第六章 树和二叉树

一、选择题

1.在线索化二叉树中,t 所指结点没有左子树的充要条件是（ ）

（A）t-〉left==NULL （B）t-〉ltag==1

（C）t-〉ltag=1 且 t-〉left=NULL（D）以上都不对

2.二叉树按某种顺序线索化后,任一结点均有指向其前趋和后继的线索,这种说法

（A）正确 （B）错误 （C）不同情况下答案不确定

3.二叉树的前序遍历序列中,任意一个结点均处在其子女结点的前面,这种说法（ ）

(A)正确 (B)错误 （C）不同情况下答案不确定

4.由于二叉树中每个结点的度最大为 2,所以二叉树是一种特殊的树,这种说法（ ）

（A）正确 （B）错误 （C）不同情况下答案不确定

5.设高度为 h 的二叉树上只有度为 0 和度为 2 的结点,则此类二叉树中所包含的结点数至少为（ ）。

（A）2h （B）2h-1（C）2h+1（D）h+1

6.已知某二叉树的后序遍历序列是 dabec。中序遍历序列是 debac,它的前序遍历序列是（ ）。

（A）acbed （B）decab（C）deabc （D）cedba

7.如果 T2 是由有序树 T 转换而来的二叉树,那么 T 中结点的前序就是 T2 中结点的（ ）

（A）前序（B）中序（C）后序（D）层次序

8.某二叉树的前序遍历结点访问顺序是 abdgcefh,中序遍历的结点访问顺序是 dgbaechf,则其后序遍历的结点访问顺序是（ ）。

（A）bdgcefha （B）gdbecfha （C）bdgaechf （D）gdbehfca

9.二叉树为二叉排序树的充分必要条件是其任一结点的值均大于其左孩子的值、小于其右孩子的值。这种说法（ ）

（A）正确（B）错误（C）不同情况下答案不确定

10.按照二叉树的定义,具有 3 个结点的二叉树有（ ）种。

（A）3（B）4（C）5（D）6

11.在一非空二叉树的中序遍历序列中,根结点的右边（ ）

（A）只有右子树上的所有结点（B）只有右子树上的部分结点

（C）只有左子树上的部分结点（D）只有左子树上的所有结点

12.树最适合用来表示（ ）。

（A）有序数据元素（B）无序数据元素

（C）元素之间具有分支层次关系的数据（D）元素之间无联系的数据

13.任何一棵二叉树的叶结点在先序、中序和后序遍历序列中的相对次序（ ）

（A）不发生改变（B）发生改变（C）不能确定 D.以上都不对

14.实现任意二叉树的后序遍历的非递归算法而不使用栈结构,最佳方案是二叉树采用（ ）存储结构。

（A）二叉链表（B）广义表存储结构（C）三叉链表（D）顺序存储结构

15.对一个满二叉树,m 个树叶,n 个结点,深度为 h,则（ ）

（A）n=h+m （B）h+m=2n（C）m=h-1（D）n=2h-1

16.如果某二叉树的前序为 stuwv,中序为 uwtvs,那么该二叉树的后序为（ ）

（A）uwvts （B）vwuts（C）wuvts （D）wutsv

17.具有五层结点的二叉平衡树至少有（ ）个结点。

（A）10（B）12（C）15（D）17

二、判断题

1.二叉树中任何一个结点的度都是 2。（ ）

2.由二叉树结点的先根序列和后根序列可以唯一地确定一棵二叉树。（ ）

3.一棵哈夫曼树中不存在度为 1 的结点。（ ）

4.平衡二叉排序树上任何一个结点的左、右子树的高度之差的绝对值不大于 2（ ）

三、填空题

1. 指出树和二叉树的三个主要差别_____,_____,_____。

2.从概念上讲,树与二叉树是两种不同的数据结构,将树转化为二叉树的基本目的是_____

3.若结点 A 有三个兄弟(包括 A 本身),并且 B 是 A 的双亲结点,B 的度是_____

4.若一棵具有 n 个结点的二叉树采用标准链接存储结构,那么该二叉树所有结点共有_____个空指针域。

5.已知二叉树的前序序列为 ABDEGCFHIJ,中序序列为 DBGEAHFIJC,写出

后序序列_____。

6.已知二叉树的后序序列为 FGDBHECA,中序序列为 BFDGAEHC ,并写出前序序列_____。

7.找出满足下列条件的二叉树

1) 先序和中序遍历，得到的结点访问顺序一样。_____

2) 后序和中序遍历，得到的结点访问顺序一样。_____

3)先序和后序遍历,得到的结点访问顺序一样。_____

8.一棵含有 n 个结点的 k 叉树，可能达到的最大深度和最小深度各是多少？_____

9.一棵二叉树有 67 个结点，这些结点的度要么是 0，要么是 2。这棵二叉树中度为 2 的结点有_____个。

10.含有 100 个结点的树有_____条边。

四、问答题

1.一棵深度为 h 的满 m 叉树具有如下性质:第 h 层上的结点都是叶结点,其余各层上每个结点都有 m 棵非空子树。若按层次从上到下,每层从左到右的顺序从 1 开始对全部结点编号,试计算:

(1)第 k 层结点数(1≤k≤h)。

(2)整棵树结点数。

(3)编号为 i 的结点的双亲结点的编号。

(4)编号为 i 的结点的第 j 个孩子结点(若有)的编号。

2.证明：一个满 k 叉树上的叶子结点数 n0 和非叶子结点数 n1 之间满足以下关系： n0=（k-1）n1+1

3.已知一组元素为(50,28,78,65,23,36,13,42,71），请完成以下操作:

(1)画出按元素排列顺序逐点插入所生成的二叉排序树 BT。

(2)分别计算在 BT 中查找各元素所要进行的元素间的比较次数及平均比较次数。

(3)画出在 BT 中删除(23〉后的二叉树。

4.有七个带权结点,其权值分别为 3,7,8,2,6,10,14,试以它们为叶结点构造一棵哈夫曼树(请按照每个结点的左子树根结点的权小于等于右子树根结点的权的次序构造〉,并计算出带权路径长度 WPL 及该树的结点总数。

5.有一电文共使用五种字符 a,b,c,d,e,其出现频率依次为 4,7,5,2,9。

(1)试画出对应的编码哈夫曼树(要求左子树根结点的权小于等于右子树根结点的权)。

(2)求出每个字符的晗夫曼编码。

(3)求出传送电文的总长度。

(4)并译出编码系列 1100011100010101 的相应电文。

五、算法设计

已知一棵具有 n 个结点的完全二叉树被顺序存储在一维数组 A[n]中,试编写一个算法输出 A[i]结点的双亲和所有孩子。

参考答案：

一、选择题

1. B 2.B 3. A 4. B 5. B 6.D 7、A 8、D 9、B 10、C 11、A 12、C 13、A 14、C 15、D 16、C 17 C

二、判断题

1× 2× 3√ 4√

三、填空题

1、①树的结点个数至少为 1，而二叉树的结点个数可以为 0；

②树种结点的最大读书没有限制，而二叉树结点的最大度数不能超过 2；

③树的结点无左右之分，而二叉树的结点有左右之分。

2、树可采用二叉树的存储结构并利用二叉树的已有算法解决树的有关问题。

3、4 4、n+1 5、DGEBHJIFCA 6、ABDEGCEH

7、①无左子树②无右子树③仅一个结点的二叉树 8、最大 n,最小 ⌊log2n⌋ +1 9、22 10、99

四、问答题

1.

答：（1）mk-1 （2）（mh-1）/（m-1）

（3）i=1 时，该结点为根，无双亲结点；否则其双亲结点的编号为（i+m-2）/m

（4）编号为 i 的结点的第 j 个孩子结点(若有)的编号为 i*m+（j-（m-1））

2.

证明：设树结点为 n，则 n=n0+n1

因是满 k 叉树，每个非叶子结点引 出 k 个分支，故有 k*n1 个分支。

除根外，每个分支引出一个结点，则树共有 k*n1 +1 个结点。

所以 n0+n1=k*n1+1

n0=(k-1)*n1+1

五、算法设计

```
void parent(int a[],int n,int i)
{
if(i==1)
{
printf("无双亲 \n");
return;
}
Else
printf("双亲:%d\n",a[(i-1)/2]);
}
void child(int a[],int n,int i) /*i 为序号 */
{
int queue[MAX],front=0,tail=0,p; /* 队列作为辅助，存储孩子的序号*/
queue[0]=i;tail++;
```

```
while(front<tail)
{
p=queue[front++];
if(p!=i)
printf("%d ",a[p-1]); /*自身不输出 */
if(2*p<=n)
queue[tail++]=2*p;
if(2*p+1<=n) queue[tail++]=2*p+1;
}
main()
{
int a[MAX],n,i;
printf(" 请输入二叉树的结点个数:");
scanf("%d",&n);
input(a,n);
printf("请输入 i:");
scanf("%d",&i);
parent(a,n,i);
child(a,n,i);
}
```

二叉树其他运算的算法（只作参考）

```
#include "stdio.h"
```

```
#include "malloc.h"
typedef struct node
{
char data;
struct node *lchild,*rchild;
}NODE;
NODE *T;
void create(NODE **T) //创建二叉树
{ char ch;
ch=getchar();
if(ch==' ') *T=NULL;
else
{ *T=(NODE *)malloc(sizeof(NODE));
(*T)->data=ch;
create(&((*T)->lchild));
create(&((*T)->rchild));? } }
void inorder(NODE *p) //中序编历二叉树
{ if(p!=NULL)
{ inorder(p->lchild);??
printf("%c ",p->data);?
inorder(p->rchild);??? }? }
int num=0;
```

```
void count(NODE *p) //统计出二叉树中单孩子的结点数方法 1
{
if(p!=NULL)
{
count(p->lchild);
if(p->lchild!=NULL&&p->rchild==NULL||p->lchild==NULL&&p->rchi
ld!=NULL)
num++;
count(p->rchild);
}
}
void count1(NODE *p,int *num1)
{
if(p!=NULL)
{
count1(p->lchild,num1);
if(p->lchild!=NULL&&p->rchild==NULL||p->lchild==NULL&&p->rchi
ld!=NULL)
(*num1)++;
count1(p->rchild,num1);
}
}
```

```
int onechild(NODE *t) //统计出二叉树中单孩子的结点数方法 2
{
int num1,num2;
if(t==NULL) return(0);
else
if(t->lchild==NULL&&t->rchild!=NULL||t->lchild!=NULL&&t->rchild
==NULL)
return(onechild(t->lchild)+onechild(t->rchild)+1);
else
{
num1=onechild(t->lchild);
num2=onechild(t->rchild);
return(num1+num2);
}
}
int sum(NODE *t) //统计出二叉树中所有结点数
{if(t==NULL) return(0);
else
return(sum(t->lchild)+sum(t->rchild)+1);
}
int leaf(NODE *t) //统计出二叉树中叶子结点数
{
```

```
if(t==NULL) return(0);
else
if(t->lchild==NULL&&t->rchild==NULL)
return(leaf(t->lchild)+leaf(t->rchild)+1);
else
return(leaf(t->lchild)+leaf(t->rchild));
}
void preorder1(NODE *root) //非递归二叉树前序编历
{ NODE *p,*s[100],*q; //q 为 p 的双亲结点
int top=0; //top 为栈顶指针
p=root;q=p;
while((p!=NULL)||(top>0))
{ while(p!=NULL)
{printf("%d ",p->data);
s[++top]=p; p=p->lchild; }
p=s[top--]; p=p->rchild; }}
void delk(NODE **root,char k) //删去并释放数据值为 k 的叶结点
{ NODE *p,*s[100],*q; //q 为 p 的双亲结点
int top=0; //top 为栈顶指针
if((*root)==NULL)return;
if((*root)->lchild==NULL
&&(*root)->rchild==NULL&&(*root)->data==k) {*root=NULL;return;}
```

```
p=*root;q=p;
while((p!=NULL)||(top>0))
{
while(p!=NULL)
{
if(p->lchild==NULL&&p->rchild==NULL &&p->data==k)
{if(q->lchild==p) q->lchild=NULL;
else q->rchild=NULL;
free(p);
return;
}
s[++top]=p; q=p; p=p->lchild; }
p=s[top--]; q=p;p=p->rchild; }}
void lev_traverse(NODE *T) //按层次从上到下,每层从右到左的顺序列出二
叉树所有结点的数据信息
{NODE *q[100],*p;
int head,tail;
q[0]=T;head=0;tail=1;
while(head<tail)
{p=q[head++];
printf("%c",p->data);
if(p->rchild!=NULL)
```

```
q[tail++]=p->rchild;
if(p->lchild!=NULL)
q[tail++]=p->lchild;
}}
int depth(NODE *t) //求二叉树的深度
{
int num1,num2;
if(t==NULL) return(0);
if(t->lchild ==NULL&&t->rchild ==NULL) return(1);
else
{
num1=depth(t->lchild );
num2=depth(t->rchild );
if(num1>num2)
return(num1+1);
else
return(num2+1);
}
}
int onechild3(NODE *root) //非递归统计出二叉树共有多少个度为 1 的结
点
{ NODE *p,*s[100];
```

```
int top=0,num=0; //top 为栈顶指针
p=root;
while((p!=NULL)||(top>0))
{ while(p!=NULL)
{if(p->lchild!=NULL&&p->rchild==NULL
||p->lchild==NULL&&p->rchild!=NULL)
num++;
s[++top]=p; p=p->lchild; }
p=s[top--]; p=p->rchild; }
return num;
}
int like(NODE *t1,NODE *t2) //判定两颗二叉树是否相似
{
int like1,like2;
if(t1==t2&&t2==NULL) return(1);
else
if(t1==NULL &&t2!=NULL||t1!=NULL&&t2==NULL) return(0);
else
{
like1=like(t1->lchild,t2->lchild);
like2=like(t1->rchild ,t2->rchild );
return(like1&&like2);
```

```
}
}
char a[100]; //数组顺序存储二叉树
void change(NODE *t,int i) //将二叉树的链接存储转换为顺序存储
{
if(t!=NULL)
{a[i]=t->data;
change(t->lchild,2*i);
change(t->rchild,2*i+1);
}
}
int complete(NODE *t) //判断二叉树是否为完全二叉树
{
int i,flag=0;
change(t,1);
for(i=1;i<100;i++)
{if(!flag&&a[i]=='\0') flag=1;
if(flag&&a[i]!='\0') break;
}
if(i==100) return(1);
else return(0);
}
```