树与二叉树代码题

- 假设二叉树采用二叉链表存储结构，设计一个非递归算法求二叉树的高度。

思路

层次遍历，记录高度

代码

```
int Btdepth (BiTree T)
{
	if(!T)
		return 0;
	int front =-1,rear=-1;
	int last=0,level=0;
	BiTree Q[MaxSize];
	BiTree p;
	while(front<rear){
		p=Q[++front];
		if(p->lchild)
			Q[++rear]=p->rchild;
		if(p->rchild)
			Q[++rear]=p->rchild;
		if(front==last)
		{
			level++;
			last=rear;
		}
	}
	return level;
}
```

- 二叉树按二叉链表形式存储，试编写一个判别给定二叉树是否是完全二叉树的算法，

思路

根据完全二叉树的定义，具有n个结点的完全二叉树与满二叉树中编号从1~n的结点一一对应。算法思想：采用层次遍历算法，将所有结点加入队列（包括空结点）。遇到空结点时，查
看其后是否有非空结点。若有，则二叉树不是完全二叉树。

代码

```
bool IsComplete(BiTree T)
{
	InitQueue(Q);
	if(!T)
	{
		return true;
	}
	EnQueue(Q,T);
	while(!IsEmpty)
	{
		DeQueue(Q,p);
		if(p)
		{
			EnQueue(Q,p->lchild);
			EnQueue(Q,p->rchild);
		}
		else
			while(!IsEmpty(Q))
			{
				DeQueue(Q,p);
				if(p)
					return false;
			}
	}
	return true;
}
```

- 假设二叉树采用二叉链表存储结构存储，试设计一个算法，计算一棵给定二叉树的所有双分支结点个数。

思路

递归

代码

```
int DsonNodes(BiTree b)
{
	if(b=NULL)
		return 0;
	else if(b->lchild!=NULL&&b->rchild!=NULL)
		return DosnNodes(b->lchild)+DosonNodes(b->rchild)+1;
	else
```

```
        return DosnNodes(b->lchild)+DosonNodes(b->rchild);
}
```

- 设树B是一棵采用链式结构存储的二叉树，编写一个把树B中所有结点的左、右子树进行交换的函数。

思路

递归交换

代码

```
void swap(BiTree b)
{
    if(b)
    {
        swap(b->lchild);
        swqp(b->rchild);
        temp=b->lchild;
        b->rchild=b->lchild;
        b->lchild=temp;
    }
}
```

- 假设二叉树采用二叉链存储结构存储，设计一个算法，求先序遍历序列中第k(1≤k≤二叉树中结点个数)个结点的值，

思路

递归加一

代码

```
int i=1;
ElemType PreNode(BiTree b,int k)
{
    if(b==NULL)
        return "#";
    if(i==k)
        return b->data;
    i++;
    ch=PreNode(b->lchild,k);
```

```
        if(ch!="#")
                return ch;
        ch=PreNode(b->rchild,k);
                return ch;
}
```

- 已知二叉树以二叉链表存储，编写算法完成：对于树中每个元素值为x的结点，别除以它为根的子树，并释放相应的空间

思路

遍历

代码

```
void DeleteXTree(BiTree &bt)
{
        if(bt){
                DeLeteXTree(bt->lchild);
                DeLeteXTree(bt->rchild);
                free(bt);
        }
}
void Search(BiTree bt,ElemType x)
{
        BiTree Q[];
        if(bt){
                if(bt->data==x){
                        DeleteXTree(bt);
                        exit(0);
                }
                InitQueue(Q);
                EnQueue(Q,bt);
                while(!IsEmpty(Q)){
                        DeQueu(Q,p);
                        if(p->lchild){
                                if(p->lchild==x)
                                {
                                        DeleteXTree(p->child);
                                        p->child=NULL;
                                }
                                else
                                        EnQueue(Q,p->lchild);
                        }
                        if(p->rchild){
```

```
                        if(p->rchild==x){
                                DeleteXTree(p->rchild);
                                p->rchild==NULL;
                        }
                        else
                                EnQueue(Q,p->rchild);
                }
        }

    }
}
```

- 在二叉树中查找值为x的结点，试编写算法（用C语言）打印值为x的结点的所有祖先，假设值为x的结点不多于一个。

思路

算法思想：采用非递归后序遍历，最后访问根结点，访问到值为x的结点时，栈中所有元素
均为该结点的祖先，依次出栈打印即可。

代码

```
typedef struct {
 BiTree t;
 int tag;
}stack;
void Search(BiTree bt,ElemType x)
{
        stack s[ ];
        top=0;
        while(bt!=NULL||top>0)
        {
                while(bt!=NULL&&bt->data!=x){
                        s[++top].t=bt;
                        s[top].tag=0;
                        bt=bt->lchild;
                }
                if(bt!=NULL&&bt->data==x)
                {
                        printf("所查找节点的所有祖先节点的值为:\n");
                        for(i=1;i<=top;i++)
                        printf("%d",s[i].t->data);
                        exit(1);
                }
```

```
            while(top!=0&&s[top].tag==1){
                top--;
            }
            if(top!=0){
                s[top].tag=1;
                bt=s[top].t->rchild;
            }
        }
}
```

- 设一棵二叉树的结，点结构为(LLINK,INEO,RLINK),ROOT为指向该二叉树根结点的指针，p和q分别为指向该二叉树中任意两个结，点的指针，试编写算法ANCESTOR(ROOT,p,q,r),找到p和q的最近公共祖先结点r.

思路

后续遍历，压栈比较

代码

```
typedef struct{
        BiTree t;
        int tag==0;
}stack;
stack s[ ],s1[ ];
BiTree Ancestor(BiTree ROOT,BiTNode *p,BiTNode *q)
{
       top=0,bt=ROOT;
       while(bt!=NULL||top>0)
       {
            while(bt!=NULL)
            {
                s[++top].t=bt;
                s[top].tag=0;
                bt=bt->lchild;
            }
            while(top!=0&&s[top].tag==1)
            {
                if(s.[top].t==p){
                    for(i=1;i<=top;i++)
                    s1[i]=s[i];
                    top1=top;
                }
                if(s[top].t==q)
                    for(i=top;i>0;i--)
```

```
                                for(j=top1;j>0;j--)
                                        if(s1[j].t==s[i].t)
                                                return s[i].t;
                }
                top--;
        }
        if(top!=0)
        {
                s[top].tag=1;
                bt=s[top].t->rchild
        }
        return NULL;
}
```

- 假设二叉树采用二叉链表存储结构，设计一个算法，求非空二叉树b的宽度（即具有结点数最多的那一层的结点个数)

思路

层次遍历

代码

```
typedef struct {
        BiTree data[MaxSize];
        int level[MaxSize];
        int front,rear;
}QU;
int BiTWidth(BiTree b){
        BiTree p;
        int k,max,i,n;
        QU.front=QU.rear=-1;
        QU.rear++;
        QU.data[QU.rear]=b;
        QU.level[QU.rear]=1;
        while(QU.front<QU.rear){
                QU.front++;
                p=QU.data[QU.front];
                k=QU.level[QU.front];
                if(p->child!=NULL)
                {
                        QU.rear++;
                        QU.data[QU.rear]=p->lchild;
                        QU.level[QU.rear]=k+1;
                }
```

```
            if(p->rchild!=NULL)
            {
                QU.rear++;
                QU.data[QU.rear]=p->rchild;
                QU.level[QU.rear]=k+1;
            }
        }
        max=0,i=0;
        k=1;
        while(i<QU.rear){
            n=0;
            while(i<=QU.rear&&QU.level[i]==k)
            {
                n++;
                i++;
            }
            k=QU.level[i];
            if(n>max)
                max=n;
        }
        return max;
}
```

- 设有一棵满二叉树（所有结点值均不同）， 已知其先序序列为p,设计一个算法求其后序序列post;

思路

递归

代码

```
void PreToPost(ElemType pre[ ],int l1,int h1,ElemType post[ ],int l2,int h2){
        int half;
        if(h1>=l1){
        post[h2]=pre[l1];
        half=(h1-l1)/2;
        PreToPost(pre,l1+1,l1+half,post,l2,l2+half-1);
        PreToHost(pre,l1+half+1,h1,post,l2+half,h2-1);
        }
}
```

- 设计一个算法将二叉树的叶结点按从左到右的顺序连成一个单链表，表头指针为head。二叉树按二叉链表方式存储，链接时用叶结点的右指针域来存放单链表指

针。

思路

中序遍历

代码

```
LinkedList head,pre=NULL;
LinkedList InOrder(BiTree bt)
{
	if(bt)
	{
		InOrder(bt->lchild);
		if(bt->lchild==NULL&&bt->rchild==NULL)
		{
			if(pre=NULL){
				head=bt;
				pre=bt;
			}
			else
			{
				pre->rchild=bt;
				pre=bt;
			}
		}
		InOrder(bt->rchild);
			pre->rchild=NULL;
	}
	return head;
}
```

- 试设计判断两棵二叉树是否相似的算法。所谓二叉树T和T2相似，指的是T和T2都是空的二叉树或都只有一个根结点；或者T的左子树和T2的左子树是相似的，且T的右子树和T2的右子树是相似的。

思路

递归

代码

```
int similar(BiTree T1,BiTree T2)
{
	int lefts,rights;
	if(T1==NULL&&T2==NULL)
		return 1;
	else
		if(T1==NULL||T2==NULL)
			return 0;
	else{
		lefts=similar(T1->lchild,T2->lchild);
		rights=similar(T1->rchild,T2->rchild);
		return lefts&&rights;
	}
}
```

【2014 统考真题】二叉树的带权路径长度（WPL）是二叉树中所有叶结点的带权路径长度之和。给定一棵二叉树 T，采用二叉链表存储，结点结构为

left	weight	right

其中叶结点的 weight 域保存该结点的非负权值。设 root 为指向 T 的根结点的指针，请设计求 T 的 WPL 的算法，要求：

1）给出算法的基本设计思想。

2）使用 C 或 C++语言，给出二叉树结点的数据类型定义。

3）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

思路

1）算法的基本设计思想。

① 本问题可采用递归算法实现。根据定义：

二叉树的 WPL 值 = 树中全部叶结点的带权路径长度之和

= 根结点左子树中全部叶结点的带权路径长度之和 + 根结点右子树中全部叶结点的带权路径长度之和

叶结点的带权路径长度 = 该结点的 weight 域的值×该结点的深度

设根结点的深度为 0，若某结点的深度为 d 时，则其孩子结点的深度为 $d+1$。

在递归遍历二叉树结点的过程中，若遍历到叶结点，则返回该结点的带权路径长度，否则返回其左右子树的带权路径长度之和。

② 若借用非叶结点的 weight 域保存其孩子结点中 weight 域值的和，则树的 WPL 等于树中所有非叶结点 weight 域值之和。

采用后序遍历策略，在遍历二叉树 T 时递归计算每个非叶结点的 weight 域的值，则树 T 的 WPL 等于根结点左子树的 WPL 加上右子树的 WPL，再加上根结点中 weight 域的值。

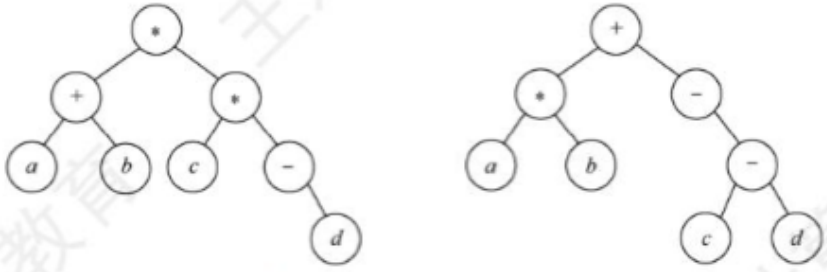
在递归遍历二叉树结点的过程中，若遍历到叶结点，则 return 0 并且退出递归，否则递归计算其左右子树的 WPL 和自身结点的权值。

代码

```
typedef struct node{
        int weight;
        int node *left,*right;
}BiTree;
```

```
int WPL(BiTree *root)
{
        return WPL1(root,0);
}
int WPL(BiTree *root,int d)
{
        if(root->left==NULL&&root->right==NULL)
                return (root.weight*d);
        else
                return WPL1(root->left,d+1)+WPL(root->right,d+1);
}
```

【2017统考真题】请设计一个算法，将给定的表达式树（二叉树）转换为等价的中缀表达式（通过括号反映操作符的计算次序）并输出。例如，当下列两棵表达式树作为算法的输入时：



输出的等价中缀表达式分别为(a+b)*(c*(-d))和(a*b)+(-(c-d))。

二叉树结点定义如下：

```
typedef struct node{
    char data[10];                  //存储操作数或操作符
    struct node *left, *right;
}BTree;
```

要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用C或C++语言描述算法，关键之处给出注释。

思路

中序遍历

代码

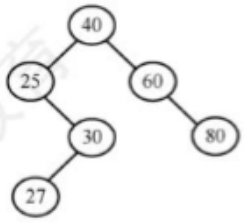
```
void BtreeToE(BTree *root){
        By=treeToExp(root,1);
}
```

```
void BtreeToExp(BTree *root,int deep){
        if(root==NULL) return;
        else if(root->left==null&&root->right==NULL)
                printf("%s",root->data);
        else
                if(deep>1) printf("(");
                BtreeToExp(root->left,deep+1);
                        printf("s",root->data);
                BtreeToExp(root->right,deep+1);
                if(deep>1)
                        printf(")");
}
```

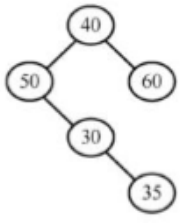
【2022 统考真题】已知非空二叉树 T 的结点值均为正整数，采用顺序存储方式保存，数据结构定义如下：

```
typedef struct {                    //MAX_SIZE 为已定义常量
    int SqBiTNode[MAX_SIZE];        //保存二叉树结点值的数组
    int ElemNum;                    //实际占用的数组元素个数
} SqBiTree;
```

T 中不存在的结点在数组 SqBiTNode 中用-1 表示。例如，对于下图所示的两棵非空二叉树 T_1 和 T_2，



二叉树 T_1



二叉树 T_2

T_1 的存储结果如下：

T1.SqBiTNode	40	25	60	-1	30	-1	80	-1	-1	27		

T1.ElemNum=10

T_2 的存储结果如下：

T2.SqBiTNode	40	50	60	-1	30	-1	-1	-1	-1	-1	35	

T2.ElemNum=11

请设计一个尽可能高效的算法，判定一棵采用这种方式存储的二叉树是否为二叉搜索树，若是，则返回 true，否则，返回 false。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

思路

对于采用顺序存储方式保存的二叉树，根结点保存在SqBiTNode【0】中：当某结点保存在SqBiTNode[i]中时，若有左孩子，则其值保存在SqBiTNode【2i+1]中：若有右孩子，则其值保存在SqBiTNode[2i+2]中：若有双亲结点，则其值保存SqBiTNode[(i-1)/2]中。

二叉搜索树需要满足的条件是：任意一个结点值大于其左子树中的全部结点值，小于其

右子树中的全部结点值。中序遍历二叉搜索树得到一个升序序列。使用整型变量va1记录中序遍历过程中已遍历结点的最大值，初值为一个负整数。若当前遍历的结点值小于或等于val,则算法返回false,否则，将val的值更新为当前结点的值。

代码

```
#define false 0;
#define true 1;
typedef int bool
bool judgeInOrderBST(SqBiTree bt,int k,int *val){
	if(k<bt.ElemNum&&bt.SqBiTNode[k]!=-1)
	{
		if(!judgeInOrderBST(bt,2*k+1,val)) return false;
		if(bt.sQBiTNode[k]<=*val) return false;
		*val=bt.SqBiTNode[k];
		if(!judgeInOrderBST(bt,2*k+2,val)) return false;
	}
	return true;
}
```