Chapter 5 树和二叉树

书P246. 5.4

解：设度为0的结点个数为n_0，结点总数为 n，分支数为B，则

$$n=n_0+n_1+n_2+\ldots+n_m$$

$$n=B+1=n_1+2n_2+\ldots+mn_m+1$$

解上述两式得：$n_0 = 1+\sum_{i=1}^{m}(i-1)\ n_i$

补充： 已知一棵含有n个结点的树中，只有度为k的分支结点和度为0的叶子结点，问该树中有多少个叶子结点？

解：设度为0和度为k的结点个数分别为n_0和n_k，
结点总数为 n，分支数为B，则

$$n=n_0+n_k$$

$$n=B+1=kn_k+1$$

解上述两式得：$n_0=(nk-n+1)/k$

补充： 在具有n个结点的满二叉树中，度为0、度为1和度为2的结点个数分别是多少？在具有n个结点的完全二叉树中，度为0、度为1和度为2的结点个数又分别是多少？

书P247. 5.9

解：（1）完全二叉树的叶结点只位于层次最高的两层，由于第k层有m个叶子，因此该完全二叉树最少的结点个数为：$2^{k-1}-1+m$

$2^{k-1}-1$：前k-1层的结点个数

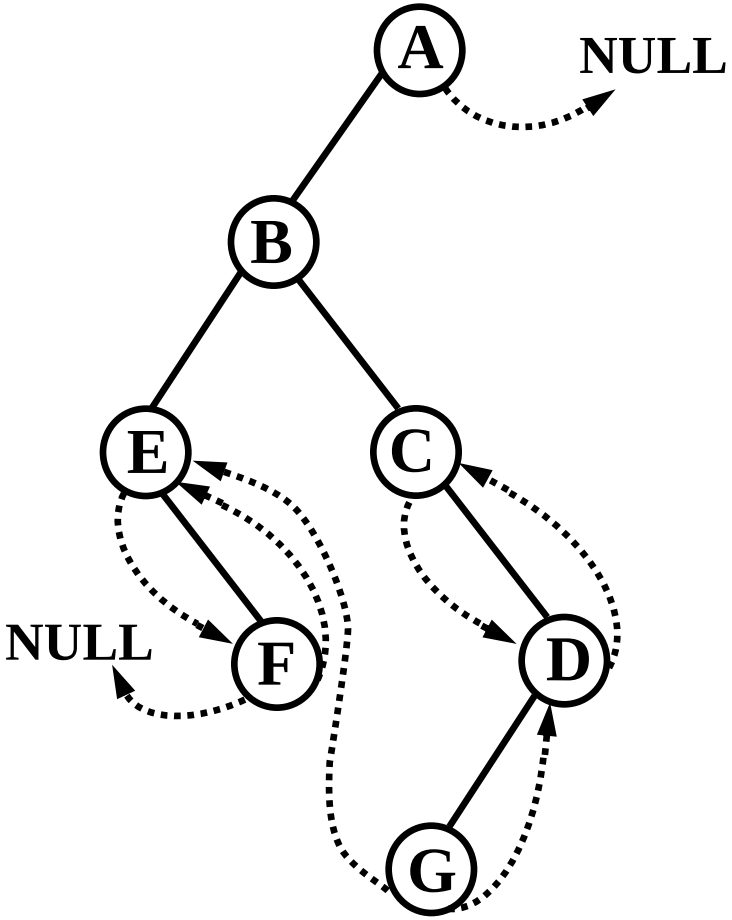
最多情况下，前k层必满，而且第k层中除去m个叶子外，其余每个结点都有两个孩子。因此该完全二叉树最多的结点个数为：$2^k-1+(2^{k-1}-m)\times 2= 2^{k+1}-2m-1$

（2）该完全二叉树的深度为 k 或 k+1

补充题：

画出下列二叉树的后序线索：

（后序遍历序列：**FEGDCBA**）



课堂练习：

画出和下列已知序列对应的树T:

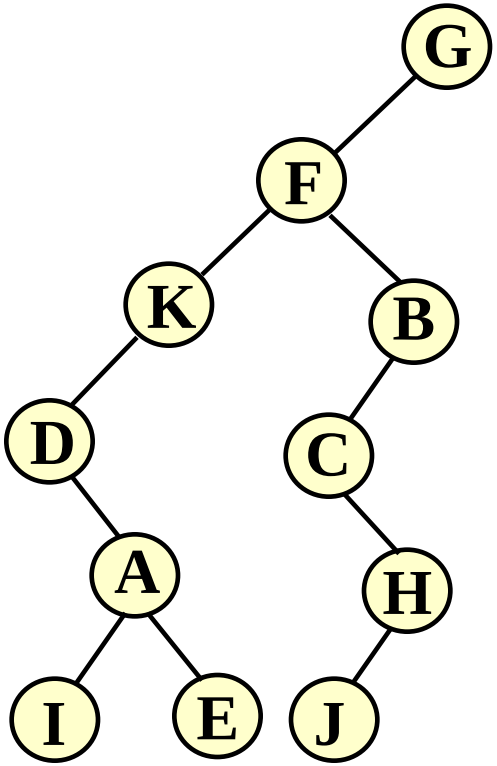
树的先根次序访问序列为：GFKDAIEBCHJ；

树的后根次序访问序列为：DIAEKFCJHBG。

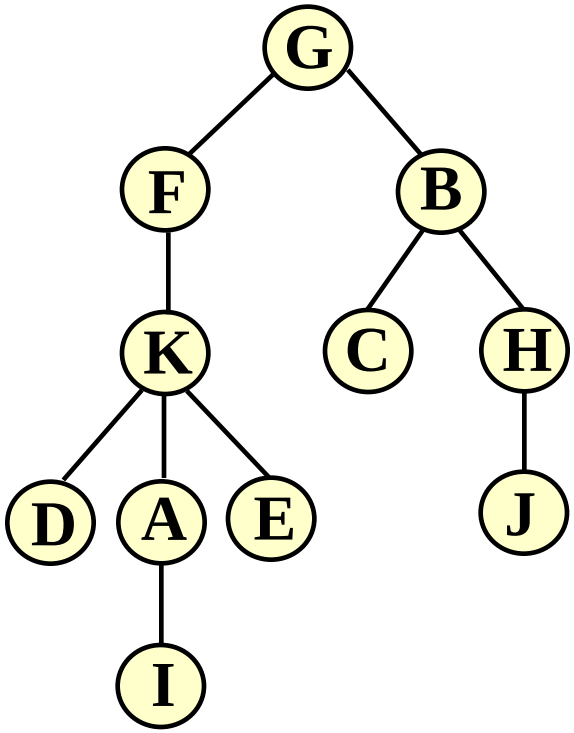
二叉树

对应 ⇒ 前序序列

中序序列 ⇒ 对应



二叉树

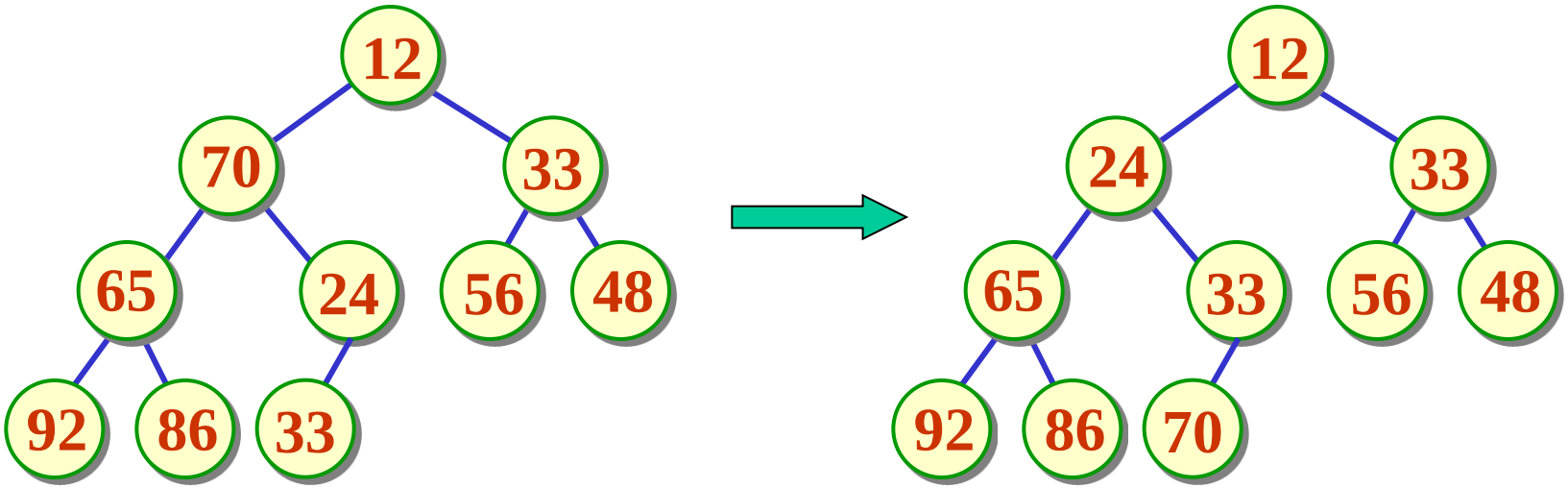


树T

书P247-248. 5.16 题目改为：判断以下序列是否是堆？若是堆，指出是最小堆还是最大堆；若不是堆，将它调整为最小堆。

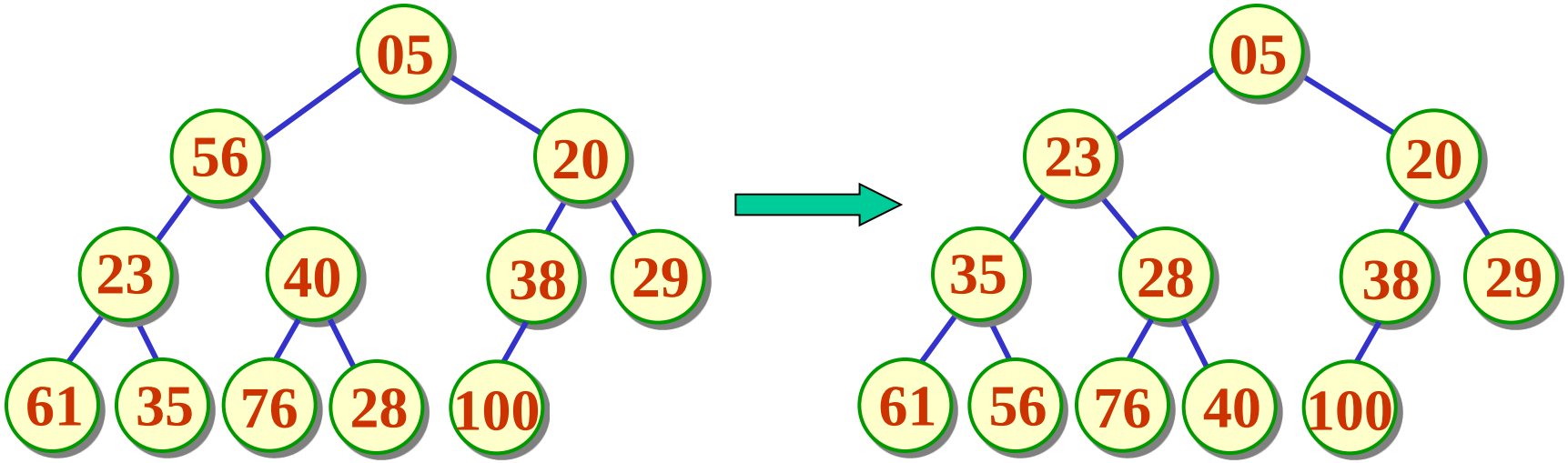
解：（1）、（3）是堆，是最大堆。

（2）不是堆，调整的最小堆如下：



（12, 24, 33, 65, 33, 56, 48, 92, 86, 70）

（4）不是堆，调整的最小堆如下：



（05, 23, 20, 35, 28, 38, 29, 61, 56, 76, 40, 100）

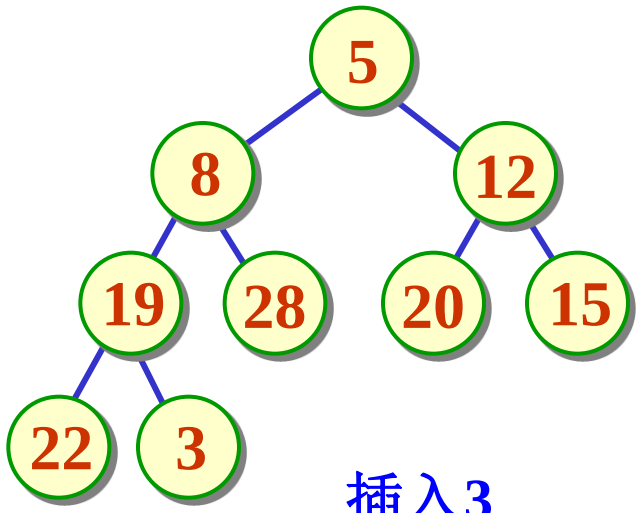
（2009考研题）已知关键字序列 5, 8, 12, 19, 28, 20, 15, 22 是小根堆（最小堆），插入关键字3，调整后得到的小根堆是（ A ）。

A. 3, 5, 12, 8, 28, 20, 15, 22, 19

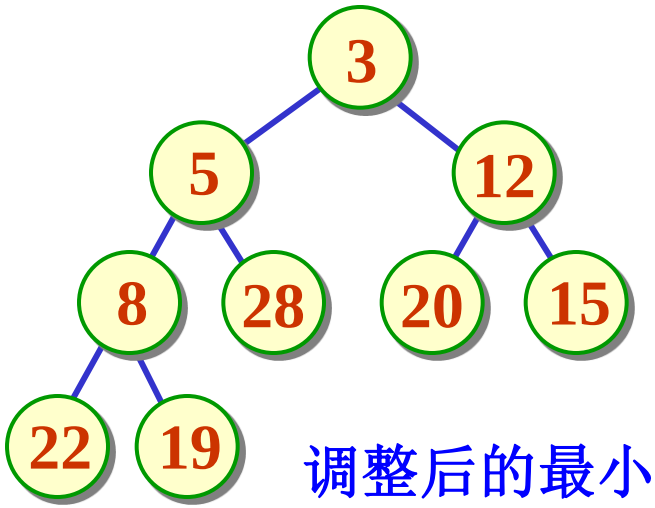
B. 3, 5, 12, 19, 20, 15, 22, 8, 28

C. 3, 8, 12, 5, 20, 15, 22, 28, 19

D. 3, 12, 5, 8, 28, 20, 15, 22, 19



插入3

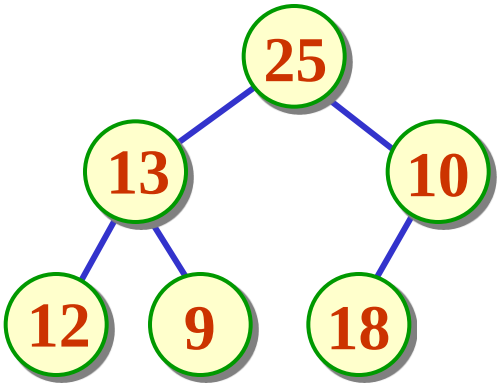


调整后的最小堆

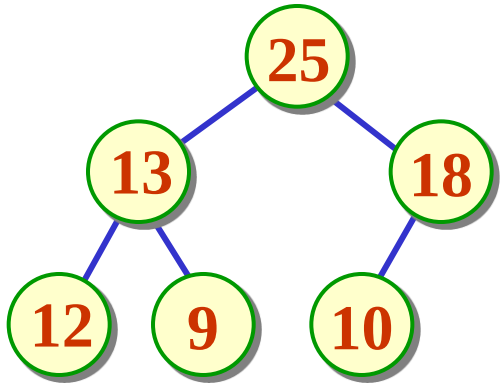
（2011考研题）：

已知序列 25, 13, 10, 12, 9 是大根堆，在序列尾部插入新元素18，将其再调整为大根堆，调整过程中元素之间进行的比较次数是（ **B** ）。

A. 1　　B. 2　　C. 4　　D. 5



插入18



调整后的最大堆（大根堆）

（2010考研题）对n（n≥2）个权值均不相同的字符构造哈夫曼树。下列关于该哈夫曼树的叙述中，错误的是（A）

A. 该树一定是一棵完全二叉树

B. 树中一定没有度为1的结点

C. 树中两个权值最小的结点一定是兄弟结点

D. 树中任一非叶结点的权值一定不小于下一层任一结点的权值

（2013考研题）已知三叉树*T* 中6个叶结点的权分别是2, 3, 4, 5, 6, 7，*T* 的带权（外部）路径长度最小是（B）

A. 27　　B. 46　　C. 54　　D. 56

（2014考研题）5个元素有4种编码方案，下列不是前缀编码的是（D）

A、01, 0000, 0001, 001, 1

B、011, 000, 001, 010, 1

C、000, 001, 010, 011, 100

D、0, 100, 110, 1110, 1100

书P248. 5.23

（1）统计二叉树中叶结点个数

```
template <class T>
int BinaryTree<T>::Leaves(BinTreeNode<T> *subTree) const {
//利用二叉树前序遍历算法计算二叉树的叶结点个数
    if (subTree == NULL) return 0;          //空树
    if (subTree->leftChild==NULL &&
            subTree->rightChild==NULL) return 1;
    return Leaves(subTree->leftChild)+Leaves(subTree->rightChild);
}
```

书P248. 5.23

（2）交换二叉树中每个结点的左右子女（镜像）

```
template <class T>
void BinaryTree<T>::Exchange(BinTreeNode<T> *subTree) {
//利用二叉树前序遍历算法交换二叉树中每个结点的左右子女
    BinTreeNode<T> *temp;
    if (subTree == NULL) return;
    if (subTree->leftChild!=NULL || subTree->rightChild!=NULL) {
        temp = subTree->leftChild;
        subTree->leftChild = subTree->rightChild;
        subTree->rightChild = temp;
        Exchange(subTree->leftChild);
        Exchange(subTree->rightChild);
    }
}
```

书P248. 5.26

```
template <class T>
void Find_Print(BinaryTree<T> &BT, int n, T x, T path[], int
    &count) {
//按二叉树后序遍历的非递归算法求结点x的所有祖先结点
    typedef struct {
       BinTreeNode<T> *ptr;        //二叉树结点指针
       int tag;                    //退栈标记
    } snode;
    snode *S;    snode w;     int i, top;
    BinTreeNode<T> * p = BT.getRoot();
    S = new snode[n];    top = -1;    //初始化栈
    do {
        while (p != NULL) {
            w.ptr = p;  w.tag = 0;
            S[++top] = w;              //进栈
            p = p->leftChild;
        }
```

```
        int continue1 = 1;                    //继续循环标记, 用于tag=1
        while (continue1 && top!=-1) {
            w = S[top--];   p = w.ptr;        //退栈
            switch (w.tag) {                  //判断栈顶的tag标记
                case 0:  w.tag = 1;   S[++top]=w;
                         continue1 = 0;
                         p = p->rightChild;   break;
                case 1:  if (p->data==x) {    //找到x结点
                            for (i=0; i<=top; i++) path[i]=S[i].ptr->data;
                            count=top+1;   delete [ ]S;    return;
                         }
            }
        }
    } while (top!=-1);          //继续遍历其他结点
    count=0;   delete [ ]S;   //未找到x结点
}
```

补充：书P249-250. 5.36

假设一棵树采用父结点表示法，父结点指针数组为 int parent[MaxSize]，根结点存于parent[0]，求树中两个结点p和q的最近公共祖先结点。（常见面试题目）

```
int CommonAncestry(int parent[], int MaxSize, int p, int q)
{
    int i, j;
    for (i=p;  i!=-1;  i=parent[i])
        for (j=q;  j!=-1;  j=parent[j])
            if (i==j) return i;
}
```