一、选择题

1.树形结构最适合用来描述\_\_\_\_\_\_\_

A．有序的数据元素 B. 无序的数据元素

C.数据元素之间具有层次关系的数据 D. 数据元素之间没有关系的数据

2. 对于一棵具有n个结点，度为4的树而言，\_\_\_\_\_\_\_\_

A．树的深度最多是n-4 B．树的深度最多是n-3

C．第i层上最多有 D．最少在某一层上正好有4个结点

3．“二叉树为空”意味着二叉树\_\_\_\_\_\_\_

A．由一些未赋值的空结点组成 B．根结点无子树 C．不存在 D．没有结点

4．按照二叉树定义，具有3个结点的二叉树有\_\_\_\_\_\_种形态（不考虑数据信息的组合情况）

A．2 B．3 C．4 D．5

5．若一棵度为7的树有8个度为1的结点，有7个度为2的结点，有6个度为3的结点，有5个度为4的结点，有4个度为5的结点，有3个度为6的结点，有2个度为7的结点，则该树一共有\_\_\_\_\_\_个结点。

A．35 B．28 C．77 D．78

6．若一棵二叉树有10个度为2的结点，则该二叉树的叶结点的个数为\_\_\_\_\_\_

A．9 B．11 C．12 D．不确定

7．若一棵满二叉树有2047个结点，则该二叉树中叶节点的个数为\_\_\_\_\_\_

A. 512 B. 1024 C. 2048 D. 4096

8．深度为h的满二叉树的第i层有\_\_\_\_\_\_个结点。

A． B．2i-1 C．2h-1 D．2h-1

9．深度为h的非空完全二叉树的第i层有\_\_\_\_\_\_个结点。

A． B．2i-1 C．2h-1 D．2h-1

10．具有n个结点的非空完全二叉树的深度为\_\_\_\_\_

A．n-1 B．n C． D．+1

11. 具有2000个结点的非空二叉树的最小深度为\_\_\_\_\_\_

A. 9 B. 10 C. 11 D. 12

12. 若某完全二叉树的深度为h，则该完全二叉树中至少有\_\_\_\_\_个结点

A．2h  B．2h-1 C．2h+1 D．2h-1

13. 若二叉树的前序序列与后序序列的次序正好相反，则该二叉树一定是\_\_\_\_\_\_的二叉树

A．空或仅有一个结点 B．其分支结点无左子树

C．其分支结点无右子树 D．其分支结点的度都为1

14. 任何一棵非空二叉树的叶节点在前序遍历、中序遍历与后序遍历中的相对位置\_\_\_\_\_\_

A．都会发生改变 B．不会发生改变

C．有可能会发生改变 D．部分会发生改变

15. 对于一个数据元素序列，按照逐点插入方法建立一棵二叉排序树，该二叉排序树的形状取决于\_\_\_\_

A．该序列的存储结构 B．序列中数据元素的取值范围

C．数据元素的输入次序 D．使用的计算机的软、硬件条件

16. 对一棵二叉排序树进行\_\_\_\_\_遍历，可以得到该二叉树的所有结点按值从小到大排列的序列

A．前序 B．中序 C．后序 D．按层次

17. 除了前序遍历(DLR),中序遍历(LDR)、后序遍历(LRD)外，二叉树的遍历方法还可以有DRL，RDL和RLD3种。对于一颗二叉排序树，采用\_\_\_\_\_\_遍历方法可以得到该二叉排序树的所有结点按值从大到小排列的序列

A．LDR B．LRD C．RLD D．RDL

18. 在二叉排序树中进行查找的效率与\_\_\_\_\_\_\_相关

A． B．二叉排序树的结点的个数

C．被查找结点的度 D．二叉排序树的存储结构

19. 随机情况下，在具有n个结点的二叉排序树中查找一个结点的时间复杂度为\_\_\_\_\_\_\_

A． B． C． D．

20. 下列名词术语中，与数据存储结构有关系的是\_\_\_\_\_\_\_

A． B．2i-1 C．线索二叉树 D．二叉排序树

21. 平衡二叉树中任意结点的平衡因子只能是\_\_\_\_\_\_\_之一

A. 0,1,2 B. 0,1 C. -1,+1 D. 0,-1,+1

二、填空题

1．任何非空树中有且仅有一个结点没有前驱结点，该结点就是树的\_\_\_\_\_\_\_。

2．树的层次定义为\_\_\_\_\_\_\_\_\_。

3．度为k的树中第i层最多有\_\_\_\_\_\_\_\_\_个结点。

4．深度为h的k叉树最多有\_\_\_\_\_\_\_\_个结点。

5. 非空二叉树一共有\_\_\_\_\_\_\_\_\_基本形态。

6. 非空二叉树中第i层最多有\_\_\_\_\_\_\_\_个结点。

7. 深度为h的二叉树最多有\_\_\_\_\_\_\_\_\_个结点。

8. 具有n个结点的完全二叉树的深度h=\_\_\_\_\_\_\_\_\_\_\_。

9. 若二叉树有n0个叶结点，n2个度为2的结点，则n0与n2的关系是\_\_\_\_\_\_\_\_。

10. 若具有n个结点的非空二叉树有n0个叶结点，则该二叉树有\_\_\_\_\_\_\_\_个度为2的结点，\_\_\_\_\_\_\_\_个度为1的结点。

11. 对具有n个结点的完全二叉树按照层次从上往下，每一层从左往右的次序对所有结点进行编号，编号为i的结点的双亲结点的编号为\_\_\_\_\_\_\_，左孩子的编号为\_\_\_\_\_\_\_，右孩子的编号为\_\_\_\_\_\_\_\_。

12. 若具有n个结点的二叉树采用二叉链表存储结构，则该链表中有\_\_\_\_\_\_\_个指针域，其中有\_\_\_\_\_\_\_\_个指针域用于链接孩子结点，\_\_\_\_\_\_\_个指针域空闲放置着NULL。

13. 二叉树的遍历方式通常有\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_4种。

14. 已知二叉树的前序遍历序列为ABDCEFG,中序遍历序列为DBCAFEG,其后序遍历序列为\_\_\_\_\_\_。

15. 已知某完全二叉树采用顺序存储结构，结点的存放次序为A,B,C,D,E,F,G,H,I,J，该完全二叉树的后序序列为\_\_\_\_\_\_\_\_。

16. 在某二叉树中，若结点A有左孩子Y和右孩子X，则在结点的前序序列、中序序列和后序序列中，结点\_\_\_\_\_\_\_\_\_一定在结点\_\_\_\_\_\_\_的前面。

17. 除了按层次遍历外利用\_\_\_\_\_\_\_\_\_对二叉树进行遍历可以不用堆栈。

18. 对二叉排序树进行\_\_\_\_\_\_\_\_，可以得到该二叉树中所有结点组成的按值有序排列的序列。

19. 采用逐点插入法建立序列的二叉排序树后，查找数据元素62共进行\_\_\_\_\_次元素间的比较。

20. 具有n0个叶结点的哈夫曼树共有\_\_\_\_\_\_\_个结点。

三、解答题

1. 若结点A有3个兄弟（包括A本身），并且B是A 的双亲结点，问结点B的度是多少

2. 若一棵树有n1个度为1的结点，n2个度为2的结点……，nm的结点，试问：该树一共有多少个叶结点（即度为0的结点个数n0）？请写出推导过程。

3. 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。问：

(1). 第k层有多少个结点?

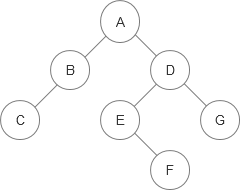
(2). 整棵树有多少个结点

(3). 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，编号为i的结点的双亲结点的编号是什么？编号为i的结点的第j个孩子结点（若存在的话）的编号是什么？

4. 一棵度为2的树与一棵二叉树有什么区别？

5. 请分别画出具有3个结点的树和具有3个结点的二叉树的所有形态

6. 请分别列出如图所示二叉树的所有叶结点与分支结点，并分别指出各结点的度以及所在的层次数。



7. 若一棵具有n个结点的二叉树采用二叉树链表存储结构。试问：该二叉树链表中共有多少个空指针域，写出推导过程

8. 已知某算数表达式的中缀形式为，后缀形式为，请写出其前缀形式（利用二叉树的遍历序列）

9. 已知某二叉树的前序序列为ABDEGCFHIJ,中序序列为DBGEAHFIJC,请写出后序序列

10. 已知某二叉树的中序遍历序列为CBGEAFHD,后序遍历序列为CGEBHFDA，请画出该二叉树的前序线索二叉树的二叉链表结构的表示

11. 已知按前序遍历二叉树的结果为ABC，试问有几种不同的二叉树可以得到这一遍历结果。

12. 请按照算法SORTTREE画出对应于序列(15,20,15,7,9,18,6)的二叉排序树

13. 给定一组权值，请构造出相应的哈夫曼树，并计算其带权的路径长度WPL

14. 试证明具有n0个叶结点的哈夫曼树的分支总数为2(n0-1)

15. 已知非空二叉树采用顺序存储结构，结点的数据信息依次存放于数组BT[0..M]中，（假设结点的数据信息为整型数，若数组元素为0，则表示该元素对应 的结点在二叉树中不存在）。请写出二叉树前序遍历的非递归算法。

16. 已知两棵二叉树都采用二叉链表存储结构，根结点的地址分别为T1和T2。请写一算法，判断两棵二叉树是否相似（即具有相同的形态），并给出相应信息。

17. 已知两棵二叉树都采用二叉链表存储结构，根结点的地址分别为T1和T2。请写一算法，判断两棵二叉树是否是相同的二叉树，并给出相应信息。

18. 已知二叉树采用二叉链表存储结构，根结点地址为T，请写一算法，释放该二叉树中所有结点所占用的空间。

19. 已知二叉树采用二叉链表存储结构，根结点的地址为T，请写一非递归算法统计出该二叉树共有多少个度为1的结点？要求算法中用到的堆栈采用链式存储结构

20. 已知非空二叉树采用二叉链表存储结构根结点地址为T，请写出非递归算法，该算法打印数据信息为item的结点的所有祖先结点。假设数据信息为item的结点不多于一个。

21. 已知非空二叉树采用顺序存储结构，结点的数据信息一次存放于数组中，（若元素值为0，表示该元素对应的结点在二叉树中不存在）。请写一算法，生成该二叉树的二叉链表结构。

22. 已知某具有n个结点的二叉树的前序序列与中序序列分别为和，并且各结点的数据值均不相同。试写一非递归算法生成该二叉树的二叉链表结构。

23. 已知二叉树采用二叉链表存储结构，根结点地址为T。试写一算法，判断该二叉树是否为完全二叉树，

并给出相应信息。

24. 已知二叉树采用二叉链表存储结构，根结点地址为T。试写一算法，删去并释放数据值为key的叶结点。

25. 已知二叉树采用二叉链表存储结构，根结点地址为T。试写一算法，求以数据信息为item的结点为根的子树的深度。

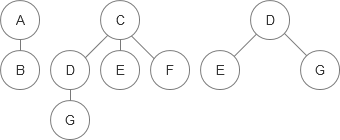
26. 已知二叉树采用二叉链表存储结构，根结点的地址为T。试写一算法，算法的功能是按层次从上到下，每层从右到左的顺序依次列出二叉树所有结点的数据信息。

27. 已知二叉树采用二叉链表存储结构，根结点的地址为T。试写一算法，打印该二叉树所有左子树的根结点的数据信息。

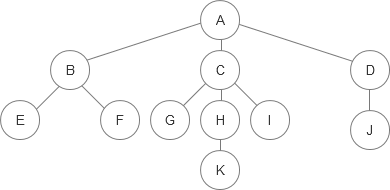
28. 已知二叉树采用二叉链表存储结构，根结点的地址为T。请写一非递归算法，该算法判断二叉树是否为二叉排序树。若是二叉排序树，算法返回1，否则返回0.（假设各结点的数据互不相同）

29. 已知非空二叉树采用二叉链表结构，请写出非递归算法，生成该二叉树的中序线索二叉树。

30. 将图示的树林转换成一颗二叉树。



31. 分别写出如图所示的树的前序遍历序列与后序遍历序列



32. 已知某树林转化为二叉树后所对应的顺序存储结构如下，请画出该树林。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | H | C | E | I | J | D | F | G |

33. 下面关于二叉排序树的说法正确吗，若认为不正确，请分别举一例说明

(1). 任意一个分支结点的值都大于其左孩子（若存在）的值，并且小于或等于其右孩子（若存在）的值，这样的二叉树为二叉排序树

(2).采用“逐点插入法”将1个结点插入二叉排序树，该结点总是插在某个叶结点的下面。

参考答案：

1. 选择题

1、C

2、B

3、D

4、D

5、D

6、B

7、B

8、A

9、A

10、D

11、C

12、D

13、D

14、B

15、C

16、B

17、D

18、A

19、D

20、C

21、D

1. 填空题

1．根结点

2．根结点为第1层，若某结点在第i层，则其孩子结点（若存在的话）在第i+1层

3．

4.

5．4

6．

7、

8．

9、

10、

11、[i/2] 2i 2i+1

12、2n n-1 n+1

13、前序遍历 中序遍历 后序遍历 按层次遍历

14、DCBFGEA

15、HIDJEBFGCA

16、Y X

17、线索二叉树

18、中序遍历

19、3

20、

三、解答题

1．结点B的度为3

2．若设ni分别表示度为i的结点数目，则该树的叶结点总数为：

该结论的推导过程如下：

设树的分支总数为B，有： B=N-1

由于

综上所述：可得结论：

3.

4. 答：度为2的树不一定是有序树，而一个度为2的二叉树一定是有序树

5. 答：略

6. 答：略

7. 答：具有n个结点二叉树若采用二叉链表作为存储结构，则链表中有n+1个存放着NULL的指针域。因为整个链表有2n个指针域，二叉树有n-1个分支（每个分支占用一个指针域），剩余n+1个指针域并未存放结点的指针

8. 答：前缀形式为

9. 答：后序遍历序列：DGEBHJIFCA

10. 答：前序遍历序列：ABCEGDFH

11. 答：有5种不同的儿茶素可以得到这一遍历结果

12. 答：略

13. 答：略

14. 证明：设哈夫曼树有n个结点。由于哈夫曼树没有度为1的结点，设n0和n2分别为叶结点和度为2的结点数目，有n=n0+n2.根据二叉树性质得n2=n0-1,因此有n=2n0-1.若B表示哈夫曼树的分支数目，则有B=n-1=2n0-1-1=2(n0-1).证毕

15. 算法：

#define MaxN 100 // MaxN>=M

void PREORDER(datatype BT[], int M)

{

int STACK[MaxN],i,top=-1;

i=0;

do{

while(i<M&&<BT[i]!=0){

VISIT(BT[i]); //访问结点BT[i]

STACK[++top]=i; //BT[i]的位置i进栈

i=i\*2+1; //找到BT[i]的左孩子位置

}

i=STACK[top--]; //从堆栈中退出一个结点的位置

i=i\*2+2; //找到BT[i]的右孩子的位置

}while(!((i==M||BT[i]==0)&&top<0));

}

16. 算法：

int SIMILAR(BTREE T1,BTREE T2)

{

if(T1 == NULL&&T2 == NULL) //两者都为二叉树，相似

reruen 1;

if(T1 != NULL && T2 !==NULL //左子树和右子树分别都相似

&& SIMILAR(T1->lchild,T2->lchild) &&SIMILAR(T1->rchild, T2->rchild))

return 1;

return 0; //二叉树不相似

}

17. 算法：略

18.

void RELEASE(BTREE&T)

{

// 利用后序遍历的递归算法删除二叉树

if(T!==NULL){

RELEASE(T->lchild);

RELEASE(T->rchild);

free(T);

}

}

19. 算法核心思想：这是一个利用二叉树的遍历操作的简单例子。下面的算法利用了前序遍历的非递归算法。在遍历过程中，每当访问一个结点就判断该结点是否只存在一棵子树，同时记录满足该条件的结点数目

typedef struct node{

BTREE data;

struct node \*link;

}BLNode,\*BLinkList; //定义连接堆栈的结构

int COUNT(BTREE T)

{

// 利用前序遍历的非递归算法统计度为1的结点数目

BTREE p=T;

BLinkList q,top=NULL;

int n=0;

if(T!=NULL)

do{

while(p!=NULL){

if(p->lchild!=NULL&&p->rchild==NULL||

P->rchild!=NULL&&p-lchild==NULL)

n++; //度为1的结点数目增1

q=(BLinkList)malloc(sizeof(BLNode)); //申请一个新的链结点

q->data = p;

q->link = top;

top=q; //以上三条语句将p所指结点的地址进栈

p=p->lchild; //将p移到其左孩子结点

}

p=top->data; //退栈

q=top;

top=top->link;

free(q);

p=p->rchild; //将p移到其右孩子结点

}while(!(p==NULL&&top==NULL));

return n; //返回度为1的结点数目

}

20. 算法核心思想：这里为了简化算法，假设每个结点的数据信息为一个十进制证书，并且约定，一个结点的祖先定义为根结点到该结点的双亲结点路径上的所有结点。因此，本题利用后序遍历二叉树的非递归算法达到目的。在遍历过程中，当找到满足条件的结点时，堆栈中依次保存着该结点的祖先结点。

#define NodeNum 100 //定义二叉树中结点最大数目

void ANCESTOR(BTREE T,int item)

{

BTREE STACK1[NodeNum],p=T;

int STACK2[NodeNum],top=-1,flag;

if(T!=NULL&&T->data!=item) //若二叉树不空且根结点不满足条件

do{

while(p!=NULL){

STACK1[++top]=p; //当前p所指结点的地址进栈

STACK2[top]=0; //标标志0进栈

p=p->lchild; //将p移到其左孩子结点

}

p=STACK1[top]; //栈1的顶结点（地址）退栈送p

flag=STACK2[top--]; //栈2的顶元素（标志）退栈送flag

if(flag==0){

STACK1[++top]=p; //当前p所指结点的地址再次进栈

STACK2[top]=1; //标志1进栈

p=p->rchild; //将p移到其右孩子结点

}else{

if(p->data==item){ //若p所指结点满足条件

while(top!=1) //依次输出p所指结点的祖先

printf("%4d",STACK1[top--]->data);

break; //退出算法，结束

}else{

p=NULL; //置p为NULL

}

}

}while(!(p==NULL&&top==-1));

}

21. 算法：

#define MaxN 100

BTREE BUILDBTREE(datatype BT[])

{

BTREE T,PTR[MaxN];

int i,j;

PTR[0] = (BTREE)malloc(sizeof(BTNode)); //申请根结点的空间

PTR[0] -> data=BT[0];

PTR[0] -> lchild=NULL;

PTR[0] -> rchild=NULL;

T=PTR[0]; //以上语句建立根结点

for(i=1;i<MaxN;i++)

if(BT[i]!=0){ //对应结点在二叉树中存在

PTR[i] = (BTREE)malloc(sizeof(BTNode));

PTR[i]->data = BT[i];

PTR[i]->lchild = NULL;

PTR[i]->rchild = NULL;

j=(i-1)/2;

if(i-2\*j-1==0)

PTR[j]->lchild=PTR[i];

else

PTR[j]->rchild=PTR[i];

}

return T;

}

22. 算法：

int SEARCHPOS(datatype INOD[], int n, datatype item)

{

int i;

for(i=0;i<n;i++)

if(INOD[i]==item)

return i+1;

}

void INSERTITEM(BTREE &T, datatype item)

{

BTREE p,q;

int ord;

p = (BTNode)malloc(sizeof(BTREE));

p->data=item;

p->lchild=p->rchild=NULL;

if(T==NULL)

T=p;

else{

ord=SEARCHPOS(INOD, n, item)

q = T;

while(1)

if(ord<SEARCHPOS(INOD,n,q->data))

if(q->lchild != NULL)

q=q->lchild;

else{

q->lchild=p;

break;

}

else

if(q->rchild!=NULL)

q=q->rchild;

else{

q->rchild=p;

break;

}

}

}

BTREE BUILDBTREE(datatype PREOD[], int n)

{

BTREE T = NULL;

int i;

for(i=0;i<n;i++)

INSERTITEM(T, PREOD[i]);

return T;

}

23. 算法：

#define NodeNum 100

int TESTCOMTREE(BTREE T)

{

BTREE QUEUE[NodeNum], p;

int front,rear,flag=1,comflag=1;

if(T!=NULL){

QUEUE[0]=T;

front=-1;

rear=0;

while(front<rear){

p=QUEUE[++front];

if(p->lchild==NULL){

flag=0;

if(p->rchild==NULL){

comflag = 0;

}

}else{

comflag = flag;

QUEUE[++rear]=p->lchild;

if(p->rchild!=NULL)

QUEUE[++rear]=p->rchild;

else

flag=0;

}

}

}

return comflag;

}

24. 略

25. 算法：

#define NodeNum 100

int GET\_SUB\_DEPTH(BTREE T,datatype item){

BTRE STACK[M],p=T;

int top=-1;

if(T!=NULL)

do{

while(p!=NULL){

if(p->data==item)

return GET\_DEPTH(p);

STACK[++top]=p;

p=p->lchild;

}

p=STACK[top--];

p=p->rchild;

}while(!(p==NULL&&top==-1))

return 0;

}

int GET\_DEPTH(BTREE p)

{

int m,n;

if(!p)

return 0;

else{

m=GET\_DEPTH(p->lchild);

n=GET\_DEPTH(p->rchild);

return (m>n!m:n)+1;

}

}

26. 算法：

#define NodeNum 100

void LAYERORDER(BTREE T){

BTREE QUEUE[M],p;

int front,rear;

if(T!=NULL){

QUEUE[0]=T;

front = -1;

rear=0;

while(front<rear){

p=QUEUE[++front];

printf("%d", p->data);

if(p->lchild!=NULL)

QUEUE[++rear]=p->lchild;

else(p->rchild!=NULL)

QUEUE[++rear]=p->rchild;

}

}

}

27. 算法：

#define NodeNum 100

void PRINTLEFT(BTREE T)

{

BTREE STACK[M],p=T;

int top=-1;

if(T!=NULL)

do{

while(p!=NULL){

STACK[++top]=p;

p=p->lchild;

if(p!=NULL)

PRINT(p->data);

}

p = STACK[top--];

p=p->rchild;

}while(!(p==NULL&&top==-1));

}

28. 算法

#define NodeNum 100

int TESTSORTTREE(BTREE T)

{

BTREE STACK[NodeNum],p=T;

int top=-1;

datatype priordata = MinValue;

if(T!=NULL)

do{

while(p!=NULL){

STACK[++top]=p;

p=p->lchild;

}

p=STACK[top--];

if(p->data<priordata)

return 0;

priordata = p->data;

p=p->rchild;

}while(!(p==NULL&&top==-1))

return 1;

}

29. 算法：

#define NodeNum 100

TBTREE INTHREAD(TBTREE T)

{

TBTREE HEAD,p=T,prior,STACK[NodeNum];

int top=-1;

HEAD=(TBTREE)malloc(sizeof(TBNode));

HEAD ->lchild=T;

HEAD->rchild=HEAD;

HEAD->lbit=1;

HEAD->rbit=1;

prior=HEAD;

do{

while(p!=NULL){

STACK[++top]=p;

p=p->lchild;

}

p=STACK[top--];

if(p->lchild==NULL){

P->lchild=prior;

p->lbit=0;

}

else

p->lbit=1;

if(prior->rchild==NULL){

prior->rchild=p;

prior->rbit=0;

}

else{

prior->rbit=1;

}

prior=p;

p=p->rchild;

}while(!(p==NULL&&top==-1));

prior->rchild=HEAD;

prior->rbit=0;

return HEAD;

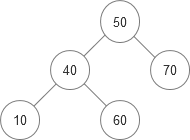
}

30. 略

31. 略

32. 略

33. 答：（1）不正确，如下图二叉树，完全符合体重叙述，但不是二叉排序树



（2）不正确，如下图，如果有数据信息为60的新结点插入，新结点就不是插入叶结点下面，而是作为此时数据信息为50的根结点的右孩子插入二叉树中。

