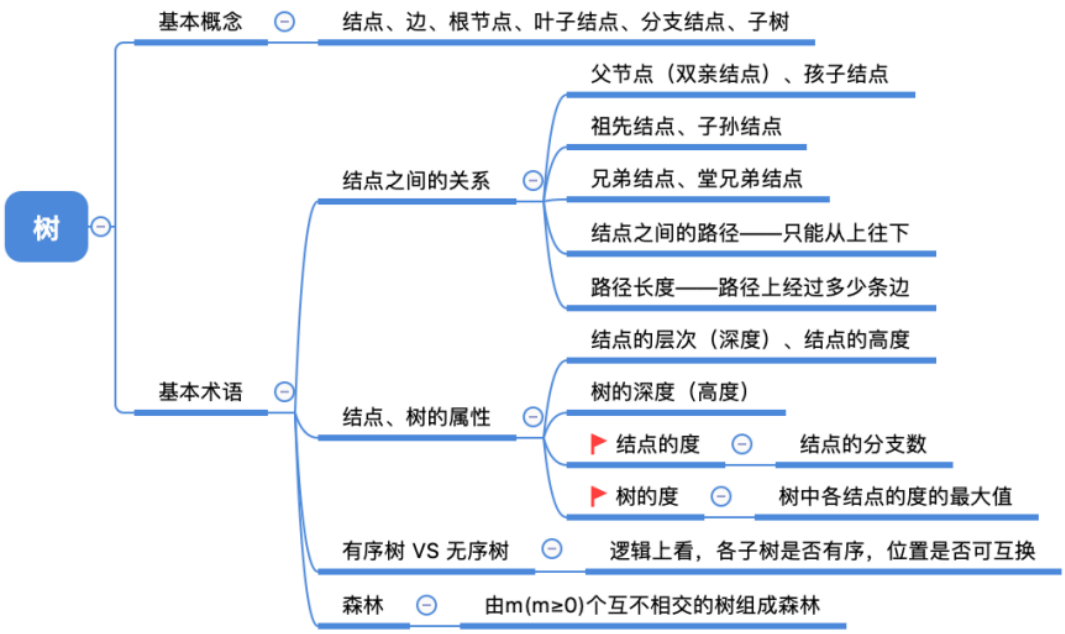
# 第五章、树和二叉树

## 一、基本概念

### 1.树的定义及基本术语

**总览：**



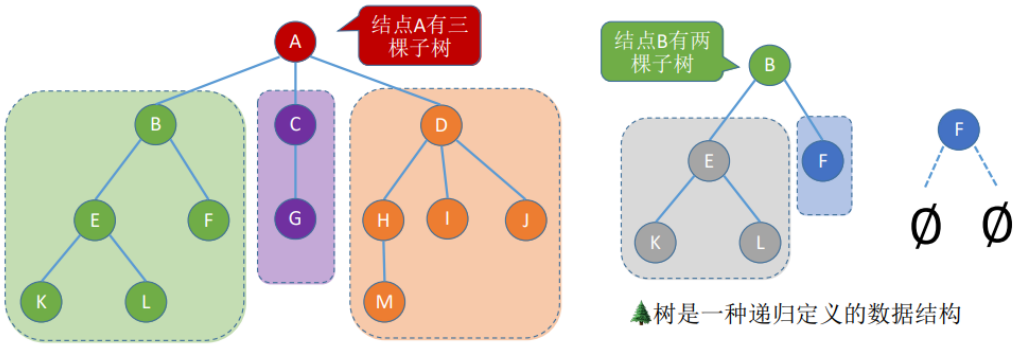

1. 树的定义：树是n个结点的有限集合。（树的定义是一种递归定义。）

   当n=0，为 *空树* ；

   当n>0，需要满足一下两个条件：

   （1） *有且仅有一个特定的称为根的结点* ；

   （2）取余结点可以分为m个 *互不相交的有限集合，其中每个集合本身又是一棵树* ，并称为根的子树。




2. 树的基本术语

   根结点：非空树中没有前趋的结点；

   结点的度：结点拥有的子树数；

树的度：树内各结点的度的最大值；

结点的深度：

树的深度：树中结点的最大层次；

终端结点、非终端结点

双亲、孩子、兄弟、祖先、子孙、堂兄弟、

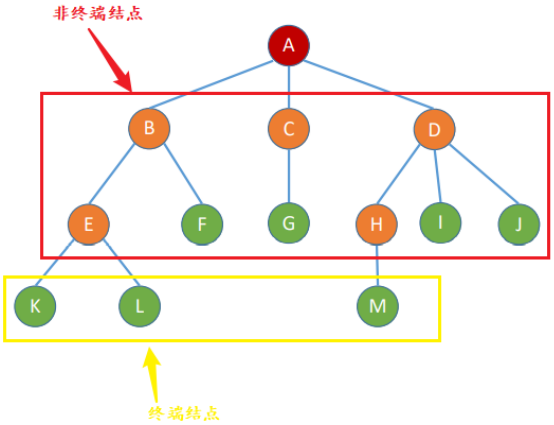
有序树：树中结点的各子树从左至右有次序；

无序树：树中结点的各子树无次序。

路径：两个结点之间的路径，只能从上往下数

路径长度：经过几条边。

森林：森林是m（m≥0）棵互不相交的树的集合

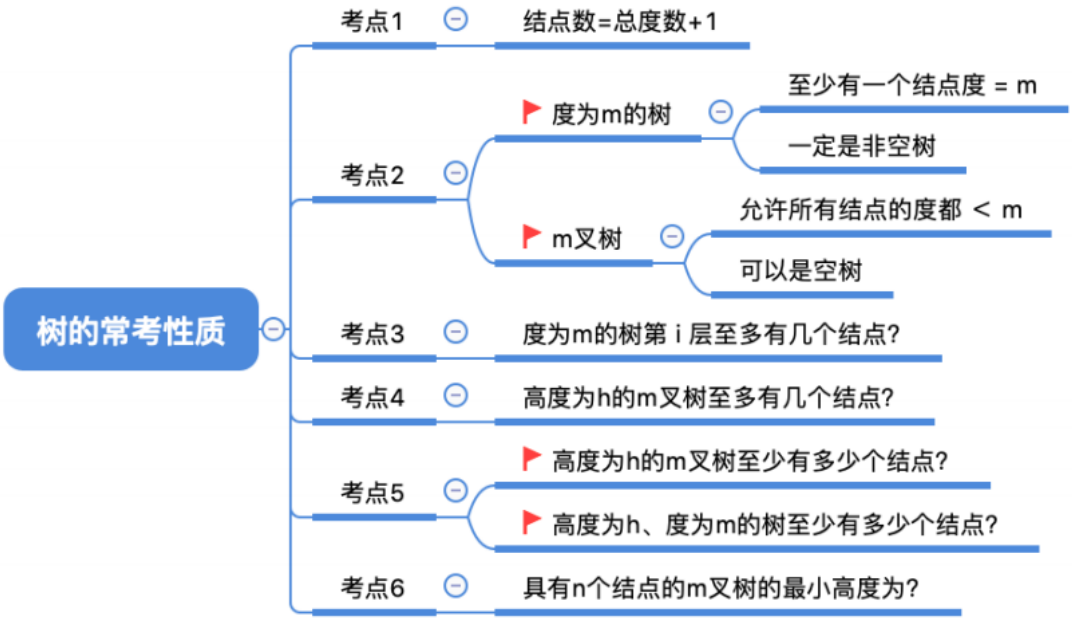



A是根结点
E结点的度是：2
A树的度是：3
B结点的深度：3
A树的深度：4
A到K的路径长度是：3

A是所有结点的祖先
B的双亲是A，A的孩子有BCD
F是E的兄弟结点
A下面的所有结点都是A子孙

## 2.树的性质

**总览：**




1. 结点数=总度数+1。
2. 度为m的树和m叉树的区别。

| 度为m的树 | m叉树 |
| --- | --- |
| 任意结点的度 ≤ m | 任意结点的度 ≤ m |
| 至少有一个结点度 = m（有m个孩子） | 允许所有结点的度都 < m |
| 一定是非空树，至少有m+1个结点 | 可以是空树 |

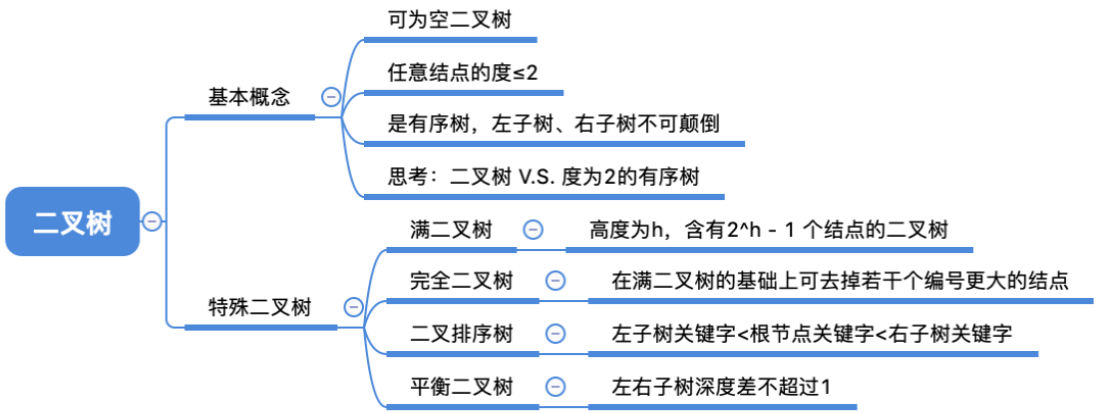
3. 度为 `m` 的树第 `i` 层至多有$m^{i-1}$结点， `m` 叉树第 `i` 层至多有$m^{i-1}$个结点。
4. 高度为 `h` 的 `m` 叉树至多有$\frac{m^h-1}{m-1}$，至少有 `h` 个节点，高度为 `h` 、度为 `m` 的树至少有 `h+m-1` 个结点。
5. 具有 `n` 个结点的 `m` 叉树的最小高度为$\lceil \log_m(n(m-1)+1) \rceil$。

**需要会推导**

# 二、二叉树的基本概念

## 1.二叉树的定义和基本术语

**总览：**



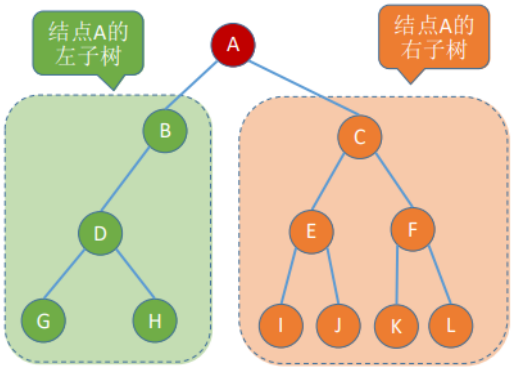

1. 二叉树定义：（二叉树也是是递归定义的数据结构）

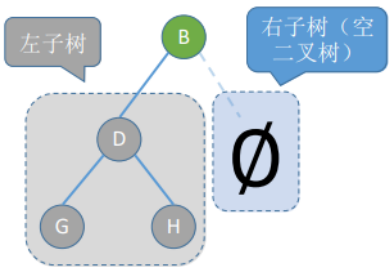
   二叉树是n（n≥0）个结点的有限集合：

   （1）或者为空二叉树，即n = 0。

   （2）或者由一个根结点和两个互不相交的被称为根的左子树和右子树组成。*左子树和右子树又分别是一棵二叉树。*

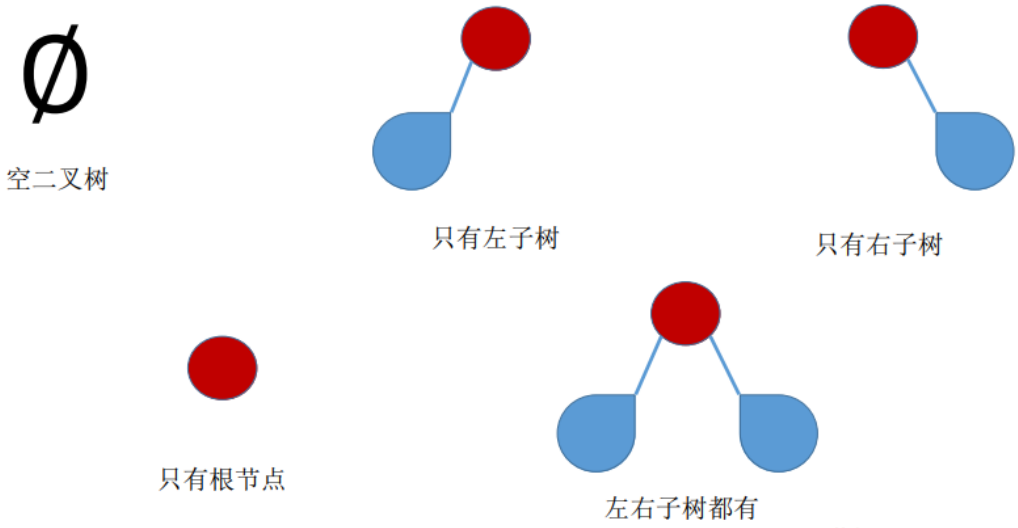
   **特点：①每个结点至多只有两棵子树 ②左右子树不能颠倒（二叉树是有序树）**




🌲二叉树是递归定义的数据结构

2. 二叉树的五种形态



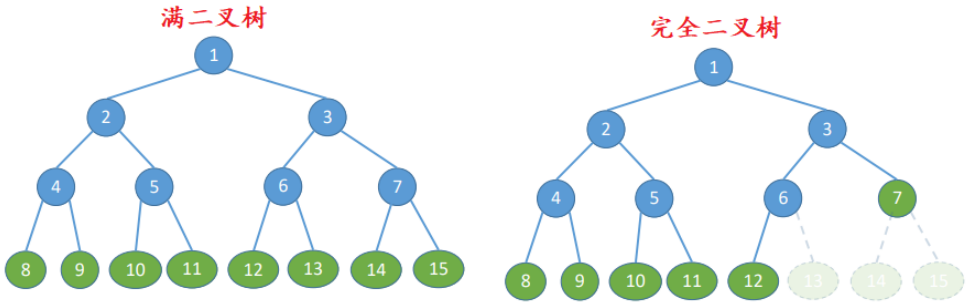

3. 几种特殊的树

   1. 满二叉树：一棵高度为 h，且含有$2^h-1$个结点的二叉树。

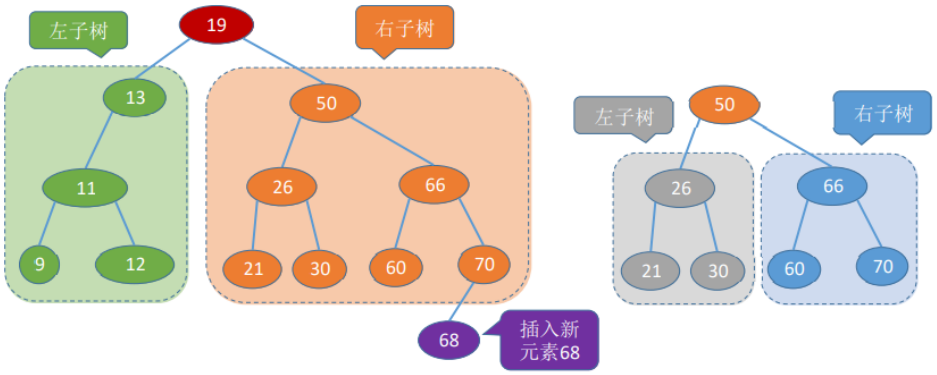
      *特点： ①只有最后一层有叶子结点 ②不存在度为 1 的结点 ③按层序从 1 开始编号，结点 i 的左孩子为 2i，右孩 子为 2i+1；结点 i 的父节点为$\lfloor i/2 \rfloor$ 。*

   2. 完全二叉树：当且仅当其每个结点都与高度为 h 的 满二叉树中编号为 1~n 的结点一一对应时，称为 完全二叉树。

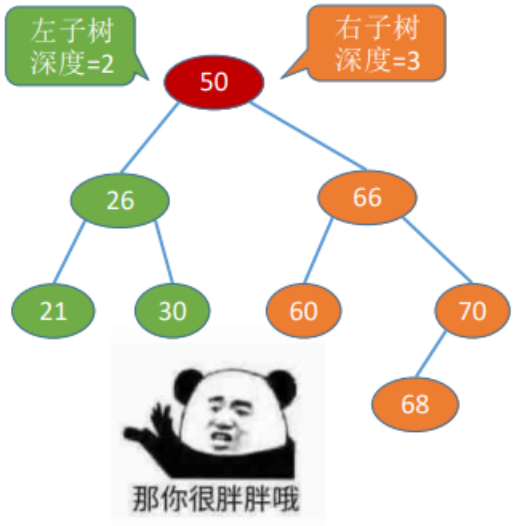
      *特点： ①只有最后两层可能有叶子结点 ②最多只有一个度为1的结点 ③按层序从 1 开始编号，结点 i 的左孩子为 2i，右孩 子为 2i+1；结点 i 的父节点为$\lfloor i/2 \rfloor$。④ i≤$\lfloor n/2 \rfloor$为分支结点， i>$\lfloor n/2 \rfloor$为叶子结点。*




   3. 二叉排序树：左子树上所有结点的关键字均小于根结点的关键字； 右子树上所有结点的关键字均大于根结点的关键字。 左子树和右子树又各是一棵二叉排序树。




   4. 平衡二叉树：树上任一结点的左子树和右子树的深度之差不超过1。



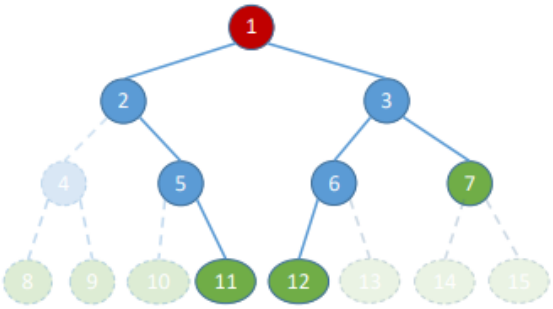

## 2.二叉树的性质

1. 二叉树性质：
    1. 设非空二叉树中度为0、1和2的结点个数分别为$n_0$、$n_1$和$n_2$，则$n_0 = n_2 + 1$。
    2. 二叉树第i层至多有$2^{i-1}$个结点、m叉树第i层至多有$m^{i-1}$个结点。
    3. 高度为h的二叉树至多有$2^h - 1$个结点（满二叉树）、高度为h的m叉树至多有$\frac{m^h-1}{m-1}$个结点。
2. 完全二叉树：
    1. 具有n个（n > 0）结点的完全二叉树的高度h为$\lceil log_2(n+1) \rceil$或$\lceil log_2 n \rceil + 1$。
    2. 完全二叉树最多只有一个度为1的结点，即 n1=0或1，若完全二叉树有2k个（偶数）个结点，则 必有 `n1=1， n0 = k， n2 = k-1` 若完全二叉树有2k-1个（奇数）个结点，则 必有`n1=0， n0 = k， n2 = k-1`

## 3.二叉树的存储结构

1. **顺序存储**

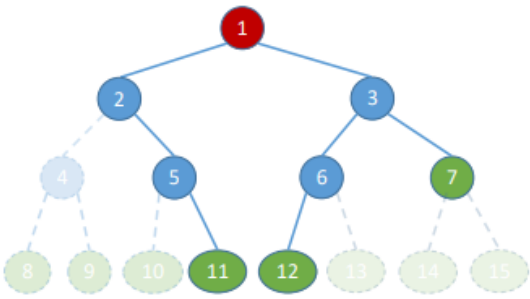
```
#define MaxSize 100
struct TreeNode{
    ElemType value; // 结点中的数据元素
    bool isEmpty;   // 结点是否为空
};
TreeNode t[MaxSize];
```

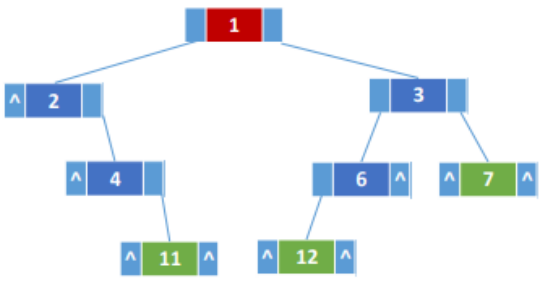




2. **链式存储**

```
// 二叉树的二叉链表
typedef struct BiTNode{
    ElemType data;  // 数据域
    struct BiTNode *lchild,*rchild;// 左右孩子指针
}BiTNode,*BiTree;
// 二叉树的三叉链表
typedef struct BiTNode{
    ElemType data;  // 数据域
    struct BiTNode *lchild,*rchild;// 左右孩子指针
    struct BiTNode *parent; // 双亲结点，方便找到双亲
}BiTNode,*BiTree;
```

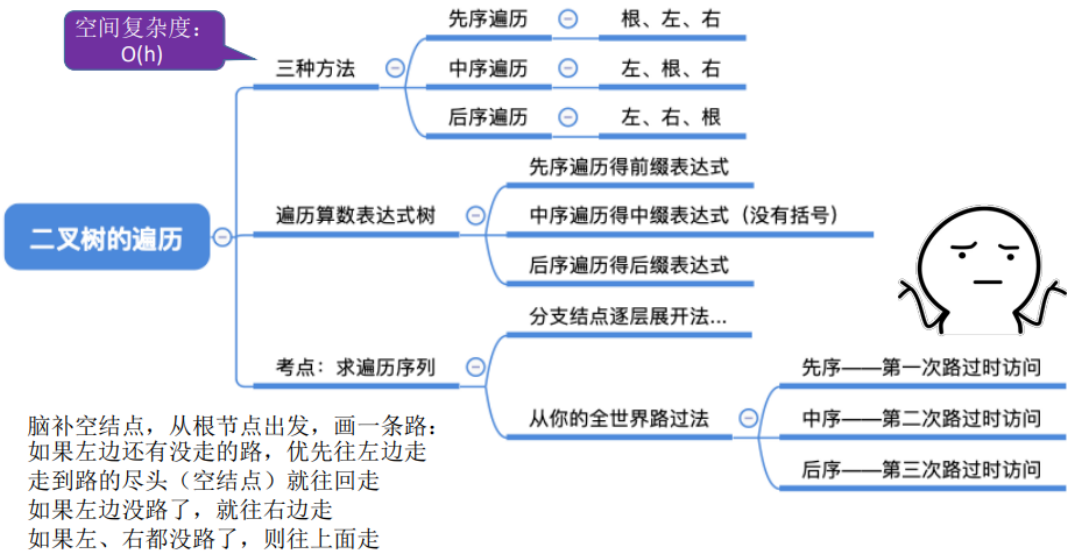


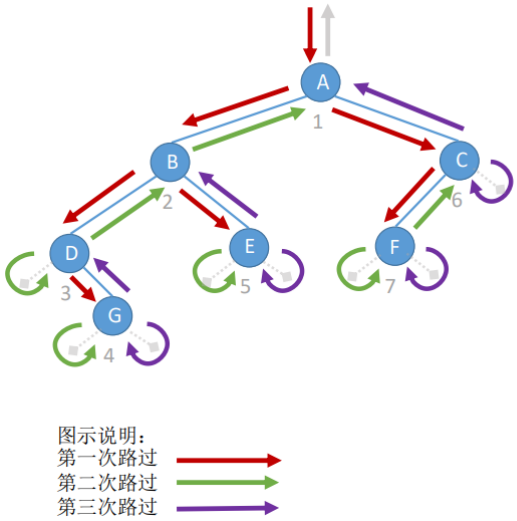

- **n个结点的二叉链表共有n+1个空链域**
- **高度为 h 且只有 h 个结点的单 支树（所有结点只有右孩子），也至少需 要 2h-1 个存储单元**
- **二叉树的顺序存储结构，只适合存 储完全二叉树**

# 三、二叉树的遍历及线索二叉树

**总览：**




# 1.二叉树的遍历




1. 先序遍历：根、左、右（NLR）

```
void PreOrder(BiTree T){
    if(T != NULL){
        visit(T);
        PreOrder(T->lchild);
        PreOrder(T->rchild);
    }
}
```

2. 中序遍历：左、根、右（LNR）

```
void PreOrder(BiTree T){
    if(T != NULL){
        visit(T);
        PreOrder(T->lchild);
        visit(T);
        PreOrder(T->rchild);
    }
}
```

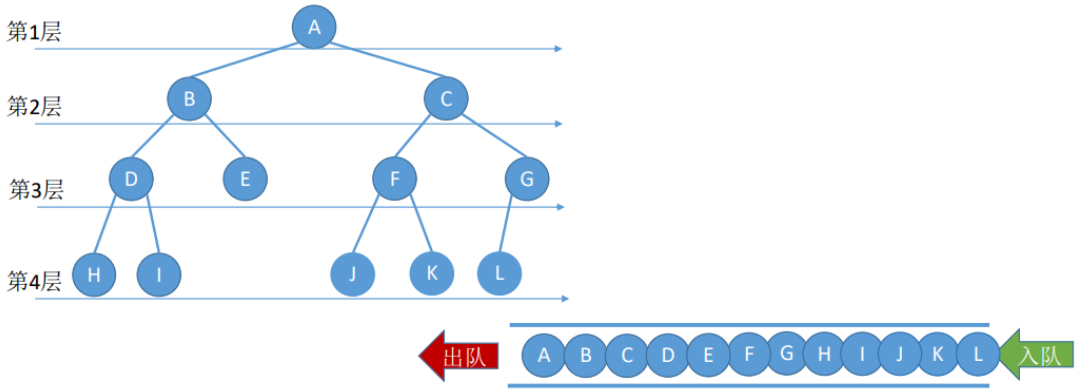
3. 后序遍历：左、右、根（LRN）

```
void PreOrder(BiTree T){
    if(T != NULL){
        visit(T);
        PreOrder(T->lchild);
        PreOrder(T->rchild);
        visit(T);
    }
}
```

4. 应用—求树的深度

```
int treeDepth(BiTree T){
    if(T == NULL)
        return 0;
    else{
        int l = treeDepth(T->lchild);
        int r = treeDepth(T->rchild);
        // 树的深度=Max(左子树深度，右子树深度)+1
        return l>r ? l+1 :r+1;
    }
}
```

## 2.二叉树的层次遍历




算法思想：
①初始化一个辅助队列
②根结点入队
③若队列非空，则队头结点出队，访问该结点，并将其左、右孩子插入队尾（如果有的话）
④重复③直至队列为空

```
void LevelOrder(BiTree T){
    LinkQueue Q;
    InitQueue(Q);
    BiTree p;
    EnQueue(Q,T);   // 将根结点入队
    while(!isEmpty(Q)){
        DeQueue(Q,p);   // 根结点出队
        visit(p);
        if(p->lchild!=NULL){
            EnQueue(Q,p->lchild);
```

```
        }
        if(p->rchild!=NULL){
            EnQueue(Q,p->rchild);
        }
    }
}
```

## 3.由遍历构造二叉树

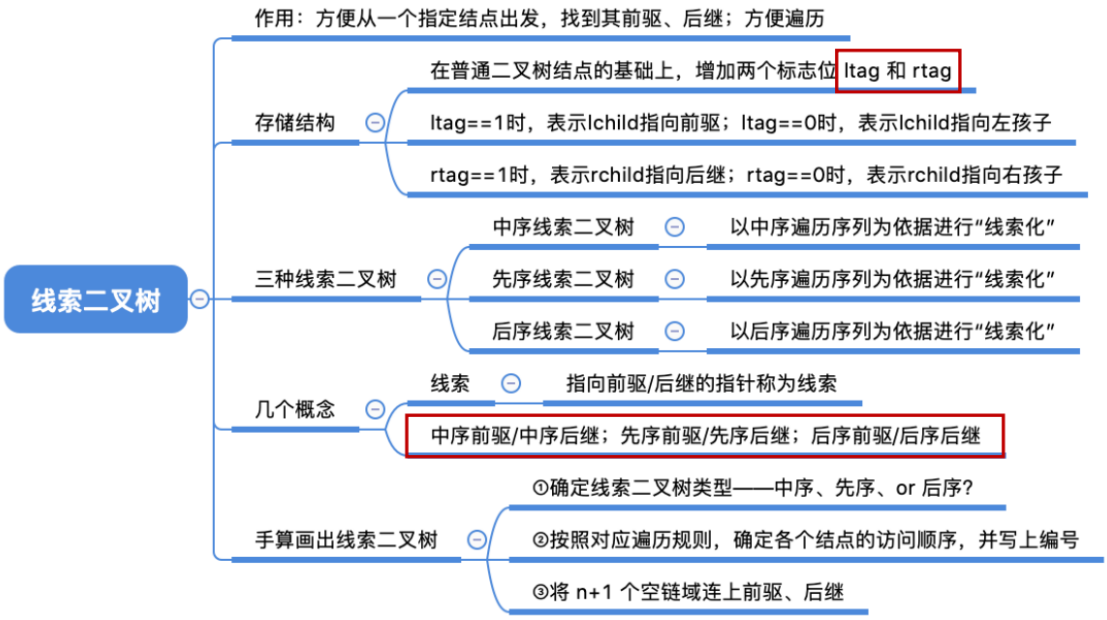
**结论：若只给出一棵二叉树的 前/中/后/层 序遍历序列中的一种，不能唯一确定一棵二叉树**


前(后、层)序遍历定根，中序遍历定左右

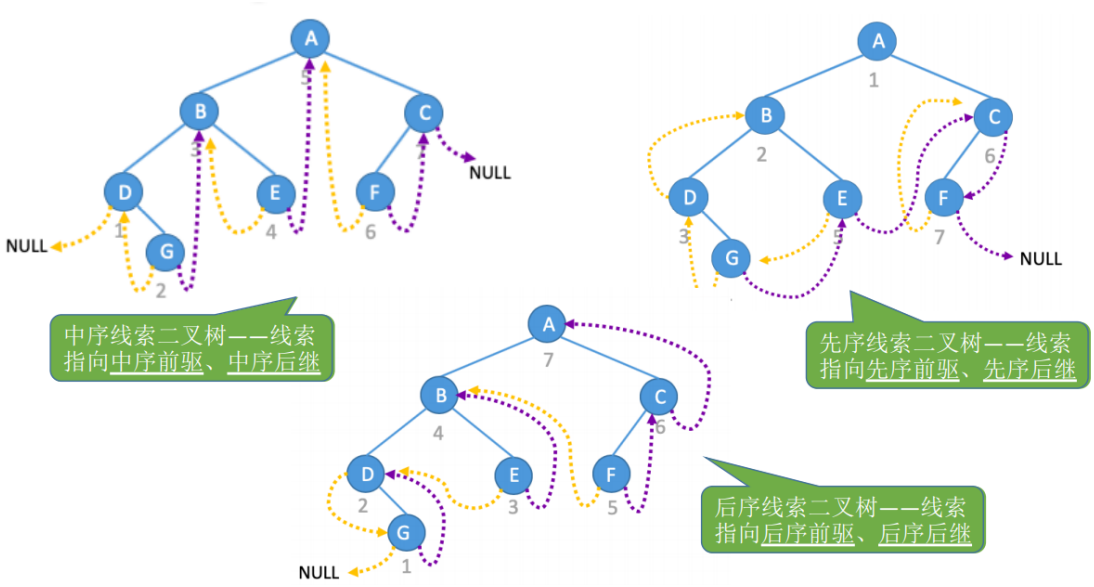
## 4.线索二叉树

**是一种物理结构**

**总览：**




**链式存储结构：当不存在左右孩子时，左指前趋结点，右指后继。前趋和后继由遍历顺序决定。**



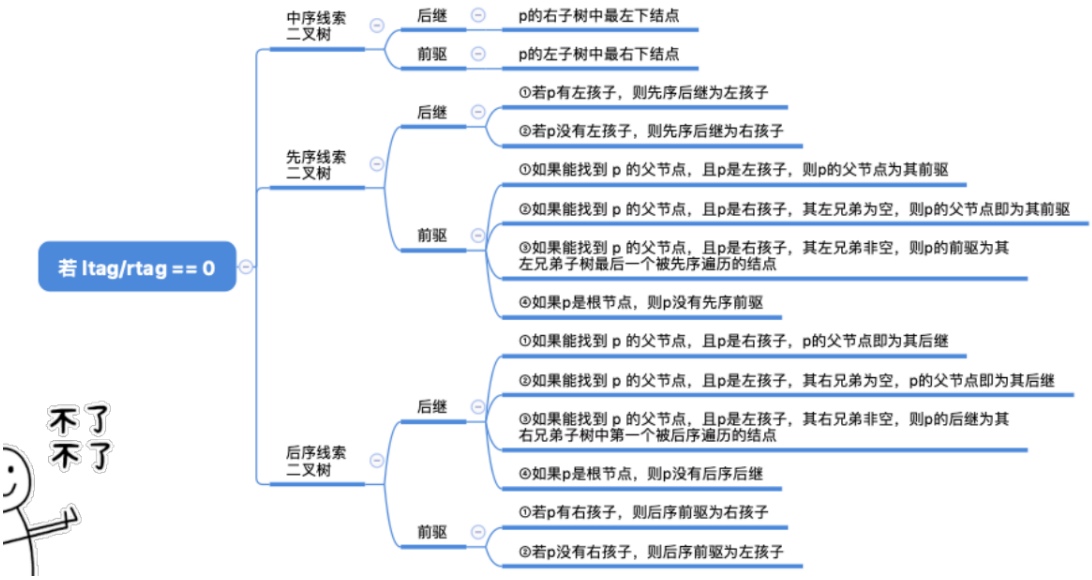

```c
// 线索二叉树结点
typedef struct ThreadNode{
    ElemType data;  // 数据域
    struct ThreadNode *lchild,*rchild;// 左右孩子指针
    int ltag,rtag;//tag==0，表示指针指向孩子,tag==1，表示指针是“线索”
}ThreadNode,*ThreadTree;
```


### 中序线索化

```c
ThreadNode *pre=NULL;
void InTread(ThreadNode T){
    if(T != NULL){
        visit(T);
        InTread(T->lchild);
        visit(T);
        InTread(T->rchild);
    }
}
void visit(ThreadNode *q){
    if(q->lchild == NULL){
        q->lchild = pre;
        q->ltag = 1;
    }
    if(pre!=NULL&&pre->rchild==NULL){
        pre->rchild = q;
        pre->rtag = 1;
    }
    pre = q;
}
```

## 5.线索二叉树的前趋后继

**总览：**



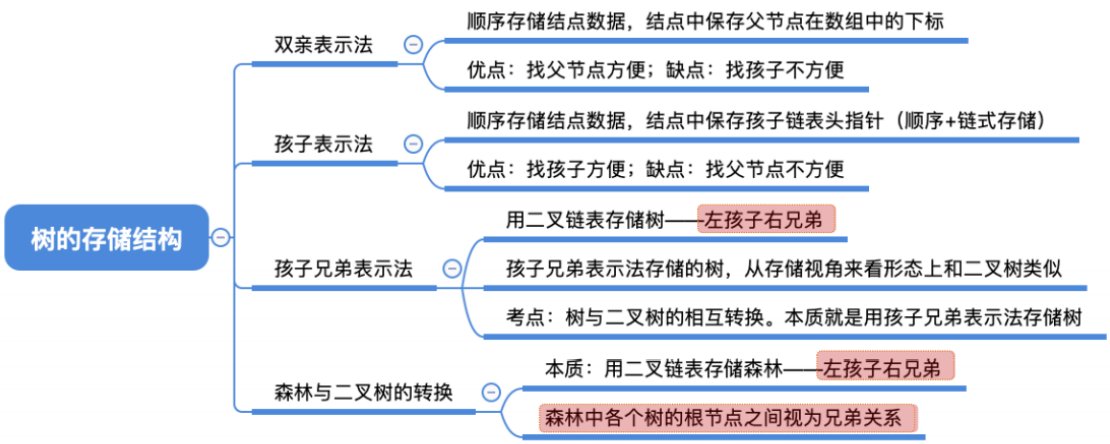

# 四、树和森林的存储及遍历

*树和二叉树的转换：左孩子右兄弟。*

*森林和二叉树的转换：将各根节点视作兄弟关系。*

## 1.树和森林的存储

**总览：**




树的定义：

树是n（n≥0）个结点的有限集合，n = 0时，称为空树，这是一种特殊情况。在任意一棵非空树中应满足：

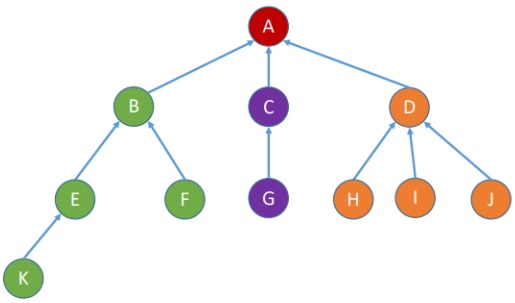
1）有且仅有一个特定的称为根的结点。

2）当n > 1时，其余结点可分为m（m > 0）个互不相交的有限集合T1, T2,..., Tm，其中每个集 合本身又是一棵树，并且称为根结点的子树。

**存储：**

1. 双亲表示法（顺序存储）：每个结点中保存指向双亲的“指针”。

```
#define MAX_TREE_SIZE 100   // 树中最多结点数
typedef struct{
    ElemType data;
    int parent;
}PTNode;
typedef struct{
    PTNode nodes[MAX_TREE_SIZE];
    int n;
}PTree;
```

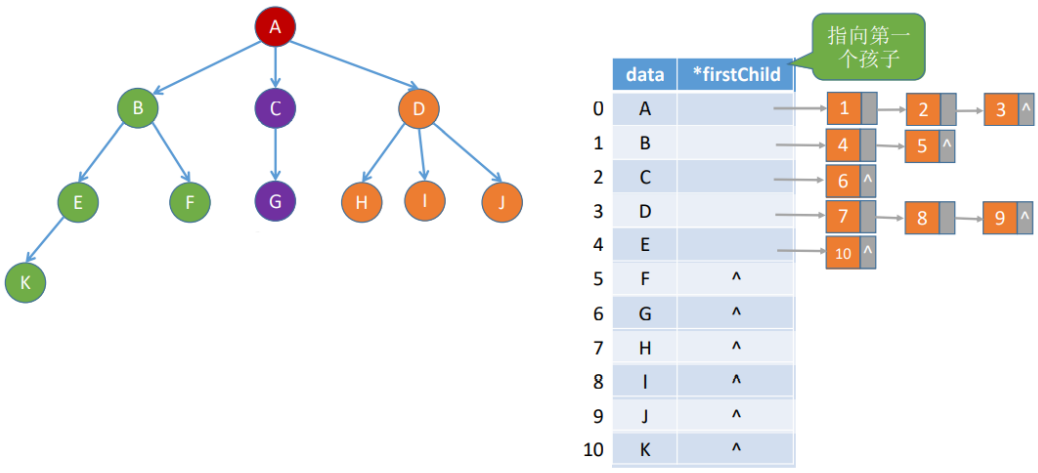



| | data | parent |
|---|---|---|
| 0 | A | -1 |
| 1 | B | 0 |
| 2 | C | 0 |
| 3 | D | 0 |
| 4 | E | 1 |
| 5 | F | 1 |
| 6 | G | 2 |
| 7 | H | 3 |
| 8 | I | 3 |
| 9 | J | 3 |
| 10 | K | 4 |
| 11 | | |
| 12 | | |
| 13 | | |

根节点固定存储在0，-1表示没有双亲

2. 孩子表示法：顺序存储各个节点，每个结点中保存孩子链表头指针。

```
struct CTNode{
    int child;      // 孩子结点在数组中的位置
    struct CTNode *next;    // 下一个孩子
}
typedef struct{
    ElemType data;
    struct CTNode *firstChild;// 第一个孩子
}CTBox;
typedef struct{
    CTBox nodes[MAX_TREE_SIZE];
    int n,r;    // 结点数和根的位置
}CTree;
```




3. 孩子兄弟表示法：左孩子右兄弟

```
typedef struct CSNode{
    ElemType data;  // 数据域
    struct CSNode *firstchild,*nextsibling; // 左孩子右兄弟
}
```

## 2.树和森林的遍历

**总览：**

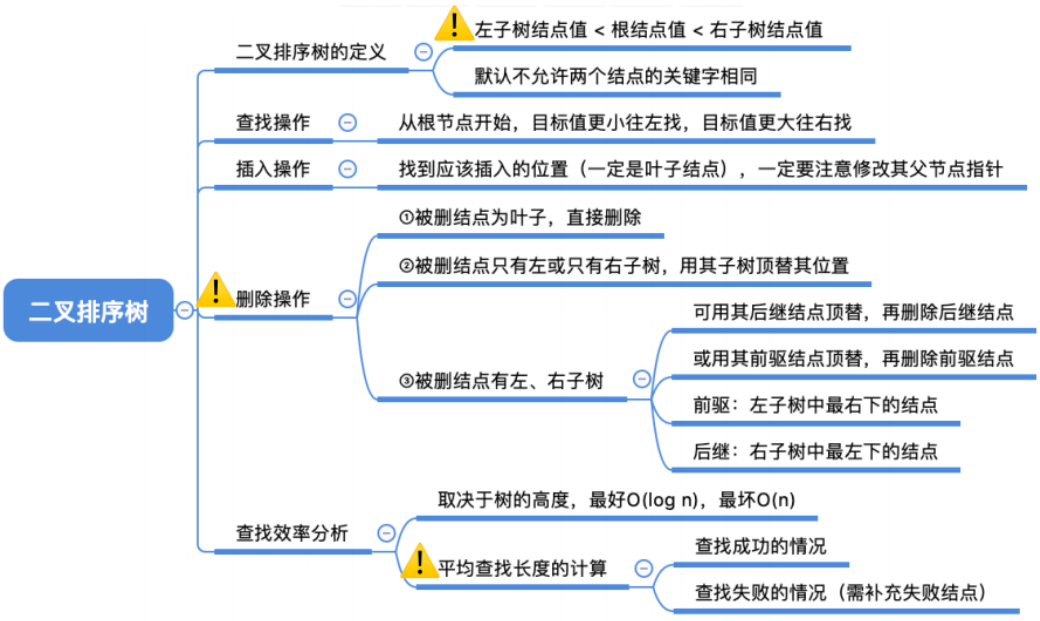

| 树 | 森林 | 二叉树 |
| --- | --- | --- |
| 先根遍历 | 先根遍历 | 先根遍历 |
| 后根遍历 | 中序遍历 | 中序遍历 |

```
// 先序遍历
void PreOrder(TreeNode *R){
    if(R!=NULL){
        visit(R);
        while(R还有下一个子树T)
            PreOrder(T);
    }
}
// 后序遍历
void PreOrder(TreeNode *R){
    if(R!=NULL){
```

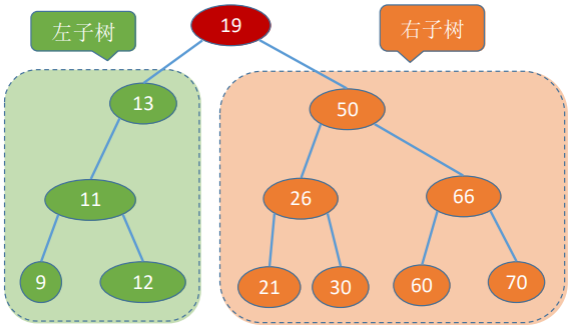
```
12          while(R还有下一个子树T)
13              PreOrder(T);
14          visit(R);
15      }
16  }
```

# 五、其它树

## 1.二叉排序树（BST）

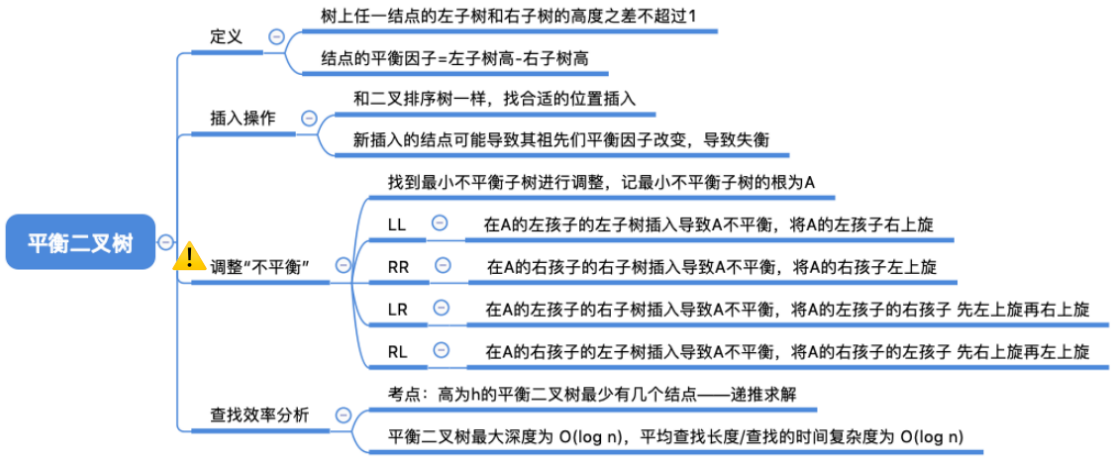



1. 定义：二叉排序树，又称二叉查找树（BST，Binary Search Tree） 一棵二叉树或者是空二叉树，或者是具有如下性质的二叉：左子树结点值 < 根结点值 < 右子树结点值、进行中序遍历，可以得到一个递增的有序序列



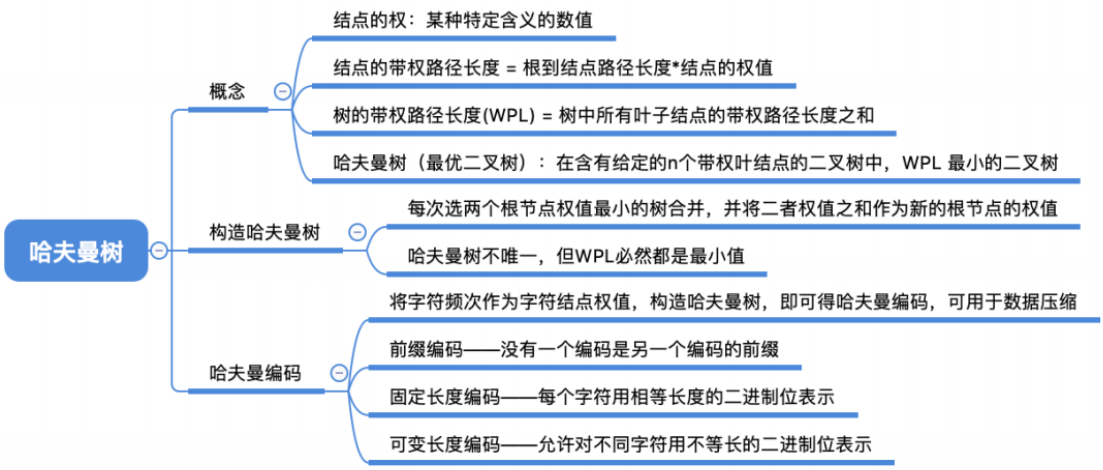
## 2.平衡二叉树

**总览：**




## 3.哈夫曼树（最优二叉树）

**总览：**




1. 带全路径长度

- 结点的权：有某种现实含义的数值（如：表示结点的重要性等）
- 结点的带权路径长度：从树的根到该结点的路径长度（经过的边数）与该结点上权值的乘积
- 树的带权路径长度：树中所有叶结点的带权路径长度之和（`WPL`）

**哈夫曼树的定义：** 在含有n个带权叶结点的二叉树中，其中 *带权路径长度（WPL）最小* 的二叉树称为哈夫曼树，也称最优二叉树。

2. **哈夫曼树的构造：**

   1）将这n个结点分别作为n棵仅含一个结点的二叉树，构成森林F。

   2）构造一个新结点，从F中选取两棵根结点权值最小的树作为新结点的左、右子树，并且将新 结点的权值置为左、右子树上根结点的权值之和。

   3）从F中删除刚才选出的两棵树，同时将新得到的树加入F中。

   4）重复步骤2）和3），直至F中只剩下一棵树为止。

   重要结论：

- 1）每个初始结点最终都成为叶结点，且权值越小的结点到根结点的路径长度越大
- 2）哈夫曼树的结点总数为2n − 1
- 3）哈夫曼树中不存在度为1的结点。
- 4）哈夫曼树并不唯一，但WPL必然相同且为最优

3. **哈夫曼编码**