**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 11 月 28 日**

**实验七 二叉树**

**一、问题描述与分析**

1、理解二叉树的结构特征及各种存储方法；

2、掌握前序遍历算法；

3、掌握二叉树的前序及层次遍历算法，设计并实现树结构相关问题算法，如求高度、叶子结点数等。

（1）编写算法函数void preorder1(bintree t)实现二叉树t的非递归前序遍历。

（2）编写算法函数void levelbintree(bintree t)，实现二叉树的层次遍历。

（3）试编写一个函数，返回一棵给定二叉树在前序遍历下的最后一个结点。

（4）假设二叉树采用链式方式存储，root为其根结点，编写一个函数int Depth(bintree t, char x)，求值为x的结点在二叉树中的层次。

（5）试编写一个函数，将一棵给定二叉树中所有结点的左、右子女互换。

（6）试编写一个递归函数bintree buildbintree(char \*pre, char \*mid, int length)，

根据二叉树的前序序列pre、中序序列mid和前序序列长度length，构造二叉树的存储结构，函数返回二叉树的树根地址。

2.分析

(1) 前序遍历首先访问根结点，再按照前序遍历的方式遍历左子树，右子树，由于是采用非递归实现，所以需要利用栈将根结点暂时保存，待访问完左子树，再将栈顶结点弹出访问右子树；

(2) 中序遍历首先访问左子树，因此需要栈保存当前根结点，保存后，先访问左子树，直到左子树访问完，再将栈顶结点弹出访问根结点和右子树；

(3) 根据前序遍历的特点，只需要不断访问当前结点的右子树，同时兼顾判断当前结点的左右子树是否都为空，若不都为空，则右子树优先，若右子树也为空，则访问左子树，直到一个结点左右子树皆为空，此时这个结点就是前序遍历最后一个结点，同理可得后序遍历方式，子树优先，并且左子树优先级最高，优先访问左子树，访问到的第一个左右子树为空的结点即为后序遍历访问第一个结点；

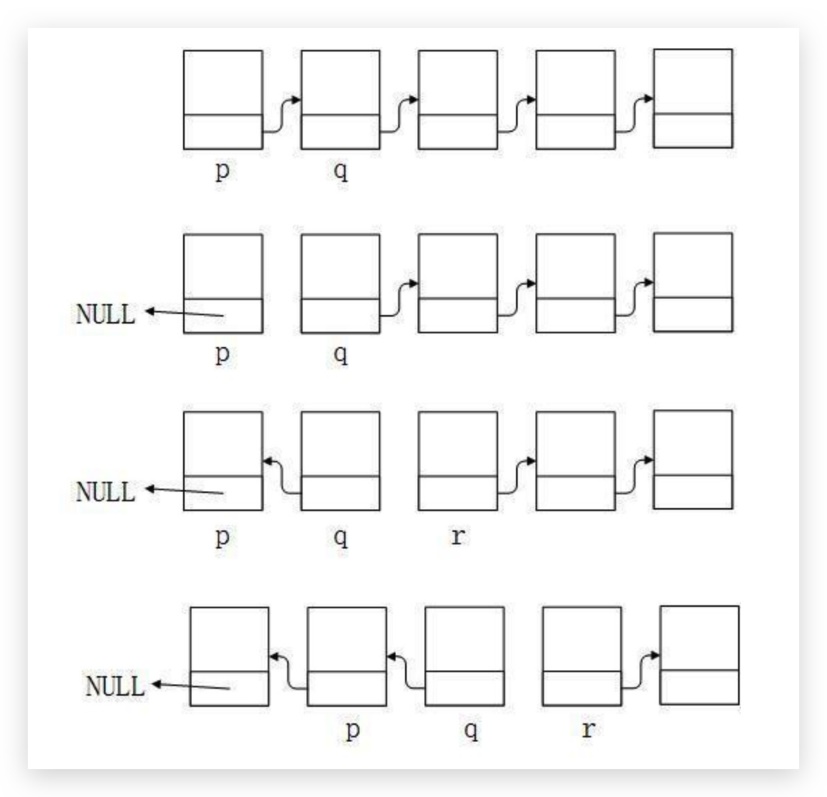
(4) 利用树栈的特点，依次将左子树入栈，然后将入栈的子树标志位置1，用于下次访问左子树，若找到x对应的结点，则返回栈顶值+1（即该结点的层次），接下来再将右子树依次入栈，执行同样的操作即可，若找不到，返回0（NULL）；

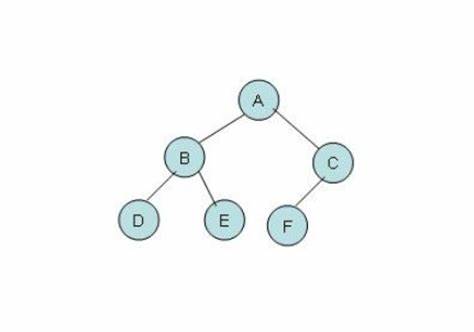
(5) 采用递归的方式实现非常简单，每个结点执行的操作就是交换左右子女，然后分别对左右子女进行交换即可；

(6) 利用递归实现，不断递归即可；

**二、数据结构与算法设计**

链表图示如下：



二叉树结构图示如下：  


**三、算法复杂度分析**

（1）单重循环，算法复杂度为O（n）；

（2）单重循环，算法复杂度为O（n）；

（3）多选择递归，算法复杂度为O（n）；

（4）多重单循环，算法复杂度为O（2n）；

（5）选择递归，算法复杂度为O（n）；

（6）选择递归，算法复杂度为O（n）；

**四、测试计划**

1.编写目的

加深对数据结构链表，二叉树的存储结构，递归程序的理解。

2.开发及运行环境Dev c++

Visual Stdio Code 1.40

3.小项测试截图

（1）前序遍历



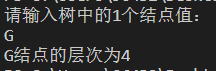
（2）中序遍历



（3）前续遍历最后一个结点和后序遍历第一个结点



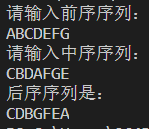
（4）求结点x所在的层次



（5）将给定二叉树的左右子女互换



（6）根据二叉树的前序序列和中序序列，以及树的结点数构造一个二叉树



**五、源程序(仅贴算法函数部分)**

**<lab7\_01>:**

void preorder1(bintree t)

{

    seqstack s;

    s.top = 0;

    while(t || (s.top != 0))

    {

        if(t)

        {

            printf("%c", t->data);

            push(&s, t);

            t = t->lchild;

        }

        else

        {

            t = pop(&s);

            t = t->rchild;

        }

    }

}

**<lab7\_02>:**

void levelbintree(bintree t)

{

    seqstack s;

    s.top = 0;

    while(t || (s.top != 0))

    {

        if(t)

        {

            push(&s, t);

            t =t->lchild;

        }

        else

        {

            t = pop(&s);

            printf("%c", t->data);

            t =t ->rchild;

        }

    }

}

**<lab7\_03>:**

bintree prelast(bintree t)

{

    if (t==NULL)  return NULL;

    else

        {

            if (t->lchild==NULL && t->rchild==NULL)  return t;

            else

                {

                    if (t->rchild )  return prelast(t->rchild);

                    else  return prelast(t->lchild);

                }

        }

}

bintree postfirst(bintree t)

{

    bintree p;

    if (t)

    {

        p=t;

        while (p && p->lchild || p->rchild)

        {

            if (p->lchild)

                p=p->lchild;

            else

               p=p->rchild;

        }

    }

    return p;

}

**<lab7\_04>:**

int Depth(bintree t,char x)

{

    seqstack  s;

    int i=0,j;

    s.top=0;

    while(t || s.top!=0)

    {

        while(t)

        {

            s.data[s.top]=t;

            s.tag[s.top]=0;

            s.top++;

            t=t->lchild;

        }

        while(s.top>0 && s.tag[s.top-1])

        {

            s.top--;

            t=s.data[s.top];

            if(t->data==x) return s.top+1;

        }

        if(s.top>0)

        {

            t=s.data[s.top-1];

            s.tag[s.top-1]=1;

            t=t->rchild;

        }

        else  t=NULL;

  }

}

**<lab7\_05>:**

void change(bintree t)

{

   bintree p;

   if (t)

   {

      p = t->lchild;

      t->lchild = t->rchild;

      t->rchild = p;

      change(t->lchild);

      change(t->rchild);

   }

}

**<lab7\_06>:**

bintree buildbintree(char \*pre, char \*mid,int length)

{

    bintree t;

    int len;

    char \*p;

    if (length<=0) return NULL;

    else

    {

        t=(bintree)malloc(sizeof(binnode));

        t->data=\*pre;

        t->lchild=t->rchild=NULL;

        p=mid;

        while ( p<mid+length && \*p!=\*pre)

            p++;

        len=p-mid;

        t->lchild=buildbintree(pre+1,mid,len);

        t->rchild=buildbintree(pre+len+1,p+1,length-1-len);

        return t;

    }

}