**《数据结构与算法》作业**

**22920212204392 黄勖**

# 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

**注：**

**一棵含有n个结点的树，有n-1个分支，即 n = 1\*4 + 2\*2 + 3\*1 + 4\*1 + 1 = 16;**

**又由于 n = n0 + n1 + n2 + n3 + n4 = n0 + 8;**

**n0 + 8 = 16，所有叶子结点个数为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：设该树中的叶子数为n0个。该树中的总结点数为n个，则有：**

**n=n0+n1+n2+…+nK ①**

**n-1=0\*n0+1\*n1+2\*n2+…+K\*nK ②**

**联立①②方程组可得：**

**叶子数为：**

**n0=1+0\*n1+1\*n2+2\*n3+...+(K-1)\*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：**

**假设二叉树的度为0，1，2的结点为n0,n1,n2,总节点数为n**

**则有按照结点求和的**

**n = n0 + n1 + n2 ①**

**按照边求和得：**

**n = n1 + 2 \* n2 + 1 ②**

**所以 ② - ①可得**

**n2 + 1 - n0 = 0**

**即**

**n0 = n2 + 1 ③**

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

**注：三叉链表比二叉链表多一个指向父结点的指针**

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

**注：画图或按照3-9操作。**

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：**

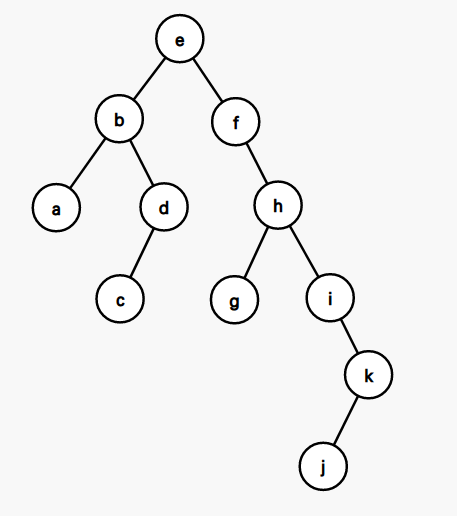
**①取先序遍历序列的第一个值，用该值构造根结点，然后在中序遍历序列中查找与该元素相等的值，这样就可以把序列分为三部分：左子树（如果有）、根结点和右子树（如果有）。**

**②将两个序列都分成三部分，这样就分别形成了根结点的左子树和右子树的先序遍历和后序遍历的序列。**

**③重复①和②步骤，直至所有结点都处理完就可以完整构成一颗二叉树了。**

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。

**答：**



**注：画于**[**https://csacademy.com/app/graph\_editor/**](https://csacademy.com/app/graph_editor/)

**输入数据：**

**11**

**e b**

**e f**

**b a**

**b d**

**d c**

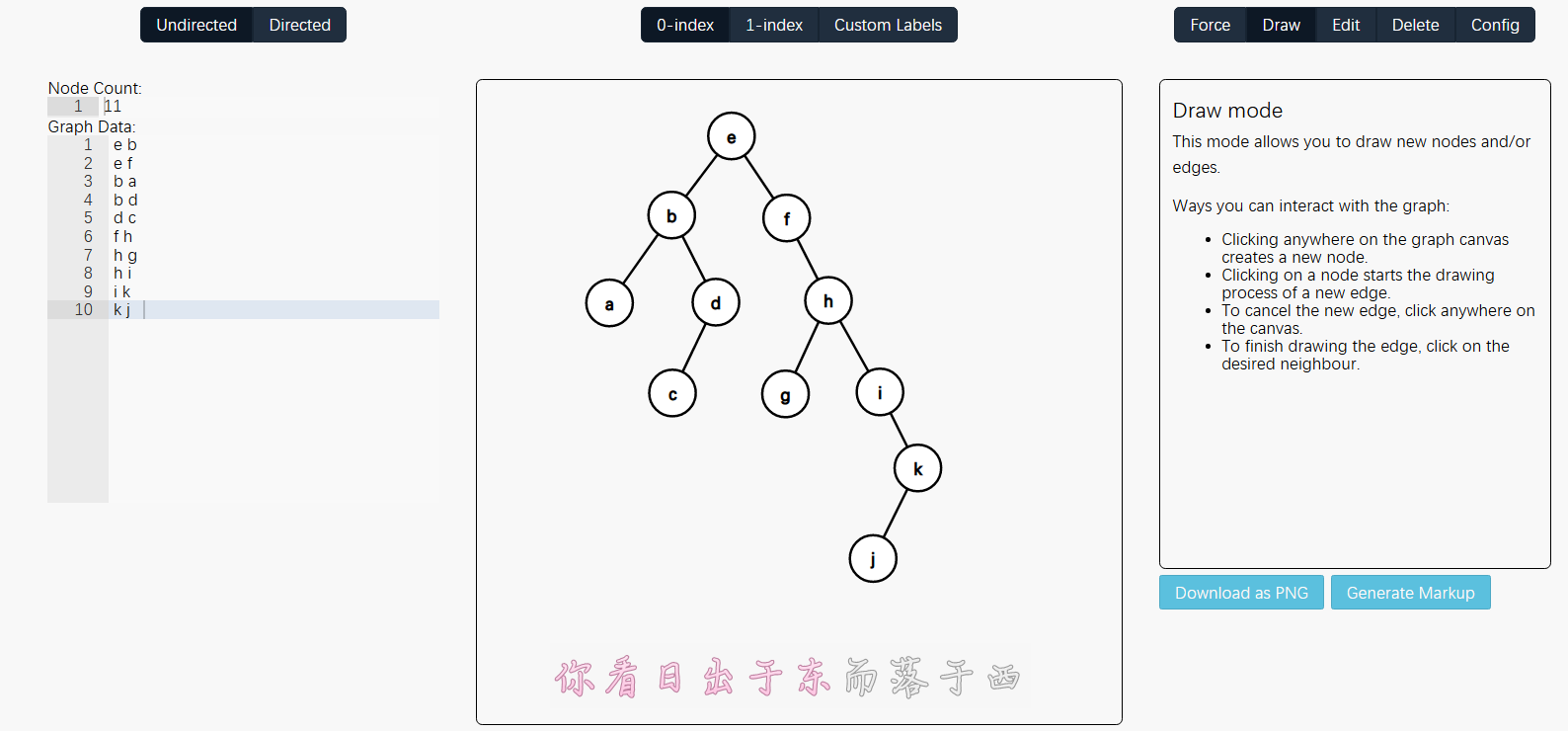
**f h**

**h g**

**h i**

**i k**

**k j**



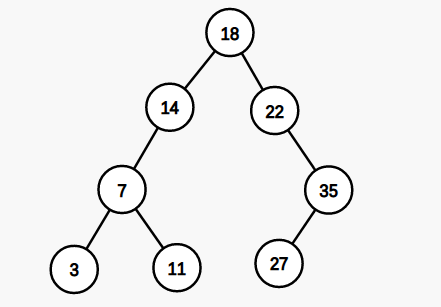
3-11 已知二叉树T的中序序列和后序序列分别为

(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。

**答：**



**输入数据：**

**8**

**18 14**

**18 22**

**14 7**

**7 3**

**7 11**

**22 35**

**35 27**

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。

**答：**

**int Find(BiTree T) {**

**if (!T) return 0;**

**if (!T->Lchild && !T->Rchild) return 1;**

**return F(T->Lchild)+F(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。

**答：**

**typedef struct BiTNode {**

**int data;**

**struct BiTNode \*lchild, \*rchlid;**

**}BiTNode,\*BiTree;**

**bool ChangLR(BiTree T)**

**{**

**if (T->lchild == NULL && T->rchlid == NULL)**

**{**

**return false;**

**}**

**else //交换当前左右子树**

**{**

**BiTree temp;**

**temp = T->lchild;**

**T->lchild = T->rchlid;**

**T->rchlid = temp;**

**}**

**ChangLR(T->lchild);**

**ChangLR(T->rchlid);**

**}**

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( **C** )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。

**答：**

**线索二叉树：根据某次遍历, 在二叉树中的相关空指针域都写入线索(后继线索或**

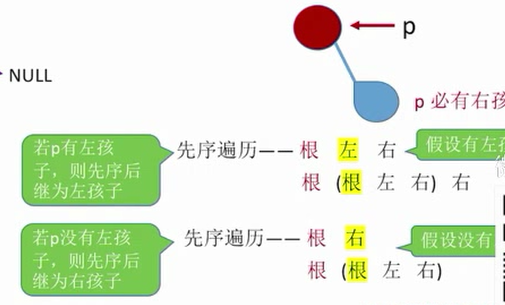
**前驱线索)，即成为线索二叉树。**

**先序后继：先序遍历中得到的后继。**

**在先序线索二叉树中找到指定结点\*p的先序后继next**

**若p->rtag==1（被线索化了），则 next=p->rchild（直接就是前驱）**

**若p->rtag==0（没有被线索化）， 必定是有右孩子的**

****

**/\*先序线索二叉树找后继\*/**

**ThreadNode \*find(ThreadNode \*p){**

**if(p->RTag == 1){**

**p = p->rchild;**

**}else{**

**if(p->lchild){**

**p = p->lchild;**

**}else{**

**p = p->rchild;**

**}**

**}**

**if(p->data)**

**return p;**

**else**

**return NULL; //最后一个节点无后继**

**}**

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。

**答：**

**typedef struct node {**

**datatype data;**

**struct node \*lchild,\*rchild;**

**} bitree;**

**bitree \*q[20];**

**int r=0,f=0,flag=0;**

**void preorder(bitree \*bt, char x) {** **//记录先序遍历**

**if (bt!=0 && flag==0)**

**if (bt->data==x) {**

**flag=1;**

**return;**

**} else {**

**r=(r+1)% 20;**

**q[r]=bt;**

**preorder(bt->lchild,x);**

**preorder(bt->rchild,x);**

**}**

**}**

**void parent(bitree \*bt,char x) { //求双亲结点**

**int i;**

**preorder(bt,x);**

**for(i=f+1; i<=r; i++) if (q[i]->lchild->data==x || q[i]->rchild->data) break;**

**if (flag==0) printf("not found x\n");**

**else if (i<=r) printf("%c",bt->data);**

**else printf("not parent");**

**}**