# 习题3 树结构

3-1设二叉树T中度为1的结点11个，度为2的结点12个，则二叉树T共有(   **C**   )个叶子结点。

(A) 11

(B) 12

(C) 13

(D) 36

3-2 设树T的度为4，其中度为1，2，3和4的结点个数分别为4，2，1，1，则T中的叶子数为( D )。

(A) 5

(B) 6

(C) 7

(D) 8

3-3 已知一棵度为k的树中，有n1个度为1的结点，n2个度为2的结点，…，nk个度为k的结点。试计算该树的叶子结点数。

**答：非叶子结点个数：n1 + n2 + ... + nk，树的边数：1\*n1 + 2\*n2 + ... + k\*nk，**

**树的总结点个数 = 边数 + 1 = 1\*n1 + 2\*n2 + ... + k\*nk + 1**

**因此，叶子结点个数 = 总结点个数 - 非叶子结点个数 = (1\*n1 + 2\*n2 + ... + k\*nk + 1) - (n1 + n2 + ... + nk)**

3-4 证明：如果二叉树T的叶子结点数为n0，度为2的结点数为n2，则n0=n2+1。

**证明：设度为1的结点个数为n1，则二叉树总节点个数n = n0 + n1 + n2.**

**除了根结点，每一个结点都有一条边与之对应，设二叉树边数为m，那么m = n-1.**

**由于度为i的结点有i个分支，则m = 0 + 1\*n1 + 2\*n2.**

**联立三个方程，得n0 = n2 + 1.**

3-5 对于任意非空二叉树，要设计出其后序遍历的非递归算法而不使用栈结构，最适合的方法是对该二叉树采用( **B** )存储结构。

(A) 二叉链表

(B) 三叉链表

(C) 索引

(D) 顺序

3-6 一棵二叉树的叶子结点在其先序、中序和后序序列中的相对位置( **C** )。

(A) 肯定发生变化

(B) 可能发生变化

(C) 不会发生变化

(D) 无法确定

3-7 设二叉树T按照二叉链表存储，则下列递归算法的主要功能是(   **B**   )。

int F(BiTree T)

{

if (!T) return 0;

x=F(T->Lchild);

y=F(T->Rchild);

if (y>x) x=y;

return x+1;

}

(A) 交换二叉树T的左右子树

(B) 计算二叉树T的高度

(C) 计算二叉树T的叶子结点数

(D) 先遍历左子树，再遍历右子树

3-8 已知二叉树T的先序序列为ABCDEF，中序序列为CBAEDF, 则T的后序序列为( **A** )。

(A) CBEFDA

(B) FEDCBA

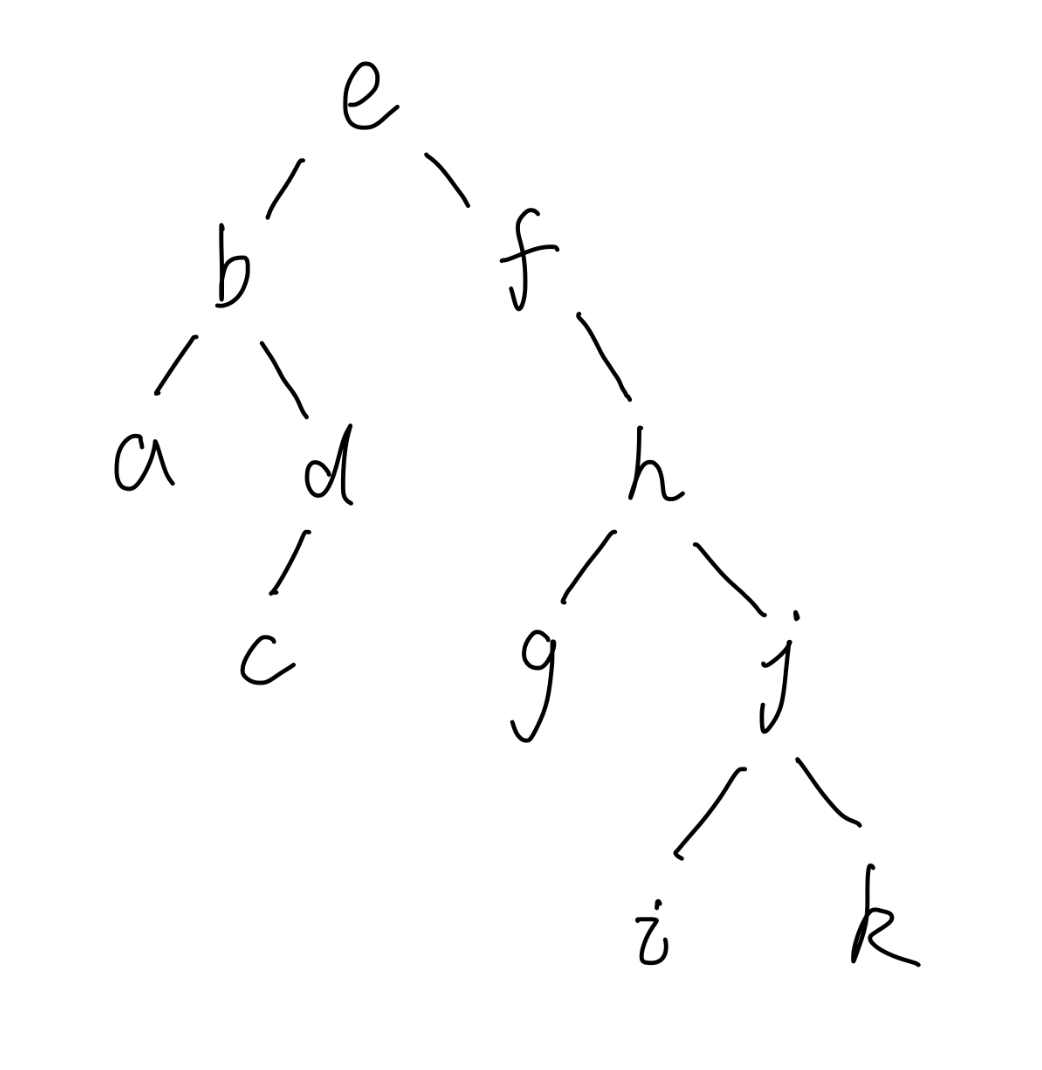
(C) CBEDFA

(D) 不确定

3-9 简述由先序序列和中序序列构造二叉树的基本操作方法。

**答：先从先序序列中取出第一个元素作为根结点的值，接着在中序序列中找到根结点的位置，将根结点左右两部分分成左子树和右子树。递归调用构造函数来构建左子树，其中先序序列为左子树部分，中序序列也是左子树部分，再递归调用构造函数来构建右子树，其中先序序列为右子树部分，中序序列也为右子树部分，最后将左子树和右子树连接到根节点上构成完整的二叉树。**

3-10 已知二叉树的先序序列为ebadcfhgjik，中序序列为abcdefghijk，试画出该二叉树。

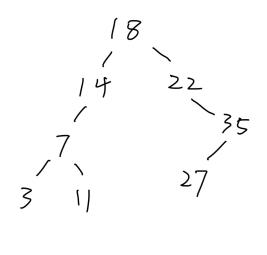


3-11 已知二叉树T的中序序列和后序序列分别为

(中序) 3, 7, 11, 14, 18, 22, 27, 35

(后序) 3, 11, 7, 14, 27, 35, 22, 18

试画出二叉树T。



3-12 已知二叉树T按照二叉链表存储，设计算法，计算T中叶子结点的数目。

//计算二叉树的叶子个数

int leaf (Tree& T) {

    if (!T)

        return 0;

    else if (!T->Lchild && !T->Rchild)  //若该节点是整个树的叶子

        return 1;

    else //否则递归求叶子个数

        return leaf(T->Lchild) + leaf(T->Rchild);

}

3-13 已知二叉树T按照二叉链表存储，设计算法，交换T的左子树和右子树。

//交换二叉树T的左右子树

void ExChange\_Tree(Tree& T) {

    Tree \*tmp = new TreeNode; //定义中间变量用于交换左右子树

    if (T)

    {

        tmp = T->rchild;

        T->rchild = T->lchild;

        T->lchild = temp;

        ExChange\_Tree(T->lchild);

        ExChange\_Tree(T->rchild);

    }

    else

        return;

}

3-14 先序后继线索化算法是根据二叉链表建立先序后继线索二叉链表，其基本原则是在前驱空指针域中写入后继线索，即将右子树的( **B** )指针写入左子树的最后一个叶子结点右指针域。

(A) 线索

(B) 根结点

(C) 前驱结点

(D) 后继结点

3-15 设计算法，在先序线索二叉树中，查找给定结点p在先序序列中的后继。

typedef int TElemType;

typedef enum PointerTag{Link,Thread};//Link==0，指针；Thread==1，线索

typedef struct BiThrNode

{

    TElemType data;

    struct BiThrNode\* lchild, \* rchild;

    PointerTag LTag, RTag;//左右标志

}BiThrNode, \*BiThrTree;

//由先序遍历创建先序线索化链表

int PreThreading(BiThrTree& T, BiThrTree& pre)

{

    if (T)

    {

        if (!T->lchild)

        {

            T->LTag = Thread;

            T->lchild = pre;

        }

        if (pre && !pre->rchild)

        {

            pre->RTag = Thread;

            pre->rchild = T;

        }

        pre = T;

        if (T->LTag == Link)

            PreThreading(T->lchild, pre);

        if (T->RTag == Link)

            PreThreading(T->rchild, pre);

    }

    return 1;

}

int PreOderThreading(BiThrTree& TT, BiThrTree T)

{

    //先序遍历二叉树T，并将其先序线索化，TT指向头节点

    BiThrTree pre;

    if (!(TT = (BiThrTree)malloc(sizeof(BiThrNode))))

        return 0;

    TT->LTag = Thread;

    TT->RTag = Link;

    TT->lchild = TT;//左子树回指

    if (!T)

        TT->rchild = TT;//若二叉树为空，右子树回指

    else

    {

        TT->rchild = T;

        pre = TT;

        PreThreading(T, pre);//先序遍历进行先序线索化

        pre->rchild = T;//最后一个结点线索化

        pre->RTag = Thread;

        TT->lchild = pre;

    }

    return 0;

}

//从二叉线索树上任一结点q开始查找结点\*p。如果找到，将\*p的后继结点指针存于q中，返回1; 否则返回0

int Find(BiThrTree& q, TElemType\* p)

{

    BiThrTree pt = q;

    if (!pt)

        return 0;

    if (pt->data == \*p)

    {

        if (pt->LTag == Link)

            q = pt->lchild;

        else

            q = pt->rchild;

        return 1;

    }

    pt = q->rchild;

    while (pt != q && pt->data != \*p)

    {

        if (pt->LTag == Link)

            pt = pt->lchild;

        else

            pt = pt->rchild;

    }

    if (pt == q)

        return 0;

    if (pt->data == \*p)

    {

        if (pt->LTag == Link)

            q = pt->lchild;

        else

            q = pt->rchild;

    }

    return 0;

}

3-16对n (n≥2)个权值均不相同的字符构造哈夫曼树T，不正确的叙述( **A** )。

(A) T一定是一棵完全二叉树

(B) T中一定没有度为1的结点

(C) T中两个权值最小的结点一定是兄弟结点

(D) T中任一分支结点的权值一定不小于下一层任一结点的权值

3-17设计一个求结点x在二叉树中的双亲结点算法。

//求结点x的双亲结点

void Parent(Tree& T,char x){

    if(T){

        if((T->lchild) && T->lchild->data == x) {    //左孩子不为空 ，判断左孩子与x是否相等

            cout << "存在" << x << "的双亲结点为:" << T->data;

            return;

        }

        if((T->rchild) && T->rchild->data == x) {   //右孩子不为空 ，判断右孩子与x是否相等

            cout << "存在" << x << "的双亲结点为:" << T->data;

            return;

        }

        else{

            Parent(T->lchild,x);

            Parent(T->rchild,x);

        }

    }

}