(T);

T=(BiTree)malloc(sizeof(BiTNode));//重新分配空间

if(!T) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

//5.探空

Status BiTreeEmpty(BiTree T)

{

if(T)

{

if(T->data != NULL)

return FALSE;

}

return TRUE;

}

//6.返回二叉树的深度

Status BiTreeDepth(BiTree T)

{

int leftdepth,rightdepth;

if(T==NULL)

return 0;

else

{

leftdepth = BiTreeDepth(T->lchild );

rightdepth = BiTreeDepth(T->rchild );

return(leftdepth>rightdepth)?(leftdepth+1):(rightdepth+1);

}

}

//7.返回二叉树的根

ElemType Root(BiTree T)

{

if(!T)

return ERROR;

char e;

e=T->data ;

return e;

}

//辅助函数

Status Search(BiTree T,BiTree cur\_p)

{

if(!T)

return FALSE;

if(T==cur\_p||Search(T->lchild,cur\_p)==TRUE||Search(T->rchild,cur\_p)==TRUE)

return TRUE;

else

return FALSE;

}

//8.若cur\_p是T中某个结点，返回cur\_p的值

ElemType Value(BiTree T,BiTree cur\_p)

{

if(!T)

return ERROR;

if(Search(T,cur\_p)==TRUE)

return cur\_p->data;

else

return '.';

}

//9.若cur\_p是T中某个非根结点，则返回它的双亲

BiTree Parent(BiTree T,BiTree cur\_p)

{

if(T==cur\_p)

return NULL;

if(T->lchild==cur\_p||T->rchild==cur\_p)

return T;

else if(Search(T->lchild,cur\_p)==TRUE)

return Parent(T->lchild,cur\_p);

else if(Search(T->rchild,cur\_p)==TRUE)

return Parent(T->rchild,cur\_p);

else

return NULL;

}

//10.若cur\_p是T中某个结点，返回cur\_p的左孩子

BiTree LeftChild(BiTree T,BiTree cur\_p)

{

if(Search(T,cur\_p)==TRUE)

return cur\_p->lchild;

else

return NULL;

}

//11.若cur\_p是T中某个结点，返回cur\_p的右孩子

BiTree RightChild(BiTree T,BiTree cur\_p)

{

if(Search(T,cur\_p)==TRUE)

return cur\_p->rchild;

else

return NULL;

}

//12.cur\_p是T中某个结点，返回cur\_p的左兄弟

BiTree LeftBrother(BiTree T,BiTree cur\_p)

{

if(Search(T,cur\_p)==TRUE&&Parent(T,cur\_p)->lchild!=cur\_p)

return Parent(T,cur\_p)->lchild;

else

return NULL;

}

//13.cur\_p是T中某个结点，返回cur\_p的右兄弟

BiTree RightBrother(BiTree T,BiTree cur\_p)

{

if(Search(T,cur\_p)==TRUE&&Parent(T,cur\_p)->rchild!=cur\_p)

return Parent(T,cur\_p)->rchild;

else

return NULL;

}

//14.先序遍历二叉树，前提T存在

Status PreOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

printf("%c ",T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

return OK;

}

//15.中序遍历二叉树，前提T存在

Status InOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

InOrderTraverse(T->lchild);

printf("%c ",T->data);

InOrderTraverse(T->rchild);

return OK;

}

//16.后序遍历二叉树，前提T存在

Status PostOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%c ",T->data);

return OK;

}

//17.层序遍历二叉树，前提T存在

Status LevelOrderTraverse(BiTree T)

{

BiTNode \*p, \*qu[MAX\_SIZE];

int front,rear;

front=rear=0;

qu[rear]=T;

rear++;

while(front!=rear)

{

p=qu[front];

front = (front+1)%MAX\_SIZE;

printf("%c\t",p->data );

if(p->lchild !=NULL)

{

qu[rear]=p->lchild ;

rear=(rear+1)%MAX\_SIZE;

}

if(p->rchild !=NULL)

{

qu[rear]=p->rchild ;

rear=(rear+1)%MAX\_SIZE;

}

}

}

//18.若cur\_p是T中某个结点，结点cur\_p赋值为value

Status Assign(BiTree T,BiTree &cur\_p,ElemType value)

{

if(Search(T,cur\_p)==TRUE)

{

cur\_p->data=value;

return OK;

}

return ERROR;

}

//19.添加孩子

Status InsertChild(BiTree T,BiTree cur\_p,int LR,BiTree c)

{

if(!T||Search(T,cur\_p)==FALSE)

return ERROR;

BiTree q;

if(LR==0)

{

q=cur\_p->lchild;

cur\_p->lchild=c;

c->rchild=q;

}

else if(LR==1)

{

q=cur\_p->rchild;

cur\_p->rchild=c;

c->rchild=q;

}

else

return ERROR;

return OK;

}

//20.删除孩子

Status DeleteChild(BiTree T,BiTree cur\_p,int LR)

{

if(T&&Search(T,cur\_p)==FALSE)

return ERROR;

if(LR==0)

{

DestroyBiTree(cur\_p->lchild);

cur\_p->lchild=NULL;

}

else if(LR==1)

{

DestroyBiTree(cur\_p->rchild);

cur\_p->rchild=NULL;

}

else

return ERROR;

return OK;

}

BiTree Point(BiTree T,char e)

{ //返回指向e的指针

struct BiTreeQueue \*bt,\*bw,\*a,\*c;

if(T)

{

bt=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

bt->t=T;

bt->next=NULL;

bw=bt;

while(bt)

{

a=bt;

if(a->t->data==e)

return a->t;

if(a->t->lchild)

{

c=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

c->t=a->t->lchild;

c->next=NULL;

bw->next=c;

bw=bw->next;

}

if(a->t->rchild)

{

c=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

c->t=a->t->rchild;

c->next=NULL;

bw->next=c;

bw=bw->next;

}

bt=bt->next;

free(a);

}

}

return NULL;

}

void menu()

{

cout<<"1.初始化 2.销毁"<<endl;

cout<<"3.创建二叉树 4.清空"<<endl;

cout<<"5.判断空树 6.二叉树深度"<<endl;

cout<<"7.根的元素值 8.找某一结点的值"<<endl;

cout<<"9.某一结点的双亲 10.左孩子"<<endl;

cout<<"11.右孩子 12.左兄弟"<<endl;

cout<<"13.右兄弟 14.先序遍历"<<endl;

cout<<"15.中序遍历 16.后序遍历"<<endl;

cout<<"17.层次遍历 18.赋值于某一结点"<<endl;

cout<<"19.添加子树 20.删除子树"<<endl;

}

int main()

{

BiTree T,c;

int choice,LR;

BiTree cur\_p;

char p,ch,value;

while(1)

{

menu();

cout<<"请选择："<<endl;

cin>>choice;

switch(choice)

{

case(1):InitBiTree(T);break;

case(2):DestroyBiTree(T);break;

case(3):CreateBiTree(T);

ch=getchar();//吸收换行符

break;

case(4):ClearBiTree(T);break;

case(5):if(BiTreeEmpty(T))

cout<<"是空树"<<endl;

else

cout<<"不是空树"<<endl;

break;

case(6):cout<<"单链表深度为"<<BiTreeDepth(T)<<endl;break;

case(7):cout<<"二叉树根的元素值为："<<Root(T)<<endl;

break;

case(8):cout<<"该操作为辅助操作，请选择其他操作";

break;

case(9):cout<<"请输入您要查看的结点：";

cin>>p;

cur\_p=Point(T,p);

cout<<p<<"的双亲为"<<Value(T,Parent(T,cur\_p));

break;

case(10):cout<<"请输入您要查看的结点：";

cin>>p;

cur\_p=Point(T,p);

cout<<cur\_p<<"的左孩子为"<<Value(T,LeftChild(T,cur\_p));

break;

case(11):cout<<"请输入您要查看的结点：";

cin>>p;

cur\_p=Point(T,p);

cout<<p<<"的右孩子为"<<Value(T,RightChild(T,cur\_p));

break;

case(12):cout<<"请输入您要查看的结点：";

cin>>p;

cur\_p=Point(T,p);

cout<<p<<"的左兄弟为"<<Value(T,LeftBrother(T,cur\_p));

break;

case(13):cout<<"请输入您要查看的结点：";

cin>>p;

cur\_p=Point(T,p);

cout<<p<<"的右兄弟为"<<Value(T,RightBrother(T,cur\_p));

break;

case(14):cout<<"先序遍历为:"<<endl;

PreOrderTraverse(T);

break;

case(15):cout<<"中序遍历为:"<<endl;

InOrderTraverse(T);

break;

case(16):cout<<"后序遍历为:"<<endl;

PostOrderTraverse(T);

break;

case(17):cout<<"层次遍历为:"<<endl;

LevelOrderTraverse(T);

break;

case(18):cout<<"请输入您要赋值的结点和所赋的值：";

cin>>p;

cin>>value;

cur\_p=Point(T,p);

Assign(T,cur\_p,value);

break;

case(19):cout<<"请输入想要添加到哪个结点下："<<endl;

cin>>p;

cur\_p=Point(T,p);

cout<<"输入0或1选择添加为左子树（0）还是右子树（1）："<<endl;

cin>>LR;

cout<<"建立非空二叉树c与T不相交且右子树为空:"<<endl;

InitBiTree(c);

CreateBiTree(c);

InsertChild(T,cur\_p,LR,c);

break;

case(20):cout<<"请输入想要删除哪个结点下的子树："<<endl;

cin>>p;

cur\_p=Point(T,p);

cout<<"输入0或1选择删除左子树（0）还是右子树（1）："<<endl;

cin>>LR;

DeleteChild(T,cur\_p,LR);

break;

}

cout<<endl;

}

}

**2、**

**题目：**实现二叉树的先序、中序、后序遍历，用递归和非递归方法；实现层次遍历。

**算法思想：**结合第一题的基本操作，直接实现递归遍历，非递归的遍历利用栈结构实现。

**运行结果：**运行输入二叉树，依次输出题目所要求的遍历。

**结果分析：**正确

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

#define STACKINITSIZE 256

#define STACKINCREMENT 128

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

typedef struct SeqStack

{

BiTNode \*pBase;

BiTNode \*pTop;

int stacksize;

}SeqStack;

Status InitStack(SeqStack &S)

{

S.pBase = (BiTNode\*)malloc(STACKINITSIZE\*sizeof(int));

if (!S.pBase)

exit (OVERFLOW);

S.pTop = S.pBase;

S.stacksize = STACKINITSIZE;

return OK;

}

Status StackEmpty(SeqStack S)

{

if (S.pTop == S.pBase )

{

return TRUE;

}

else

{

cout<<" ";

return FALSE;

}

}

Status Push(SeqStack &S , BiTree e)

{

if(S.pTop - S.pBase >=S.stacksize )

{

S.pBase = (BiTNode\*)realloc(S.pBase ,(S.stacksize + STACKINCREMENT)\*sizeof(int));

if(S.pBase ==NULL) exit(OVERFLOW);

S.pTop = S.pBase + S.stacksize ;

S.stacksize += STACKINCREMENT;

}

\*S.pTop = \*e;

S.pTop ++;

return OK;

}

Status Pop(SeqStack &S)

{

if(S.pTop == S.pBase ) return ERROR;

S.pTop--;

return OK;

}

BiTree GetTop(SeqStack &S )

{

BiTree e;

if(S.pTop == S.pBase ) return ERROR;

e = S.pTop -1;

return e;

}

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

//先序遍历二叉树，前提T存在

Status PreOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

printf("%c ",T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

return OK;

}

//先序遍历非递归

Status PreOrderTraverseStack(BiTree T)

{

SeqStack S;

BiTNode \*p;

p=T;

InitStack(S);

if(T)

Push(S,T );

while(!StackEmpty(S))

{

cout<<p->data ;

if(p->rchild )

Push(S,p->rchild);

if(p->lchild )

p=p->lchild ;

else

{

p=GetTop(S);

Pop(S);

}

}

cout<<endl;

}

//中序遍历二叉树，前提T存在

Status InOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

InOrderTraverse(T->lchild);

printf("%c ",T->data);

InOrderTraverse(T->rchild);

return OK;

}

//中序遍历非递归

Status InOrderTraverseStack(BiTree T)

{

SeqStack S;

BiTNode \*p;

p=T;

InitStack(S);

while(!StackEmpty(S)||p)

{

if(p)

{

Push(S,p);

p=p->lchild ;

}

else

{

p=GetTop(S);

Pop(S);

cout<<p->data ;

p=p->rchild ;

}

}

cout<<endl;

}

//后序遍历二叉树，前提T存在

Status PostOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%c ",T->data);

return OK;

}

/\*//后序遍历非递归

Status PostOrderTraverseStack(BiTree T)

{

SeqStack S;

BiTNode \*p,\*q;

p=T;

q = NULL;//先把p移动到左子树最下边

InitStack(S);

if (! T)

return ERROR;

while (p)

{

Push(S,p);

p = p->lchild;

}

while (!StackEmpty(S))

{

p = GetTop(S);

Pop(S);

if (p->rchild == NULL || p->rchild == q)

{

cout <<p->data;

q = p;

}

else

{

Push(S,p);

p = p->rchild;

while (p)

{

Push(S,p);

cout<<endl;

p = p->lchild;

}

}

}

cout << endl;

}\*/

//层序遍历二叉树，前提T存在

Status LevelOrderTraverse(BiTree T)

{

BiTNode \*p, \*qu[MAX\_SIZE];

int front,rear;

front=rear=0;

qu[rear]=T;

rear++;

while(front!=rear)

{

p=qu[front];

front = (front+1)%MAX\_SIZE;

printf("%c\t",p->data );

if(p->lchild !=NULL)

{

qu[rear]=p->lchild ;

rear=(rear+1)%MAX\_SIZE;

}

if(p->rchild !=NULL)

{

qu[rear]=p->rchild ;

rear=(rear+1)%MAX\_SIZE;

}

}

}

int main()

{

BiTree T;

cout<<"请输入需要遍历的二叉树："<<endl;

InitBiTree(T);

CreateBiTree(T);

cout<<"递归先序遍历：";

PreOrderTraverse(T);

cout<<endl;

cout<<"非递归先序遍历：";

PreOrderTraverseStack(T);

cout<<endl;

cout<<"递归中序遍历：";

InOrderTraverse(T);

cout<<endl;

cout<<"非递归中序遍历：";

InOrderTraverseStack(T);

cout<<endl;

cout<<"递归后序遍历：";

PostOrderTraverse(T);

cout<<endl;

/\* cout<<"非递归后序遍历：";

PostOrderTraverseStack(T);

cout<<endl; \*/

cout<<"层次遍历：";

LevelOrderTraverse(T);

cout<<endl;

}

**3、**

**题目：**设二叉树采用二叉链表存储，编写函数，对二叉树中每个元素值为x的结点，删除以它为根的子树，并释放相应空间。（习题集6.45）

**算法思想：**结合第一题的基本操作中删除左或右子树的函数，在这题中删除完整子树。

**运行结果：**运行输入二叉树和想要删除的结点，输出删除后的子树。

**结果分析：**正确

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

struct BiTreeQueue

{

BiTree t;

struct BiTreeQueue \*next;

};

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

Status DestroyBiTree(BiTree &T)

{

if(T==NULL)

return OK;

DestroyBiTree(T->lchild);

DestroyBiTree(T->rchild);

free(T);

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

Status Search(BiTree T,BiTree cur\_p)

{

if(!T)

return FALSE;

if(T==cur\_p||Search(T->lchild,cur\_p)==TRUE||Search(T->rchild,cur\_p)==TRUE)

return TRUE;

else

return FALSE;

}

Status DeleteChild(BiTree T,BiTree cur\_p)

{

if(T&&Search(T,cur\_p)==FALSE)

return ERROR;

DestroyBiTree(cur\_p->lchild);

cur\_p->lchild=NULL;

DestroyBiTree(cur\_p->rchild);

cur\_p->rchild=NULL;

return OK;

}

Status PreOrderTraverse(BiTree T)

{

if(!T)

return ERROR;

printf("%c ",T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

return OK;

}

BiTree Point(BiTree T,char e)

{ //返回指向e的指针

struct BiTreeQueue \*bt,\*bw,\*a,\*c;

if(T)

{

bt=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

bt->t=T;

bt->next=NULL;

bw=bt;

while(bt)

{

a=bt;

if(a->t->data==e)

return a->t;

if(a->t->lchild)

{

c=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

c->t=a->t->lchild;

c->next=NULL;

bw->next=c;

bw=bw->next;

}

if(a->t->rchild)

{

c=(struct BiTreeQueue \*)malloc(sizeof(struct BiTreeQueue));

c->t=a->t->rchild;

c->next=NULL;

bw->next=c;

bw=bw->next;

}

bt=bt->next;

free(a);

}

}

return NULL;

}

int main()

{

BiTree T,cur\_p;

char p;

InitBiTree(T);

cout<<"请输入二叉树（“.”表示空）"<<endl;

CreateBiTree(T);

cout<<"请输入想要删除哪个结点下的子树："<<endl;

cin>>p;

cout<<"删除前先序遍历二叉树："<<endl;

PreOrderTraverse(T);

cout<<endl;

cur\_p=Point(T,p);

DeleteChild(T,cur\_p);

cout<<"删除后先序遍历二叉树："<<endl;

PreOrderTraverse(T);

}

**4、**

**题目：**编写函数，判断给定的二叉树是否是完全二叉树。（习题集6.49）

**算法思想：**结合第一题的基本操作和书本代码判断。

**运行结果：**运行输入二叉树，判断是否是完全二叉树。

**结果分析：**正确

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

#define STACKINITSIZE 256

#define STACKINCREMENT 128

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

Status IsComplete(BiTree T)

{

BiTNode \*p,\*q[MAX\_SIZE];

int front=0,rear=0;

int n1=0;

q[rear++]=T;

while(front!=rear)

{

p=q[front];

front = (front+1)%MAX\_SIZE;

if(!p->lchild && !p->rchild )

n1=1;

else if(!p->lchild && p->rchild )

return FALSE;

else if(p->lchild && !p->rchild )

{

if(n1!=0)

return FALSE;

else

{

n1=1;

q[rear]=p->lchild ;

rear=(rear+1)%MAX\_SIZE;

}

}

else

{

if(n1)

return FALSE;

q[rear]=p->lchild ;

rear = (rear+1)%MAX\_SIZE;

q[rear]=p->rchild ;

rear = (rear+1)%MAX\_SIZE;

}

}

return true;

}

int main()

{

BiTree T;

InitBiTree(T);

cout<<"请输入二叉树（“.”表示空）"<<endl;

CreateBiTree(T);

if(IsComplete(T))

cout<<"是完全二叉树"<<endl;

else

cout<<"不是完全二叉树"<<endl;

}

**二、未调试成功程序及说明**

**1、**

**题目：**实现二叉树的后序遍历，用非递归方法。

**算法思想：**利用栈结构，利用入栈出栈的先后顺序实现。

**错误原因：**在运行到一处p指针进栈时会自动改变q指针所指，未能找出原因。

**附源程序。**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

#include<iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define MAX\_SIZE 256

#define STACKINITSIZE 256

#define STACKINCREMENT 128

typedef char ElemType;

typedef int Status;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

}BiTNode,\*BiTree;

typedef struct SeqStack

{

BiTNode \*pBase;

BiTNode \*pTop;

int stacksize;

}SeqStack;

Status InitStack(SeqStack &S)

{

S.pBase = (BiTNode\*)malloc(STACKINITSIZE\*sizeof(int));

if (!S.pBase)

exit (OVERFLOW);

S.pTop = S.pBase;

S.stacksize = STACKINITSIZE;

return OK;

}

Status StackEmpty(SeqStack S)

{

if (S.pTop == S.pBase )

{

return TRUE;

}

else

{

cout<<" ";

return FALSE;

}

}

Status Push(SeqStack &S , BiTree e)

{

if(S.pTop - S.pBase >=S.stacksize )

{

S.pBase = (BiTNode\*)realloc(S.pBase ,(S.stacksize + STACKINCREMENT)\*sizeof(int));

if(S.pBase ==NULL) exit(OVERFLOW);

S.pTop = S.pBase + S.stacksize ;

S.stacksize += STACKINCREMENT;

}

\*S.pTop = \*e;

S.pTop ++;

return OK;

}

Status Pop(SeqStack &S)

{

if(S.pTop == S.pBase ) return ERROR;

S.pTop--;

return OK;

}

BiTree GetTop(SeqStack &S )

{

BiTree e;

if(S.pTop == S.pBase ) return ERROR;

e = S.pTop -1;

return e;

}

Status InitBiTree(BiTree &T)

{

T=(BiTree)malloc(sizeof(BiTNode));

if(T==NULL) exit(OVERFLOW);

T->lchild=NULL;

T->rchild=NULL;

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

cin>>ch;

if(ch=='.')//.表示空

T = NULL;

else

{

T=(BiTNode\*)malloc(sizeof(BiTNode));

if(!T) exit(OVERFLOW);

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

//后序遍历非递归

Status PostOrderTraverseStack(BiTree T)

{

SeqStack S;

BiTNode \*p,\*q;

p=T;

q = NULL;//先把p移动到左子树最下边

InitStack(S);

if (! T)

return ERROR;

while (p)

{

Push(S,p);

p = p->lchild;

}

while (!StackEmpty(S))

{

p = GetTop(S);

Pop(S);

if (p->rchild == NULL || p->rchild == q)

{

cout <<p->data;

q = p;

}

else

{

Push(S,p);

p = p->rchild;

while (p)

{

Push(S,p);

cout<<endl;

p = p->lchild;

}

}

}

cout << endl;

}

int main()

{

BiTree T;

cout<<"请输入需要遍历的二叉树："<<endl;

InitBiTree(T);

CreateBiTree(T);

cout<<"非递归后序遍历：";

PostOrderTraverseStack(T);

cout<<endl;

cout<<endl;

}

**2、**

**题目：**CSP题目 灰度图像

**算法思想：**利用二维数组，算出邻域元素，遍历暴力求解。

**错误原因：**结果不对

**附源程序。**

#include<iostream>

using namespace std;

int main()

{

int n,L,r,t;

int a,b,c,d;

int sum=0,num=0,grey=0;

float avg;

cin>>n>>L>>r>>t;

int A[n][n];

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

cin>>A[i][j];

}

}

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

a=i-r;

if(a<0) a=0;

b=j-r;

if(b<0) b=0;

c=i+r;

if(c>=n) c=n-1;

d=j+r;

if(d>=n) d=n-1;

sum=0;

num=0;

for(int x=a;x<=c;x++)

{

for(int y=b; y<=d; y++)

{

sum=sum+A[x][y];

num++;

}

}

avg=sum/num;

if(avg<=t)

grey++;

}

}

cout<<grey;

}

**正确代码：**

#include<iostream>

using namespace std;

int main()

{

int i,j,n,L,r,t,t1,t2,grey=0;

int a,b,c,d;//邻域左上元素为A[a][b]，右下元素为A[c][d]

int a1,b1,c1,d1;

float sum,num;

cin>>n>>L>>r>>t;

int A[n][n];

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cin>>A[i][j];

}

}

for(i=0;i<n;i++)

{

j=0;

sum=0;

num=0;

a=i-r;

if(a<0) a=0;

b=j-r;

if(b<0) b=0;

c=i+r;

if(c>=n) c=n-1;

d=j+r;

if(d>=n) d=n-1;

for(t1=a;t1<c+1;t1++)

{

for(t2=b;t2<d+1;t2++)

{

sum+=A[t1][t2];

num++;

}

}

if(sum/num<=t)

grey++;

for(j=1;j<n;j++)

{

a1=i-r;if(a1<0) a1=0;

b1=j-r;if(b1<0) b1=0;

c1=i+r;if(c1>=n) c1=n-1;

d1=j+r;if(d1>=n) d1=n-1;

if(b1!=b)

{

for(t1=a;t1<c+1;t1++)

{

sum-=A[t1][b];

num--;

}

}

if(d1!=d){

for(t1=c;t1>=a;t1--){

sum+=A[t1][d1];

num++;

}

}

if(sum/num<=t) grey++;

a=a1;b=b1;c=c1;d=d1;

}

}

cout<<grey;

return 0;

}

**三、代码行数及小结**

行数：合计1000行左右

小结：此次上机重点在于二叉树，与之前的单链表有很多联系之处，并运用到了栈结构，有一定的书本给出的代码，要求我们实现。