上机一二和第六章课本习题

**吴叶赛 U201813405 电信1806**

**上机一**

1．单链表的逆置

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char data;

node \*next;

}node;

typedef node \*LinkList;

LinkList build() //单链表尾插法

{

node \*p,\*head;

int i;

head=new(node);

p=head;

for( i=1; i<=7; i++)

{

p->next=new(node);

p->next->data=i+'a'-1;

p=p->next;

}

p->next=NULL;

return(head);

}

void display(LinkList L)

{

node \*p;

p=L->next;

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void reverse(LinkList &L)

{

node \*p,\*q,\*r;

p=L->next;

if(p==NULL||p->next==NULL)

{printf("No reverse needed!\n");

return;

}

q=p->next;

p->next=NULL;

while(q!=NULL)

{

r=q->next;

q->next=p;

p=q;

q=r;

}

L->next=p;

}

int main()

{

LinkList L1;

L1=build();

display(L1);

reverse(L1);

display(L1);

}

2.单链表找中点

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void findmiddle(LinkList &L)

{

node \*p,\*q;

p=L->next;

q=L->next;

while(q->next!=NULL&&q!=NULL)

{

p=p->next;

q=q->next->next;

}

if(q->next==NULL)

{

printf("中点值为%c\n",p->data);

}

if(q==NULL)

{

printf("没有中点");

}

L->next=p;

}

删除倒数第k个元素

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char data;

node \*next;

}node;

typedef node \*LinkList;

LinkList build() //单链表尾插法

{

node \*p,\*head;

int i;

head=new(node);

p=head;

for( i=1; i<=7; i++)

{

p->next=new(node);

p->next->data=i+'a'-1;

p=p->next;

}

p->next=NULL;

return(head);

}

void display(LinkList L)

{

node \*p;

p=L->next;

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void findk(LinkList &L,int n)

{

node \*p,\*q;

p=L->next;

q=L->next;

int i;

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

{

p=p->next;

}

while(p->next!=NULL)

{

p=p->next;

q=q->next;

}

printf("倒数第%d个元素是%c",n+1,q->data);

}

int main()

{

int k;

LinkList L1;

L1=build();

display(L1);

printf("请问想要找到倒数第几个元素？");

scanf("%d",&k);

k--;

findk(L1,k);

}

删除倒数第k个元素

void deletek(LinkList &L,int n)

{

node \*p,\*q,\*r;

p=L->next;

q=L->next;

int i;

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

{

p=p->next;

}

while(p->next!=NULL)

{

p=p->next;

q=q->next;

}

r=q->next;

q->next=r->next;

}

4.判断单链表是否有环 如果有环 返回交点

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char data;

node \*next;

}node;

typedef node \*LinkList;

LinkList build() //单链表尾插法

{

node \*p,\*head,\*r;

int i;

head=new(node);

p=head;

r=head;

for( i=1; i<=7; i++)

{

p->next=new(node);

p->next->data=i+'a'-1;

p=p->next;

}

for(i=1;i<=3;i++)

{

r=r->next;

}

p->next=r;

return(head);

}

void display(LinkList L)

{

node \*p;

p=L->next;

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void circle(LinkList &L)

{

node \*p,\*q;

p=L->next;

q=L->next;

while(q->next!=NULL&&q!=p)

{

p=p->next;

q=q->next->next;

}

if(q->next==NULL)

{

printf("单链表不存在环");

}

else if(q==p)

{

p=L->next;

while(p!=q)

{

p=p->next;

q=q->next;

}

q=q->next;

printf("链表中有环存在\n");

printf("交点处的元素是:%c",q->data);

}

}

int main()

{

LinkList L1;

L1=build();

circle(L1);

}

5、判断两个单链表是否相交，如果相交，找出交点

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char data;

node \*next;

}node;

typedef node \*LinkList;

LinkList build() //单链表尾插法

{

node \*p,\*head;

int i;

head=new(node);

p=head;

for( i=1; i<=7; i++)

{

p->next=new(node);

p->next->data=i+'a'-1;

p=p->next;

}

p->next=NULL;

return(head);

}

LinkList build1(LinkList L)

{

node \*p,\*head,\*q;

int i;

q=L->next;

for(i=0;i<3;i++)

{

q=q->next;

}

head=new(node);

p=head;

p->next=q;

return(head);

}

void display(LinkList L)

{

node \*p;

p=L->next;

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void intersect(LinkList &L,LinkList &H,node\* &m)

{

node \*p,\*q,\*r;

int a=0;

int b=0;

int n;

p=L->next;

q=H->next;

while(p->next!=NULL)

{

a++;

p=p->next;

}

while(q->next!=NULL)

{

b++;

q=q->next;

}

if(p=q)

{

p=L->next;

q=H->next;

printf("两个列表存在交点\n");

if(a>b)

{

for(n=0;n<a-b;n++)

{

p=p->next;

}

while(p!=q)

{

p=p->next;

q=q->next;

}

m=q;

}

if(a=b)

{

while(p!=q)

{

p=p->next;

q=q->next;

}

m=q;

}

if(a<b)

{

for(n=0;n<b-a;n++)

{

q=q->next;

}

while(p!=q)

{

p=p->next;

q=q->next;

}

m=q;

}

}

if(p!=q)

{

printf("两个单链表不存在交点\n");

}

}

int main()

{

node \*m;

LinkList L1,L2;

L1=build();

L2=build1(L1);

display(L1);

display(L2);

intersect(L1,L2,m);

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

}

删除有序链表之中的重复节点：

void deletesame(LinkList &L)

{

node \*p,\*q;

q=L->next;

while(q!=NULL)

{

q=q->next;

if(q->data==q->next->data)

{

p=q->next;

q->next=p->next;

free(p);

}

}

}

7.约瑟夫环

void circle(LinkList L,int k,int m,char a[])

{

node \*p,\*q;

int i,j;

j=0;

p=L->next;

for(i=0;i<k-1;i++)

{

p=p->next;

}

while(p!=NULL)

{

for(i=0;i<m-1;i++)

{

p=p->next;

}

q=p->next;

p->next=q->next;

a[j]=q->data;

printf("%c",a[j]);

p=p->next;

free(q);

j++;

}

}

8. 9、合并两个升序单链表(保留重复点)，合并后为升序

void combine(LinkList &A,LinkList &B,LinkList &C)

{

node \*pa,\*qa,\*pb,\*qb,\*m;

pa=A->next;

pb=B->next;

qa=A;

qb=B;

A->next=NULL;

m=A;

C=A;

while(pa&&pb)

{

if(pa->data<=pb->data)

{

qa=pa;

pa=pa->next;

m->next=qa;

m=m->next;

m->next=NULL;

}

else

{

qb=pb;

pb=pb->next;

m->next=pb;

m=m->next;

m->next=NULL;

}

}

while(pa)

{

qa=pa;

pa=pa->next;

m->next=qa;

m=m->next;

m->next=NULL;

}

while(pb)

{

qb=pb;

pb=pb->next;

m->next=pb;

m=m->next;

m->next=NULL;

}

}

9、合并两个升序单链表(保留重复点)，合并后为降序

void sort(LinkList &A,LinkList &B,LinkList &C)

{

node \*pa,\*qa,\*pb,\*qb;

pa=A;

pa=B;

qa=pa;

qb=pb;

pa=A->next;

pb=B->next;

A->next=NULL;

C=A;

while(qa&&qb)

{

if(pa->data<pb->data)

{

qa=pa;

pa=pa->next;

qa->next=A->next;

A->next=qa;

}

else

{

qb=pb;

pb=pb->next;

qb->next=A->next;

A->next=qb;

}

}

while(pa)

{

qa=pa;

pa=pa->next;

qa->next=A->next;

A->next=qa;

}

while(pb)

{

qb=pb;

pb=pb->next;

qb->next=A->next;

A->next=qb;

}

pb=B;

free(pb);

}

10.判断链表是否对称

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char data;

node \*next;

}node;

typedef node \*LinkList;

LinkList build() //单链表尾插法

{

node \*p,\*head;

int i;

head=new(node);

p=head;

for( i=1; i<=3;i++)

{

p->next=new(node);

p->next->data=i+'a'-1;

p=p->next;

}

for(i=4;i<=7;i++)

{

p->next=new(node);

p->next->data='a'-i+7;

p=p->next;

}

p->next=NULL;

return(head);

}

void display(LinkList L)

{

node \*p;

p=L->next;

while (p!=NULL)

{

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

p=p->next;

}

printf("\n");

}

void reverse(LinkList &L)

{

node \*p,\*q,\*r;

p=L->next;

if(p==NULL||p->next==NULL)

{printf("No reverse needed!\n");

return;

}

q=p->next;

p->next=NULL;

while(q!=NULL)

{

r=q->next;

q->next=p;

p=q;

q=r;

}

L->next=p;

}

int compare(LinkList &A,LinkList &B)

{

node \*p,\*q;

p=A->next;

q=B->next;

while(p!=NULL)

{

if(p->data!=q->data||p->next==NULL)

{

break;

}

p=p->next;

q=q->next;

}

if(p->next==NULL)

return 1;

else

return 0;

}

int main()

{

int i;

LinkList L1;

LinkList L2;

L1=build();

L2=build();

display(L1);

reverse(L2);

i=compare(L1,L2);

if(i==1)

printf("链表是对称的");

else

printf("链表不对称");

}

**上机二**

0.

* 递归方法建立二叉树；

void createBiTree(BiTree &T)

{

char c;

cin >> c;

if('#' == c)

T = NULL;

else

{

T = new BiTreeNode;

T->data = c;

createBiTree(T->leftChild);

createBiTree(T->rightChild);

}

}

* 非递归方法前序遍历二叉树；

//先序遍历非递归算法的实现 用到自定义的栈

void preOrderNonrecursion(BTNode \*bt)

{

if(bt !=null)

{

BTNode \*stack[maxSize]; //先定义一个栈用于存放遍历的二叉树

int top=-1;

BTNode \*P;

stack[++top]=bt; //元素进栈

while(top!=-1) //进栈不空条件下执行

{

p=stack[top--]; //出栈操作

visit(p); //事先定义好的打印节点的操作

if(p->rchild!=null) //先右子树进栈

stack[++top]=p->rchild;

if(p->lchild!=null) //再左子树进栈

stack[++top]=p->lchild;

}

}

}

* 递归方法前序、中序、后序遍历二叉树

#include<stdio.h>

#define ElementType char

//树的结点的定义

typedef struct BTNode

{

ElementType data;//存放数据

struct BTNode \*lchild;//左孩子

struct BTNode \*rchild;//右孩子

}BTNode;

void Preorder(BTNode \* p);

void Inorder(BTNode\* p);

void Postorder (BTNode\* p);

int main(void)

{

return 0;

}

//前序遍历(递归方法)

void Preorder(BTNode\* p)

{

if(p != NULL)

{

Visit(p->data);

Preorder(p->lchild);

Preorder(p->rchild);

}

}

//中序遍历(递归方法)

void Inorder(BTNode\* p)

{

if(p != NULL)

{

Inorder(p->lchild);

Visit(p->data);

Inorder(p->rchild);

}

}

//后序遍历(递归方法)

void Postorder (BTNode\* p)

{

if(p != NULL)

{

Inorder(p->lchild);

Inorder(p->rchild);

Visit(p->data);

}

}

//访问元素

void Visit(ElementType data)

{

printf("\s ", data);

}

1. 将二叉树对称交换，即求二叉树镜像

void wys(node \*root)

{

node \*temp;

if(root==NULL)

return;

if(root->lchild==NULL||root->rchild==NULL)

return;

while(root!=NULL)

{

temp=root->lchild;

root->lchild=root->rchild;

root->rchild=temp;

wys(root->lchild);

wys(root->rchild);

}

}

1. 求二叉树的高度，即深度

int deep(node \*root)

{

int a, b;

if(root!=NULL)

{

a=deep(root->lchild);

b=deep(root->rchild);

if(a>=b)

return a+1;

else

return b+1;

}

else

return 0;

}

1. 已知前序序列和中序序列，构造二叉树

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

typedef char ElementType ;

typedef struct node

{

ElementType data ;

struct node \* leftChild ;

struct node \* rightChild ;

}BTNode;

//pre：存放先序序列 in：存放中序序列

BTNode \*createBT(char \*pre , char \*in ,int n)

{

BTNode \*b;

char \*p ;

int k ;

if(n<=0)

return NULL;

b=(BTNode \*)malloc(sizeof(BTNode));

b->data = \*pre ;

int j=0;

for(p=in;p<in+n;p++)

{

if(\*p == \*pre)

break;

}

k=p-in; //确定根节点在中序序列(in)中的位置 编号为0,1,2,...,n-1 (类似于数组中的下标号，不是逻辑序号)

b->leftChild = createBT(pre+1,in,k); //递归构造左子数

b->rightChild = createBT(pre+1+k,p+1,n-k-1); //递归构造右子树

return b ;

}

//先序遍历二叉树BinaryTree:先遍历根节点接着遍历左子树，最后遍历右子树

//(不是左孩子节点和右孩子节点，概念要分清哦(虽然节点也是一个树))

void showBTPreOrder(BTNode \*b)

{

if(b != NULL)

{

//遍历根节点

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

//遍历左子树

showBTPreOrder(b->leftChild);

//遍历右子树

showBTPreOrder(b->rightChild);

}

}

//中序遍历二叉树BinaryTree:先遍历左子树，接着遍历根节点，左后遍历右子树

//(不是左孩子节点和右孩子节点，概念要分清哦(虽然节点也是一个树))

void showBTInOrder(BTNode \*b)

{

if(b!=NULL)

{

//遍历左子树

showBTInOrder(b->leftChild);

//遍历根节点

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

//遍历右子树

showBTInOrder(b->rightChild);

}

}

int main()

{

BTNode \*b = NULL ;

char pre[] = "ABDGCEF";

char in[] = "DGBAECF" ;

b=createBT(pre,in,7); //先序遍历遍历二叉树

printf("先序遍历遍历二叉树:\n");

showBTPreOrder(b);

printf("\n"); //中序遍历遍历二叉树

printf("中序遍历遍历二叉树:\n");

showBTInOrder(b);

printf("\n");

return 0 ;

}

1. 分层遍历二叉树，即按层次从上往下，从左往右顺序访问

void Algo\_6\_47(node \*root)

{

int i, j;

bitree p[100]; //树指针数组，用来模拟队列

i = j = 0;

if(root)

p[j++] = root;

while(i<j)

{

printf("%c ", p[i]->data);

if(p[i]->lchild)

p[j++] = p[i]->lchild;

if(p[i]->rchild)

p[j++] = p[i]->rchild;

i++;

}

}

1. 求二叉树的宽度，即求最大结点数的层所具有的结点数

int count[100];

int MAX=-1;

void FindWidth(BitNode T,int k){

if(T==NULL)

return;

[k]++;

if(MAX<count[k])

MAX=count[k];

FindWidth(T->lchild,k+1);

FindWidth(T->rchild,k+1);} //MAX即为所求宽度

1. 二叉树中的结点个数

int Algo\_6(node \*root)

{

int a,b;

if(root==NULL)

return 0;

else

{

a=Algo\_6(root->lchild);

b=Algo\_6(root->rchild);

return a+b+1;

}

}

1. 非递归方法中序、后序遍历二叉树

8.求二叉树第K层的结点个数

int count[100];

int MAX=-1;

void FindWidth(BitNode T,int k){

if(T==NULL)

return;

[k]++;

FindWidth(T->lchild,k+1);

FindWidth(T->rchild,k+1);}

9. 求二叉树中叶子结点的个数

int Algo\_9(node \*root)

{

int a=0;

if(root==NULL)

return 0;

else

{

if(root->lchild==NULL&&root->rchild==NULL)

a++;

else

{

a+=Algo\_9(root->lchild);

a+=Algo\_9(root->rchild);

}

return a;

}

}

10. 判断两棵二叉树是否结构相同

bool Algo\_10(node \*root1,node \*root2)

{

if(root1==NULL&&root2==NULL)

return true;

if(!root1||!root2)

return false;

if(root1&&root2)

{

return Algo\_10(root1->lchild,root2->rchild)&&Algo\_10(root1->rchild,root2->rchild);

}

else

return false;

}

11.求二叉树中两个结点的最低公共祖先结点

Status FindPath\_6\_48(BiTree T, TElemType p, BiTree path[100])

{

BiTNode node[100];

int i = 0;

if(T==NULL)

return ERROR;

path[i] = T;

node[i] = \*T;

while(i>=0)

{

while(node[i].lchild || path[i]->data==p)

{

if(path[i]->data==p) //寻路成功

return OK;

else

{

path[i+1] = node[i].lchild;

node[i+1] = \*(node[i].lchild);

node[i].lchild = NULL;

i++;

}

}

if(node[i].rchild)

{

path[i+1] = node[i].rchild;

node[i+1] = \*(node[i].rchild);

node[i].rchild = NULL;

i++;

}

else

{

path[i] = NULL;

i--;

}

}

if(i<0) //寻路失败

return ERROR;

}

BiTree Algo\_6\_48(BiTree T, TElemType p, TElemType q) //p、q设为具体的元素，原理不变

{

BiTree path\_1[100] = {};

BiTree path\_2[100] = {};

int k, x;

if(FindPath\_6\_48(T, p, path\_1) && FindPath\_6\_48(T, q, path\_2)) //借助于路径寻找函数

{

k = 0;

if(path\_1[k]->data!=p && path\_2[k]->data!=q)

{

while(path\_1[k]->data==path\_2[k]->data)

k++;

return path\_1[k-1];

}

}

return NULL;

}

第六章课本习题

