**《数据结构与算法》作业**

# 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。

设总节点为n，n=n0+n1+n2+…+nk，n=0\*n0+1\*n1+…k\*nk，两式相减，n0＝。

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。

假设该二叉树总共有n个结点(n=n0+n1+n2)，则该二叉树总共会有n-1条边，度为2的结点会延伸出两条边，同理，度为1的结点会延伸出一条边，则可列公式：n-1 = 2\*n2 + 1\*n1 ，合并可得：2\*n2 + 1\*n1 +1 =n0 + n1 + n2 ，则 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。

1.先在先序遍历中找到根结点A，再通过A在中序遍历的位置找出左右子树。

2.在A的左子树中，通过先序找到左子树的根结点，转1。

3.在A的左右子树中，通过先序找到右子树的根结点，转1。

通过递归算法实现，

struct TreeNode\* buildTree(int\* preorder, int\* inorder, int len){

if(len == 0)

return NULL;

// 每次选中先序遍历数组中的第一个数据作为根节点

int val = preorder[0];

int index = 0;

// 找到根节点在中序遍历数组中的位置

while(inorder[index] != val)

index++;

struct TreeNode\* node = malloc(sizeof(struct TreeNode));

node->val = val;

// 递归左树

node->left = buildTree(preorder + 1, inorder, index);

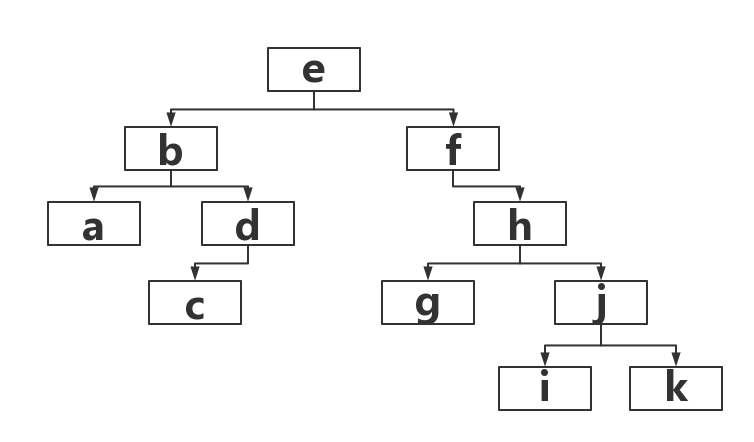
// 递归右树

node->right = buildTree(preorder + index + 1, inorder + index + 1, len - index - 1);

return node;

}

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。

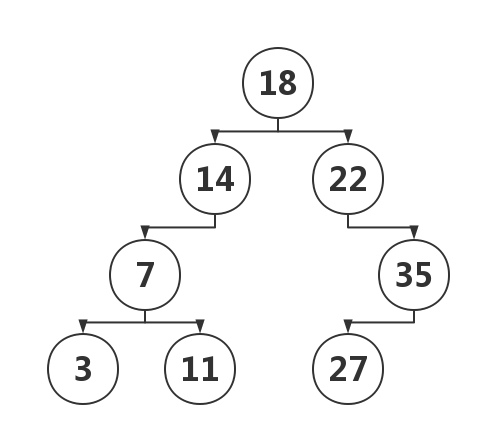


3-11 已知二叉树T的中序序列和后序序列分别为

(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。

int LeafNum(BTNode \*t)

{

int n1,n2;

if(t==NULL)

return 0;

else if(t->lchild==NULL&&t->rchild==NULL)

return 1;

else

{

n1=LeafNum(t->lchild);

n2=LeafNum(t->rchild);

return n1+n2;

}

}

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。

void SwapTree(BiTree \*T)

{

BitNode \*temp;

if (\*T)

{

temp = (\*T)->lchild;

(\*T)->lchild = (\*T)->rchild;

(\*T)->rchild = temp;

SwapTree(&((\*T)->lchild));

SwapTree(&((\*T)->rchild));

}

}

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。

BiTree PreNext(BiTree p)

{

BiTree next;

if(p->Rtag == 1)

next = p->RChild;

else if(p->LChild)

{

next = p->LChild;

}else{

next = p->RChild;

}

return next;

}

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。

BiTree FindParents(BiTree T,char x)

{

If(T != NULL)

{

if(T->lchild != NULL && T->lchild->data == x) return T;

if(T->rchild != NULL && T->rchild->data == x) return T; FindParents(T->lchild,x);

FindParents(T->rchild,x);

}

return NULL;

}