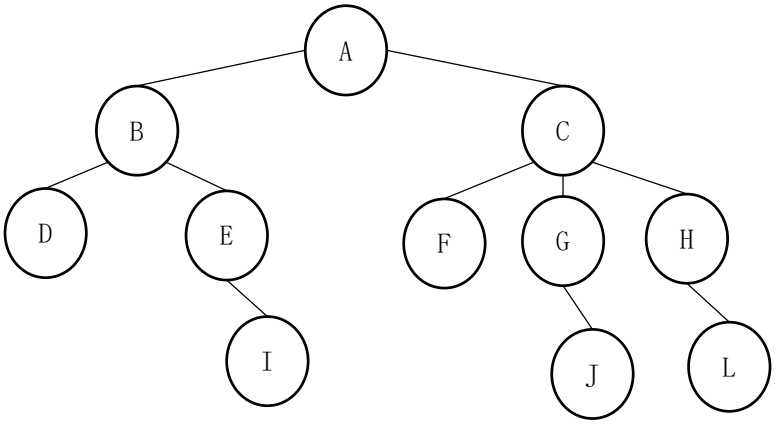
16337341_朱志儒_数据结构作业（三）

1、 树状图：



（1） 根结点：a；
叶子结点：i，d，f，j，l；
g 的双亲：c；
g 的祖先：c，a；
g 的孩子：j；
e 的子孙：i；
e 的兄弟：d；
f 的兄弟：g，h；

（2） b 的层次：2；
h 的层次：3；
树的深度：4；
以结点 c 为根的子树的深度：3；

2、 （1）第 i(1<=i<=h)层的结点个数：k^{i-1}

（2）编号 i 的双亲结点的编号：$P=(1-k^{h-1})/(1-k)+[(i-(1-k^{h})/(1-k))/k]+1$
（其中 $h=[\log_k(1-i+ki)]$）

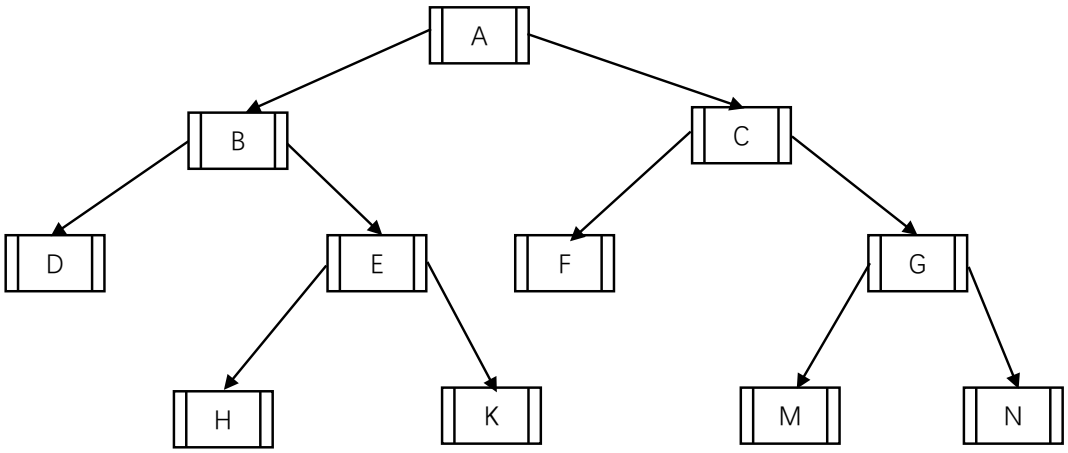
（3）编号 i 的结点的第 j 个孩子结点编号：$C=(1-k^{h+1})/(1-k)+k(i-(1-k^{h})/(1-k)-1)+j$
（其中 $h=[\log_k(1-i+ki)]$）

（4）编号 i 结点有右兄弟的条件：$(1-k^{h})/(1-k)+ka < i < (1-k^{h})/(1-k)+k(a+1)$
（其中 $a=[(i-(1-k^{h})/(1-k))/k]$，$h=[\log_k(1-i+ki)]$）
右兄弟的编号：i+1

3、 （1）顺序存储：双亲表示法

下标	info	parents
0	A	-1
1	B	0
2	C	0
3	D	1
4	E	1
5	H	4
6	K	4
7	F	2
8	G	2
9	M	8
10	N	8

链接存储：二叉链表

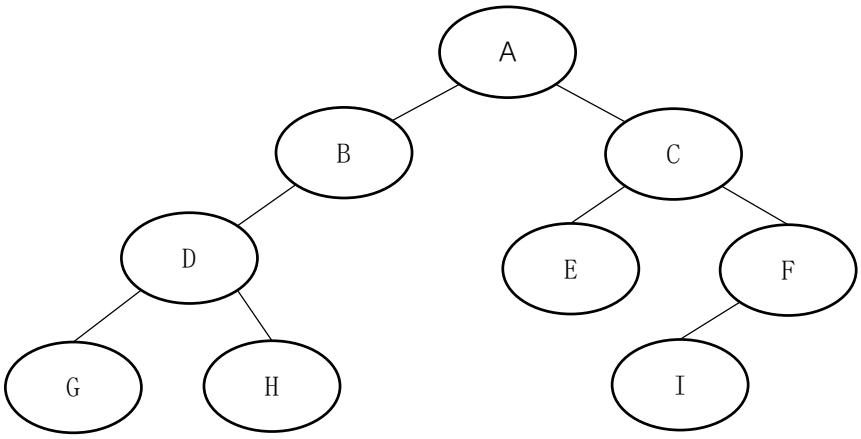


（2）先序：a, b, d, e, h, k, c, f, h, m, n;

中序：d, b, h, e, k, a, f, c, m, g, n;

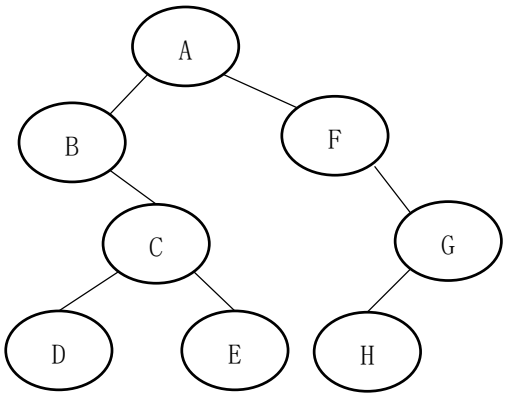
后序：d, h, k, e, b, f, m, n, g, c, a;

4、 二叉树：



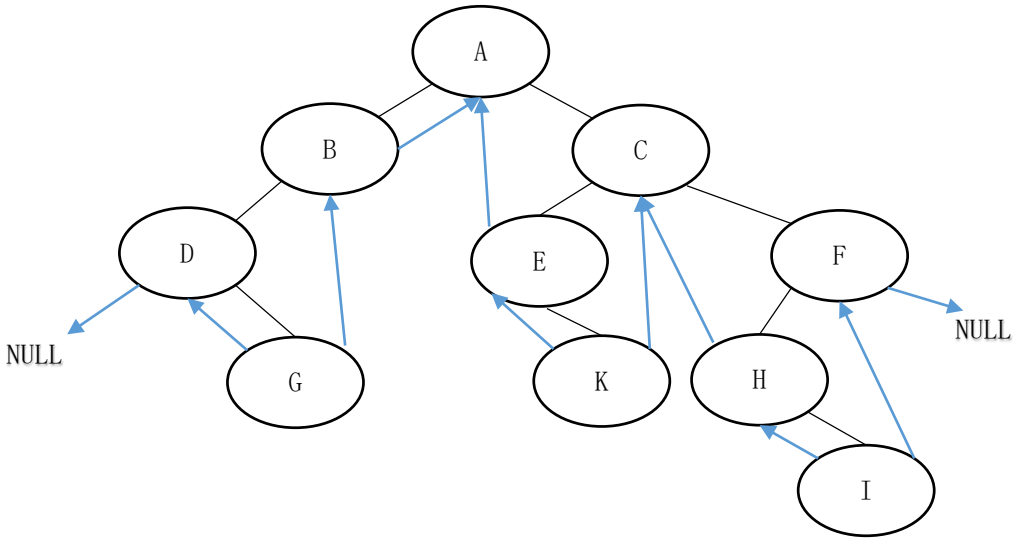
后序遍历序列：G, H, D, B, E, I, F, C, A;

5、 二叉树：

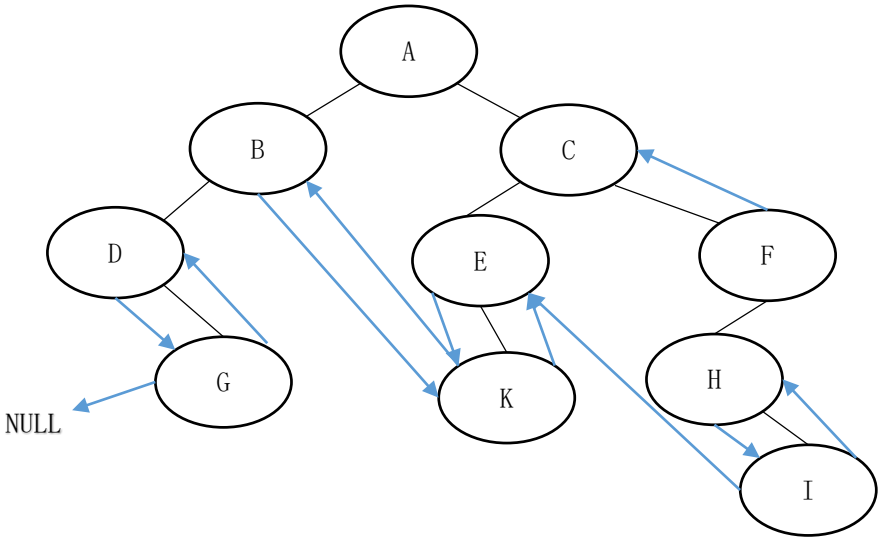


先序序列：A, B, C, D, E, F, G, H

6、 中序线索树：



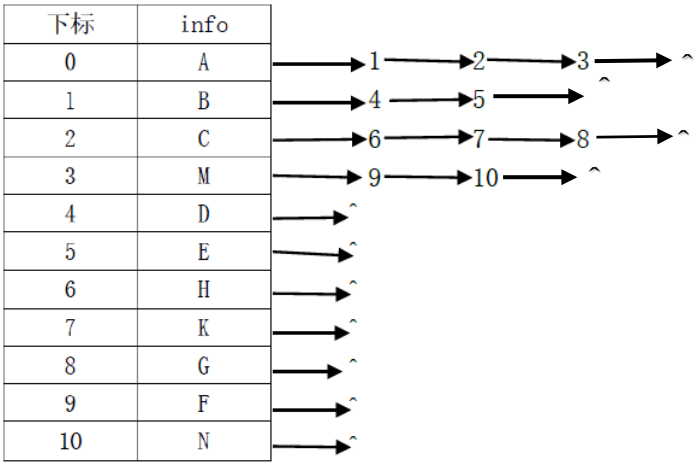
后序线索树：



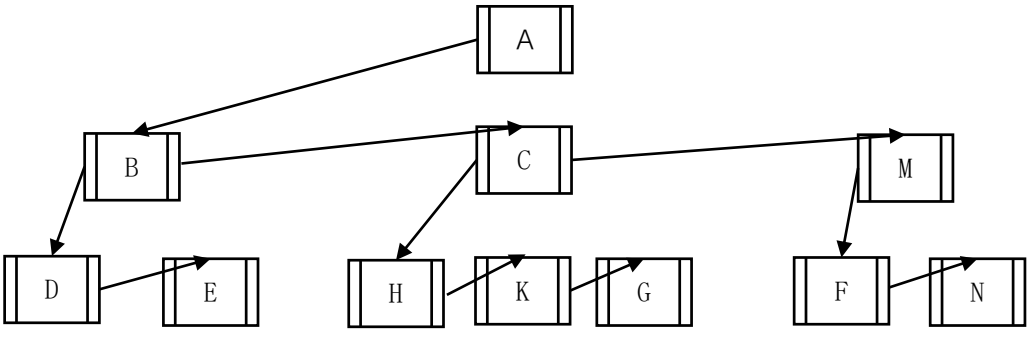
7、 （1）双亲表示法：

下标	info	parents
0	A	-1
1	B	0
2	C	0
3	M	0
4	D	1
5	E	1
6	H	2
7	K	2
8	G	2
9	F	3
10	N	3

孩子表示法：



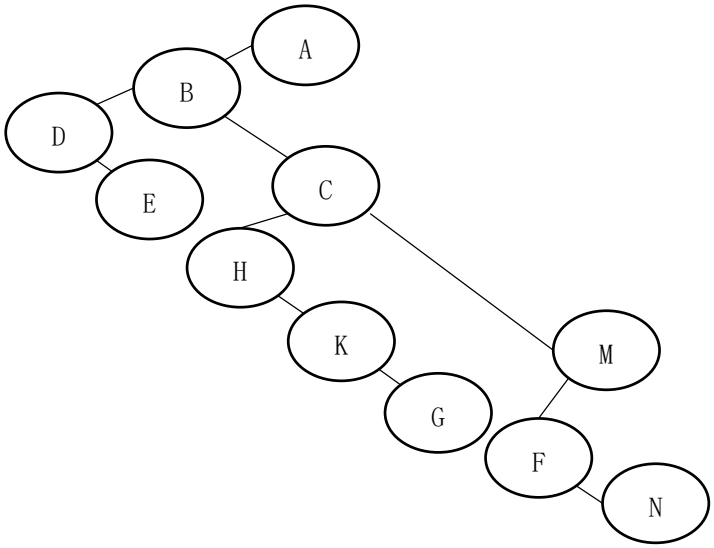
孩子兄弟表示法：



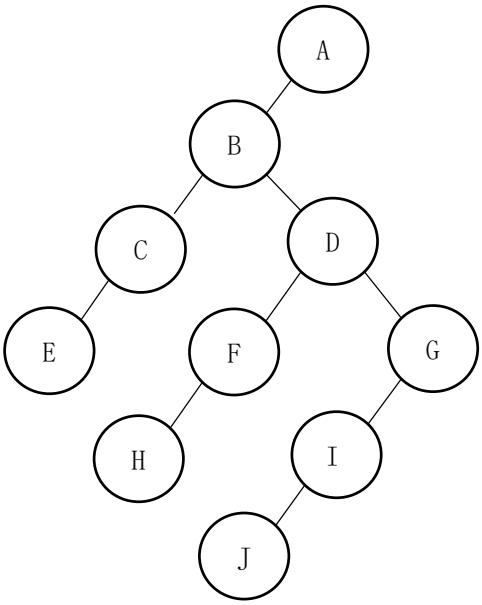
（2）先序：A，B，D，E，C，H，K，G，M，F，N；

后序：D，E，B，H，K，G，C，F，N，M，A；

（3）



8、 （1）二叉树 t 的逻辑结构：



（2）前序：A, B, C, E, D, F, H, G, I, J

中序：E, C, B, H, F, D, J, I, G, A

后序：E, C, H, F, J, I, G, D, B, A

9、

i \ j	前序遍历 n 先被访问	中序遍历 n 先被访问	后序遍历 n 先被访问
n 在 m 的左边	✔	✔	✔
n 在 m 的右边			✔
n 是 m 的祖先	✔	✔	
n 是 m 的儿子		✔	✔

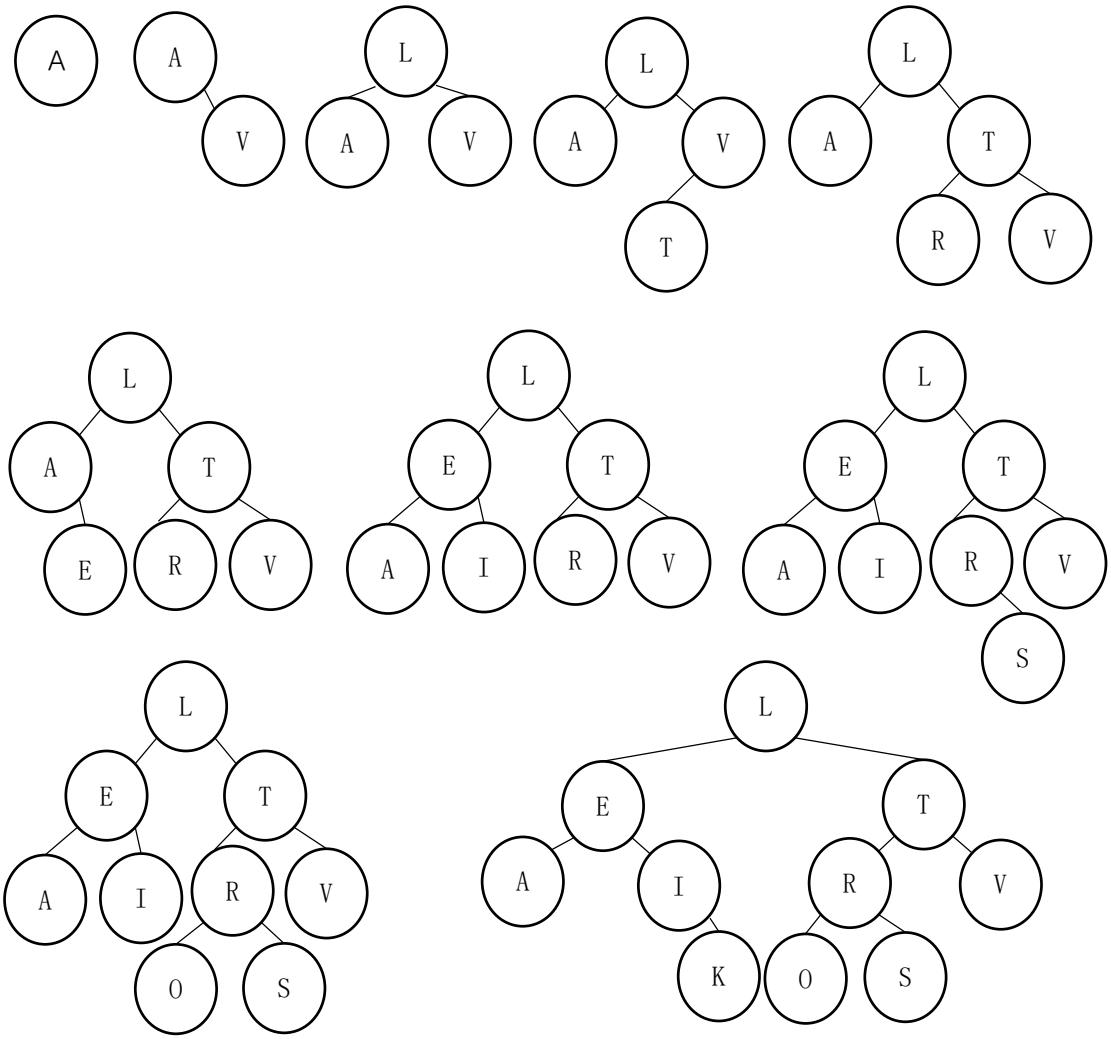
10、后序遍历非递归算法：

```
void postorder(bitree *root) {
    stack<bitree*> data;
    bitree* ptr;                                    //当前访问的结点
    bitree* pre = nullptr;                          //上一次访问的结点
    data.push(root);
    while (!data.empty()) {
        ptr = data.top();
        if (ptr->lchild == nullptr && ptr->rchild == nullptr || pre != nullptr
&& (pre == ptr->lchild || pre == ptr->rchild)) {//左右子树均为nullptr时，可以
            operate(ptr);                           //操作；左右子树均被操作后，可以
            data.pop();                             //操作
            pre = ptr;
        }
        else if (ptr->rchild != nullptr) {     //先将右子树入栈,再将左子树入栈,
            data.push(ptr->rchild);             //可以使得出栈时先访问左子树
        }
        else if (ptr->lchild != nullptr) {
            data.push(ptr->lchild);
        }
    }
}
```

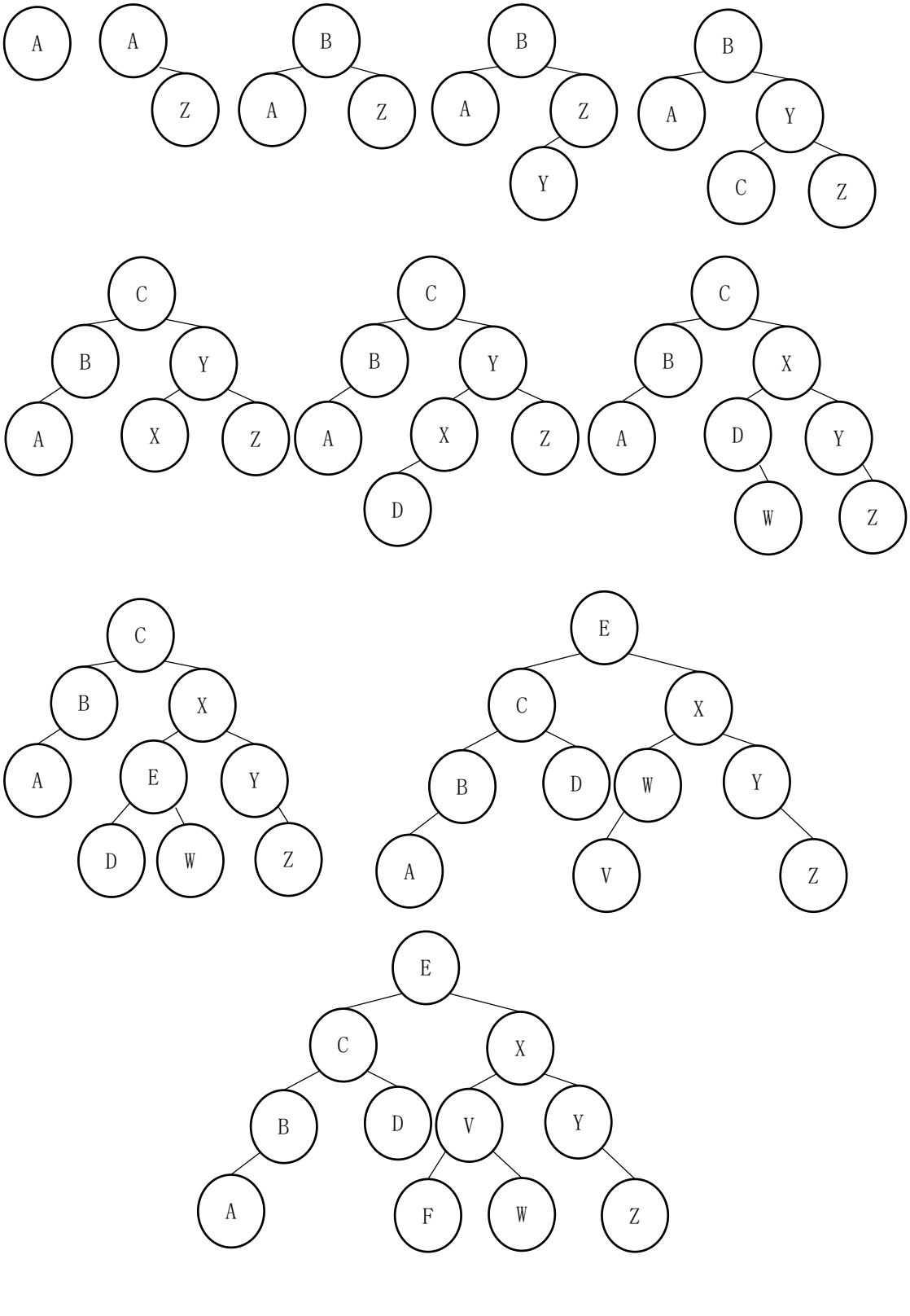
11、算法：使用前序遍历的方式查找

```
bitree *findparent(bitree *root, int value) {
    if (root->lchild == nullptr && root->rchild == nullptr) return nullptr;
    if (root->lchild != nullptr && root->lchild->val == value || root->rchild !=
nullptr && root->rchild->val == value) return root;
    bitree *ptr = nullptr;
    if (root->lchild != nullptr) ptr = return findparent(root->lchild, value);
    if (ptr == nullptr && root->rchild != nullptr) return findparent(root->rchild,
value);
    else return ptr;
}
```

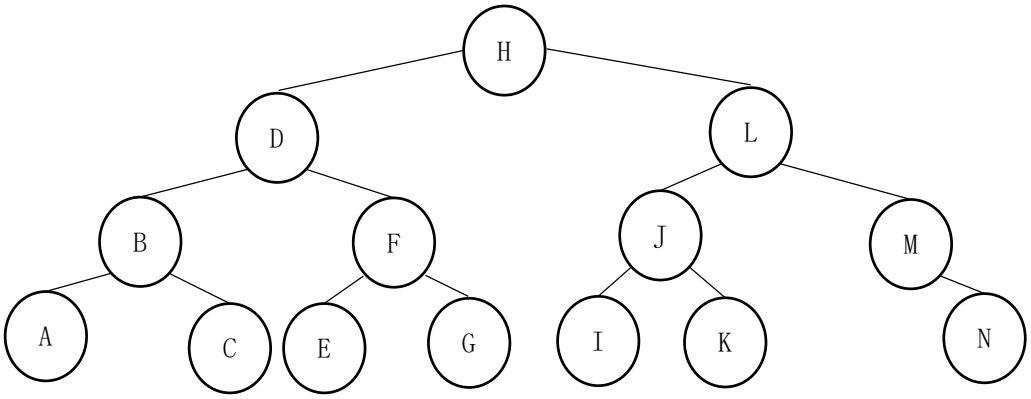
12、（1）


A
A
V
L
A
V
L
A
V
T
L
A
T
R
V
L
A
T
E
R
V
L
E
T
A
I
R
V
L
E
T
A
I
R
V
S
L
E
T
A
I
R
V
O
S
L
E
T
A
I
R
V
K
O
S

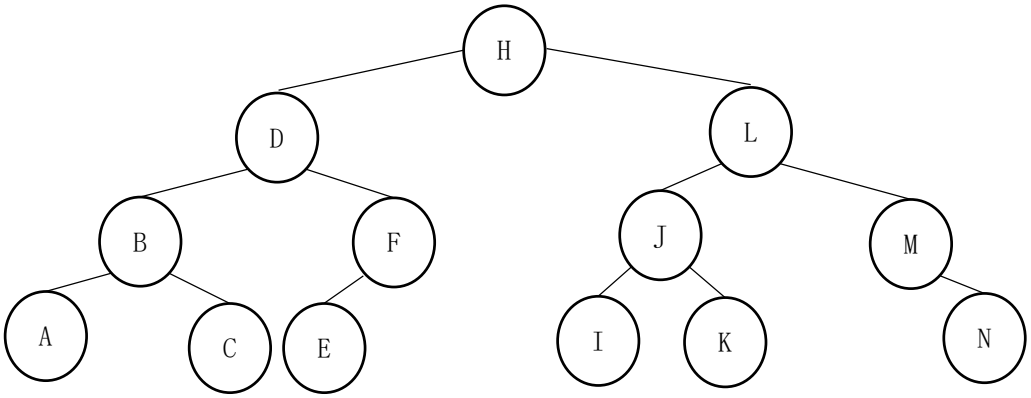
（2）


A
A
Z
B
A
Z
B
A
Z
Y
B
A
Y
C
Z
C
B
Y
A
X
Z
C
B
Y
A
X
Z
D
C
B
X
A
D
Y
W
Z
C
B
X
A
E
Y
D
W
Z
E
C
X
B
D
W
Y
A
V
Z
E
C
X
B
D
V
Y
A
F
W
Z

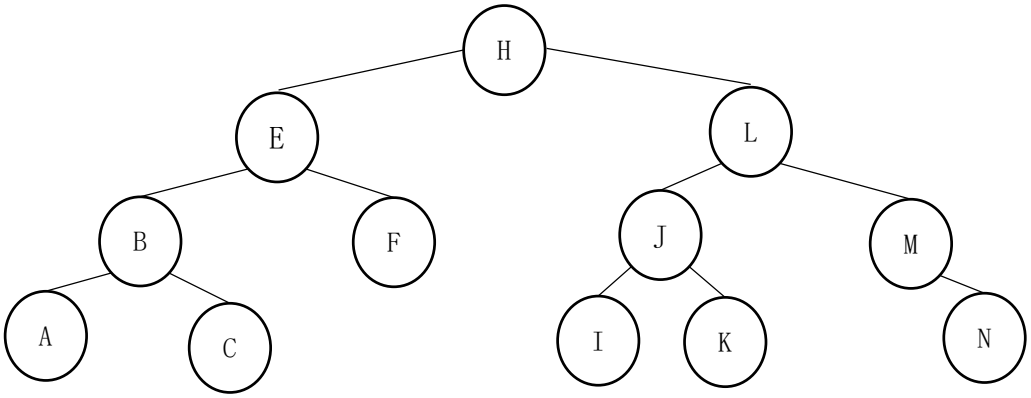
13、建立 AVL 树：



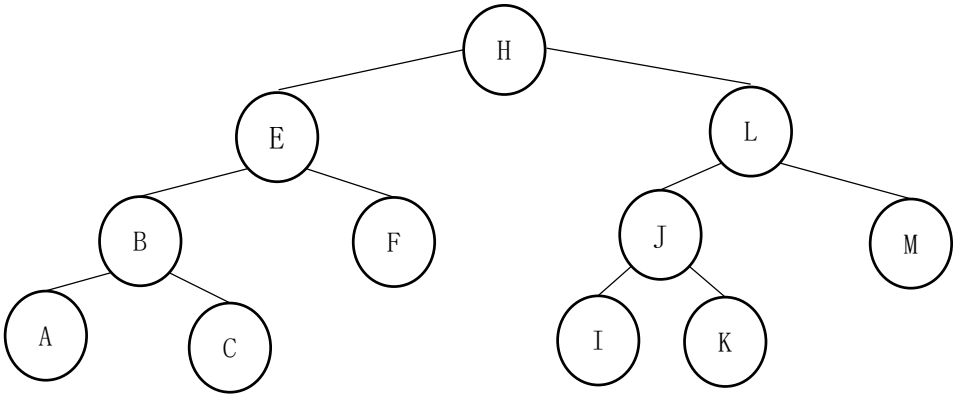
删除 G：



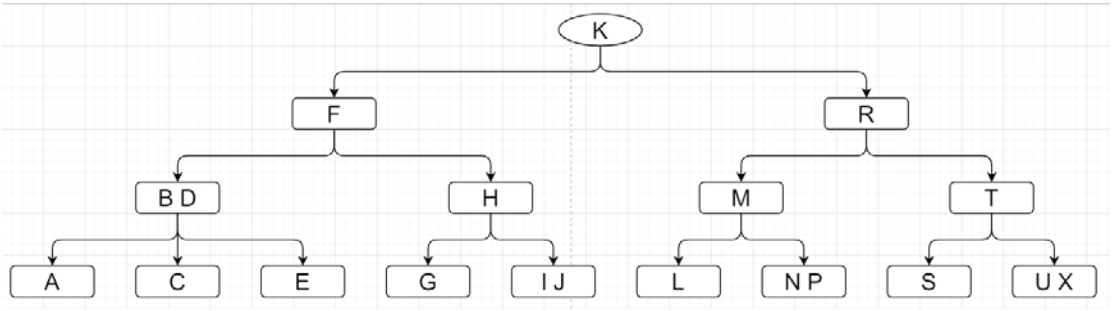
删除 D：



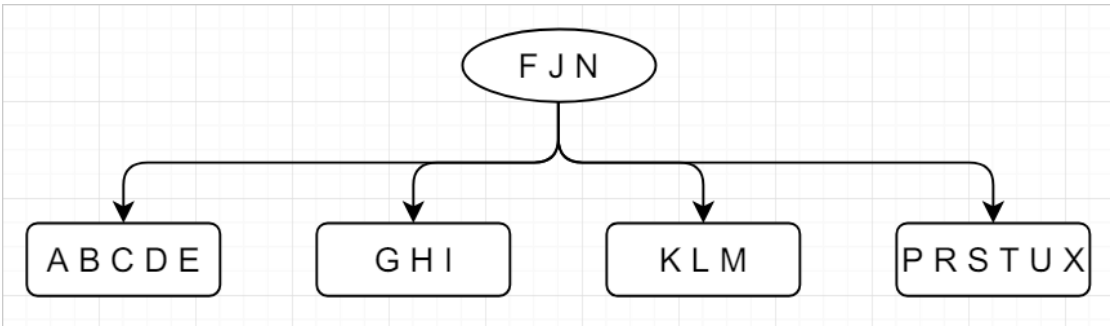
删除 N：



14、3 阶 B-树：

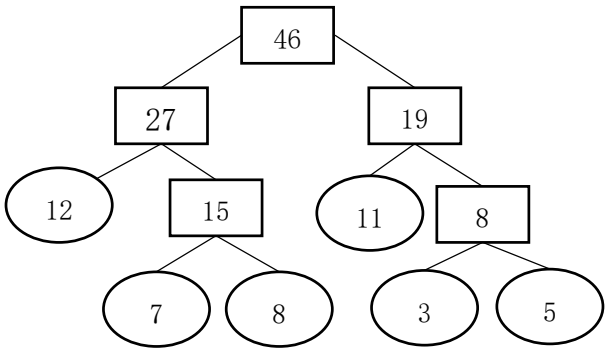


7 阶 B-树：



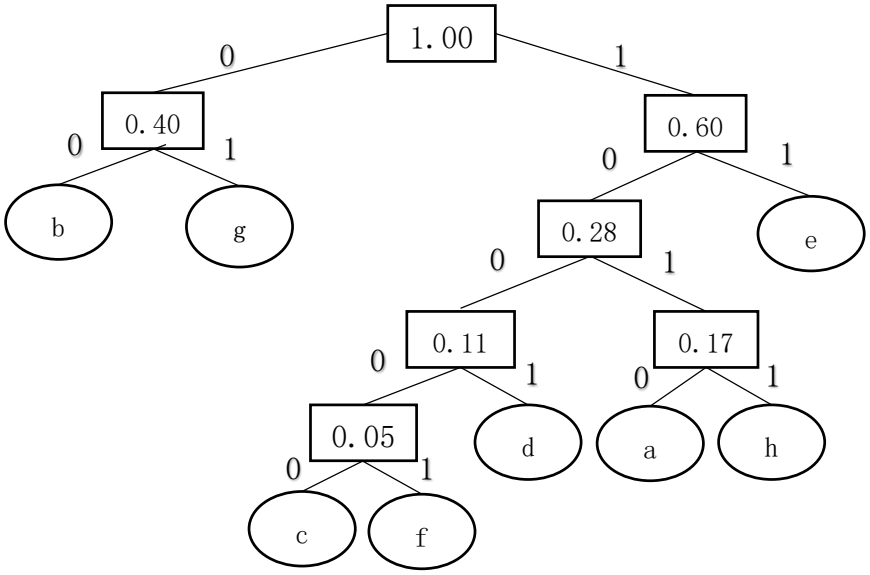
15、产生高度 3 的 5 阶 B-树的最少元素数目是 17

16、



WPL = (12 + 11) * 2 + (7 + 8 + 3 + 5) * 3 = 115

17、（1）



（2）a:1010;

b:00;

c: 10000;

d:1001;

e:11;

f:10001;

g:01;

h:1011