**1.基本概念**

树的定义：一个有n个结点的有限集（n≥0）

空树：n=0

非空树：T={r, T1, T2, …, Tm}

r：树的根结点 m：分支数 Ti：子树

每个子树的根节点有且仅有一个直接前驱，有0~多个直接后继

树的逻辑表示法：树形表示法、广义表表示法、Venn图、目录结构表示法

**树的相关术语**

孩子、双亲、兄弟

堂兄弟：双亲在同一层的结点/双亲是兄弟的结点

结点的祖先：从根结点到该结点所经分支上的所有结点

结点的子孙：该节点的孩子以及这些孩子的子孙

结点的度：结点所拥有的**子树数量**

**树的度（叉数）**：树中各个**结点的度的最大值**，通常将度为m的树称为m叉树/m次树

**树的宽度**：统计树中**每一层的结点数量**，取**最大**的数量作为树的宽度

**结点的层次/深度**：根节点在第1层，其孩子的深度＋1

**结点的高度：叶结点**的高度为1；非叶结点的高度等于它的**孩子结点高度中的最大值加1**

**（其孩子中最深的叶节点到该节点的路径长度加1）**

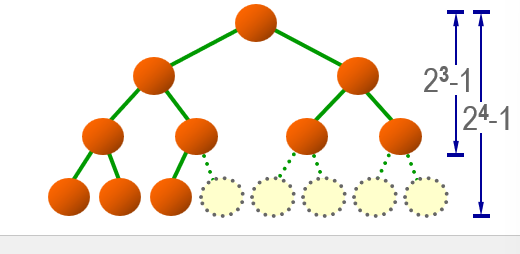
**树的层次/深度/高度**：树中(叶子)结点的最大层次 根节点深度为1

有序树（树中结点的各个子树是有次序的，不能互换）VS 无序树

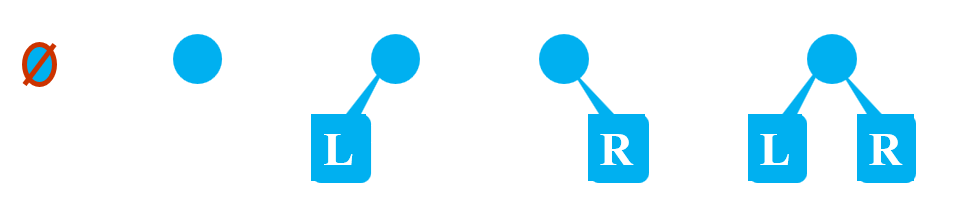
严格二叉树：除叶节点外，所有结点度数都为2

**满m叉树/满m次树**：除根结点和叶子结点外，其余结点度数均为m；所有叶子节点要在同一层

**完全m叉树/完全m次树**：

倒数第二层之前的所有结点都是满m叉的，最后一层缺少的结点必须是按层序编号后序号最大的若干点（即编号和完全m叉树要保持一致）

森林：森林是m棵树的集合(m≥0)

****

**2.二叉树（是有序树）**

**定义：**每个结点至多只有两棵子树，且二叉树的子树**有左右之分**，其次序不能任意颠倒

**性质1**：若二叉树结点的层次从 1 开始, 则在**二叉树的第 i 层(i≥1)最多有 个结点**

**性质2**：深度为 k( k≥1 )的二叉树**最少有 k 个结点，最多有 -1个结点**

**性质3**：对任何一棵二叉树，如果其叶结点有个，度为 2 的非叶结点有个，则有： **= ＋ 1**

性质：**二叉树的分支数e等于二叉树中所有结点的度的总和 e=n1+2·n2...+m·nm**

**二叉树的结点数等于所有结点的度的总和+1 n=e+1**

**n = n0+n1+n2...+nm**

**满二叉树**

正则的：只有度数为0和为2的结点

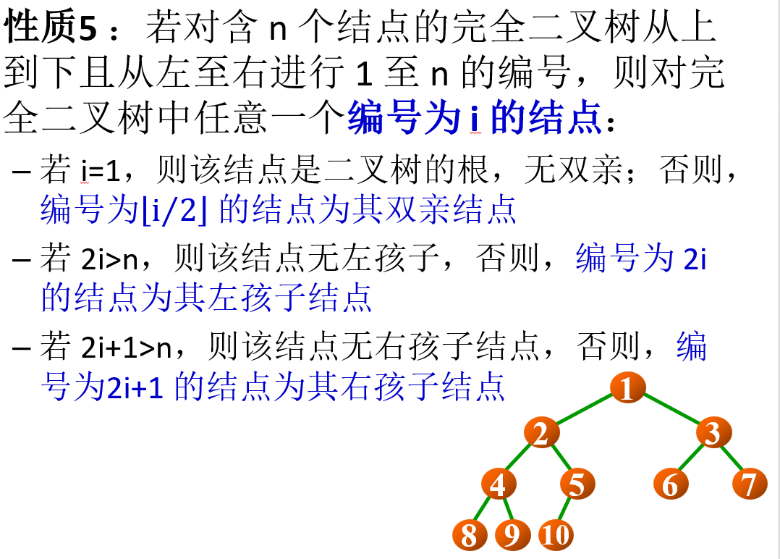
三角形结构：所有叶子节点要在同一层

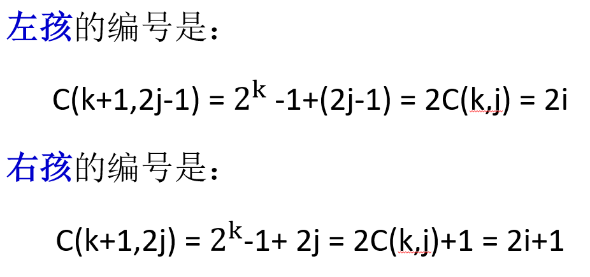
深度k，则结点数为2k-1

结点编号：第k层第j个结点编号**C(k, j)=2k-1 - 1 + j**

**完全二叉树**

**性质4**：具有 n (n≥0) 个结点的**完全二叉树的深度为 + 1）**

对于第k层第j个结点



**二叉树的存储结构**

顺序存储：数组

链式存储：二叉链表、三叉链表、双亲链表

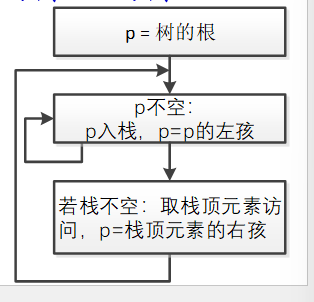
|  |  |  |  |
| --- | --- | --- | --- |
| 存储结构 | 结点成员 | | |
| 二叉链表 | data | \*lchild、\*rchild |  |
| 三叉链表 | data | \*lchild、\*rchild | \*parent |
| 双亲链表 | data | \*parent | LRTag：表示是父母的左子树还是右子树 |

**3.二叉树的遍历**

设访问根结点记作 D，遍历根的左子树记作 L，遍历根的右子树记作 R

|  |  |  |
| --- | --- | --- |
| 遍历方式 | 访问顺序 | 实现方法 |
| 先序遍历 | DLR | 访问根节点-递归遍历左子树-递归遍历右子树 |
| 中序遍历 | LDR |  |
| 后序遍历 | LRD |  |

遍历算法的非递归实现（中序遍历）

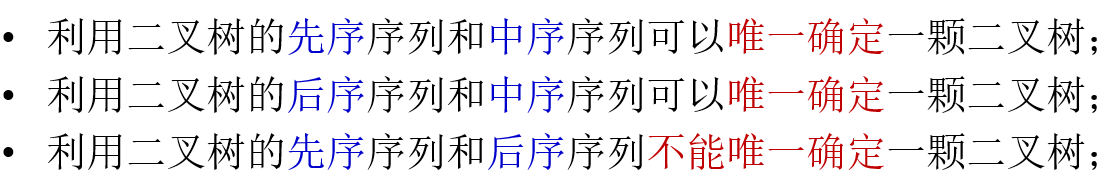


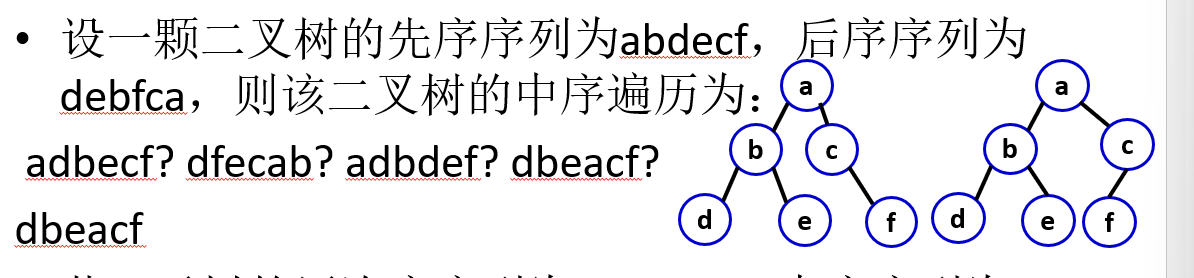
**层次遍历二叉树！！！**

**给定完全二叉树的中序遍历序列可以唯一确定一颗完全二叉树：**

1.节点个数确定完全二叉树的层数（除去最深层为满二叉树）

2.按确定好的树形态可以标出每个结点的编号





**4.Huffman树（使用三叉链表表示）**

**（两个）结点的路径长度**：从一个结点到另一个结点的路径上分支的数目

**树的路径长度**：从树**根**到**每个结点**的**路径长度之和**

**结点的带权路径长度**：从**根结点**到该结点的路径长度与**该结点的权**的乘积

**树的带权路径长度**：树中所有**叶子结点**的带权路径长度之和

**正则二叉树**(Regular 2-way Tree)/Huffman Tree：二叉树的结点的度或为0或为2(任意非叶子结点都有2个儿子)，**没有度为1的结点**

**Huffman树/最优二叉树**：一颗有 n个叶子结点的 二叉树，其叶子结点的权重分别是{, , …., } ，并且其**带权路径长度(WPL值)达到最小**（权值越大的结点离根越远）

**性质**： Huffman树中没有度为1的结点，是**正则/严格二叉树**

N个叶节点的Huffman树共有**2n-1个结点**

构造Huffman树

1.先构造一个 n 棵二叉树的集合 **F** = {, , …., }

2.选取两棵根结点的权值最小的二叉树，做为左、右子树构造一棵新的二叉树

新树的根结点权重为两个子树权重之和

3.在F中删去这两颗树，把新树加入到F中。重复2.，直到F中只剩一棵树

**（F数组只存储根节点，以根节点代表一颗子树）**

**实现方式：三叉静态链表**

**结点成员：**权重weight、lchild、rchild、parent

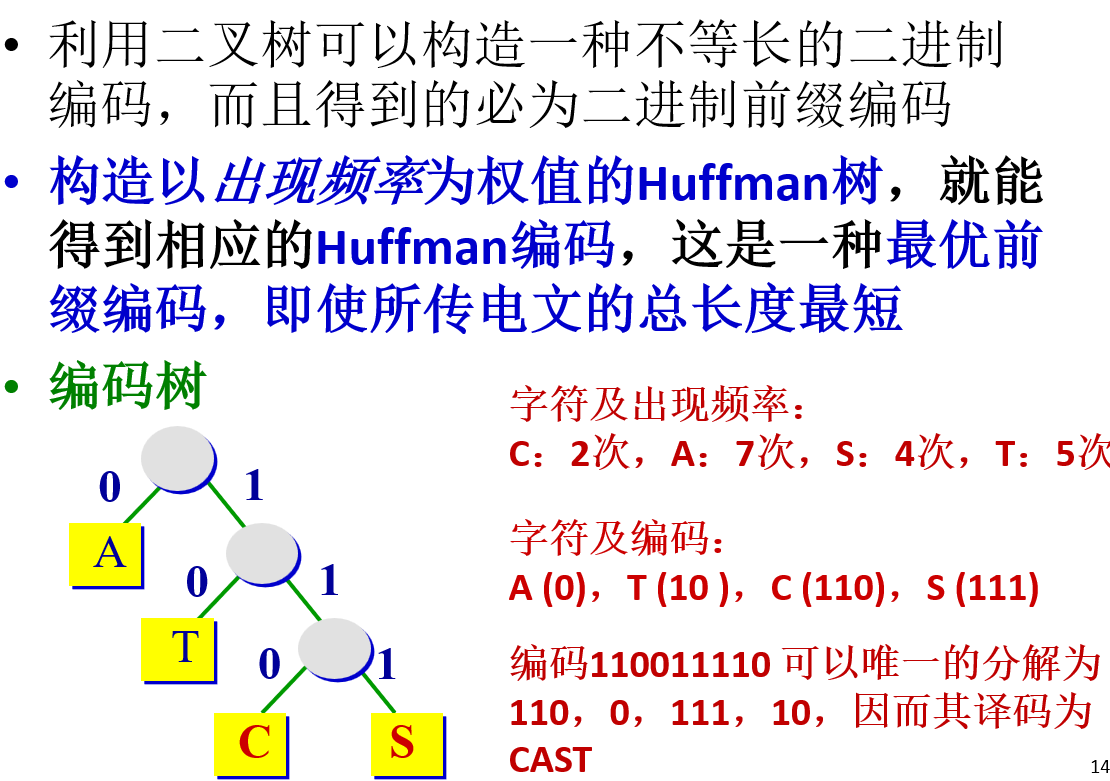
**（一）最优判定树**

**判定树**是一棵二叉树，叶子结点是比较结果，内结点是比较过程，叶子结点所带权值是概率

**最优判定树(Optimal decision tree)**：利用Huffman树，可以在构造判定树(决策树)时让**平均判定(比较)次数达到最小**

**（二）Huffman编码**

**前缀编码**：任何一个字符的编码都不是同一字符集中另一个字符的编码的前缀（**无歧义**）

****

构造k叉Huffman树：每次选k个权重最小的元素来合成一个**新的元素**，该元素权重为k个元素权重之和

**5.线索二叉树**

一棵二叉树：叶节点（度数0）n0个，度数2的结点n2个，则**n0= n2+1**

n个结点的二叉链表必然存在**2n-(n-1) = n+1个空链域 ，n-1是除了根节点其他结点都有一个parent指向自己**

**目的：为了方便地找到二叉树指定结点在某种线性序列中的直接前驱和直接后继**

**二叉树的线索化**：将某种遍历顺序下的前驱、后继关系(线索)记在树的存储结构中

**线索二叉树的建立**

结点成员: data、\*lchild、\*rchild、ltag、rtag

tag=0时指向孩子； **tag=1时左指向前驱 右指向后继**

**（一）中序线索化过程**

**1.建立全局变量**

**pre**：始终指向当前结点的前驱结点，初始为头节点地址BiThrTree

**2. 处理头结点**

**头结点**：右指针回指自身。**T不空时，其左指针指向根节点**；**T空时，左指针回指自身**

BiThrTree head = (BiThrTree)malloc(sizeof(BiThrNode)); *// 创建头结点*

head->ltag = 0; head->rtag = 1; head->rchild = head;

**若T空，左指针回指自身，结束。**

**若T不空，其左指针指向根节点，进行以下步骤：**

**3.遍历二叉树并线索化：InOrderThr(BiThrTree T)**

**InOrderThr(BiThrTree P) P指向当前结点**

(1)递归处理左子树：InThreading(p->lchild);

(2)处理当前节点：

设置前驱线索：若当前结点p的左孩子为空（p->lchild == NULL），则将 p->lchild 指 向 pre，并置 ltag=1

设置后继线索​​：若前驱节点 pre 的右孩子为空（pre->rchild==NULL），则 将 pre->rchild 指向当前节点 p，并置 rtag=1

(3)（当前根结点被访问时）更新前驱节点​​：令 pre = p，为下一个节点做准备

(4)递归处理右子树：InThreading(p->rchild);

**4. 处理首尾连接​**

线索化完成后，将最后一个节点的右线索指向头结点

pre->rchild = head; *// 最后一个节点指向头结点*

pre->rtag = 1; head->rchild = pre; *// 头结点指向最后一个节点*

**（二）先序线索化过程**

类似，但1.中访问顺序改为根 → 左 → 右，

注意左子树为空时才更新pre

同时，处理左右子树时需要检验tag是否为0，避免循环

if (p->ltag == 0) PreThreading(p->lchild); *// 避免循环*

if (p->rtag == 0) PreThreading(p->rchild);

**（三）后序线索化过程**

访问顺序：左 → 右 → 根。

注意右子树为空时才更新pre

最后处理根节点，需检查根结点的右线索，若空，需设置成指向头节点。 ??????不需要吧

**6.树**

**6.1树的存储表示**

**双亲表示法**

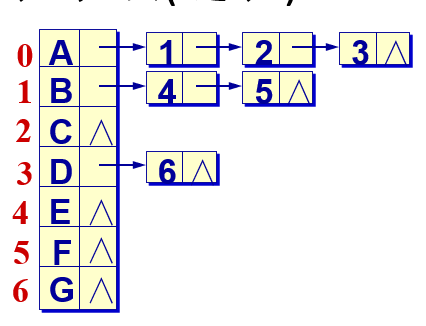
仅有一个parent指针域指向双亲（可以是指针或数组下标）

**孩子表示法**

结点用数组存储，一个结点的孩子用链表存储，

结点成员：数据data、firstchild该节点第一个孩子

孩子结点成员：int index该孩子节点在数组中的下标、next指向下一个孩子



**孩子-兄弟表示法 （左支孩子，右支兄弟）**

左指针指向第一个孩子，右指针指向同层的下一个兄弟

可将任意一棵树转化为二叉树

**树的性质**

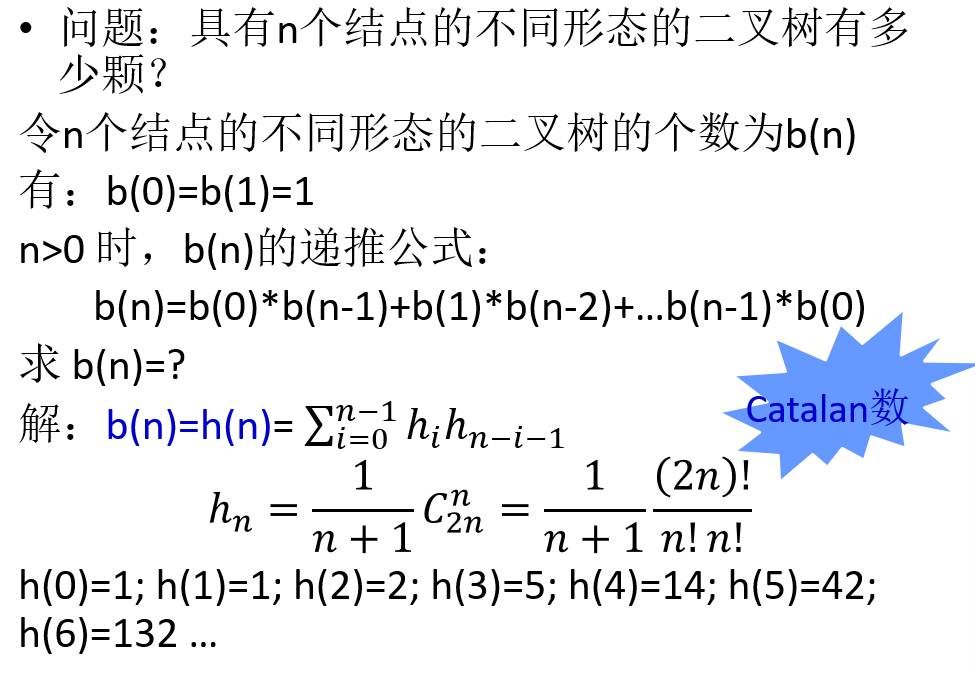
结点数 = 所有结点度数 + 1

度为m的树，其第i层上**至多** (i≥1)个结点

**高度**为**h**的m次树**至多**有( **个结点**

具有**n个结点**的m次树的**最小高度h**为

**二叉树的计数**



**6.2树的遍历**

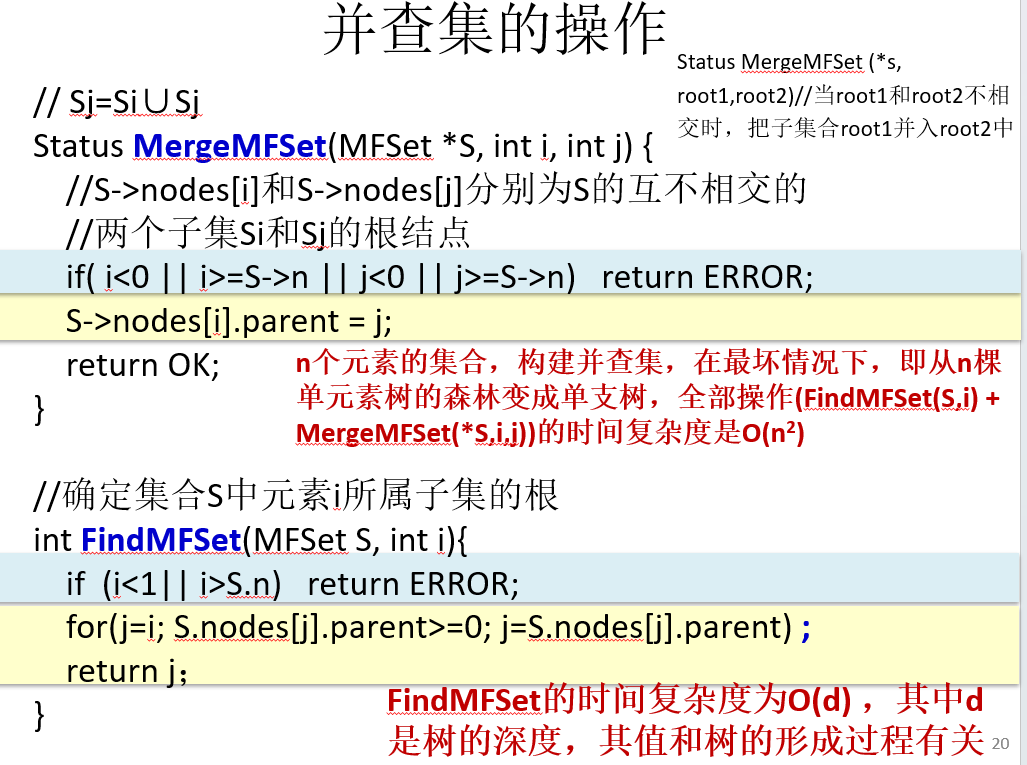
**先根遍历**：若树不空，则先访问根节点，然后依次先根遍历各棵子树

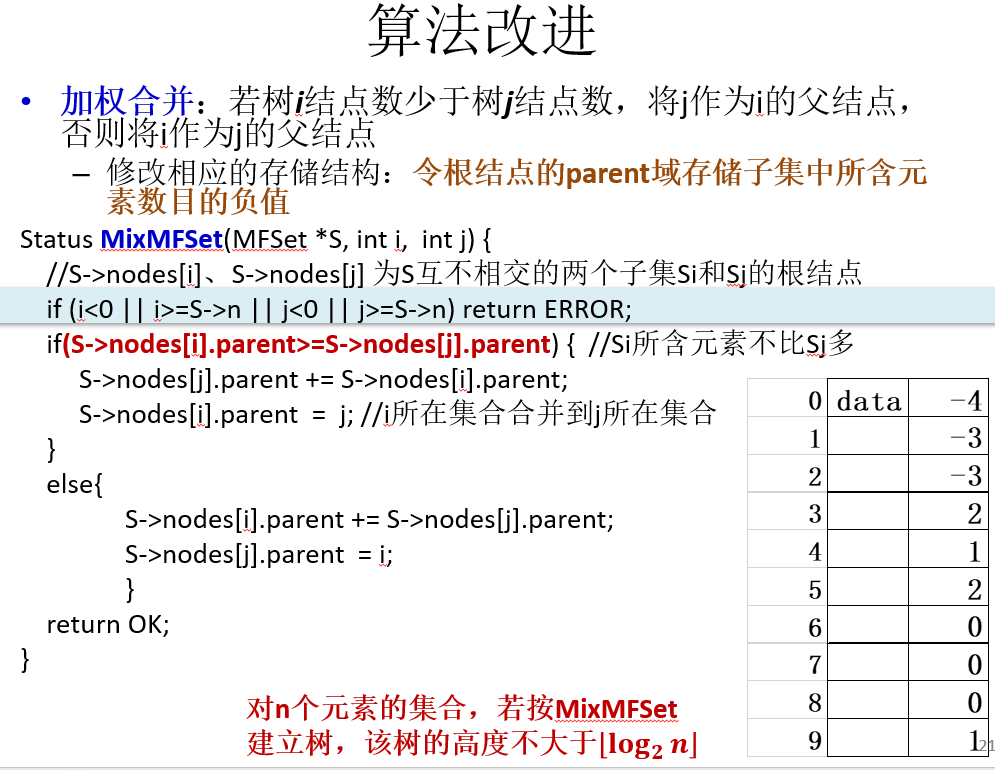
**后根遍历**：若树不空，则先依次后根遍历各棵子树，然后访问根节点

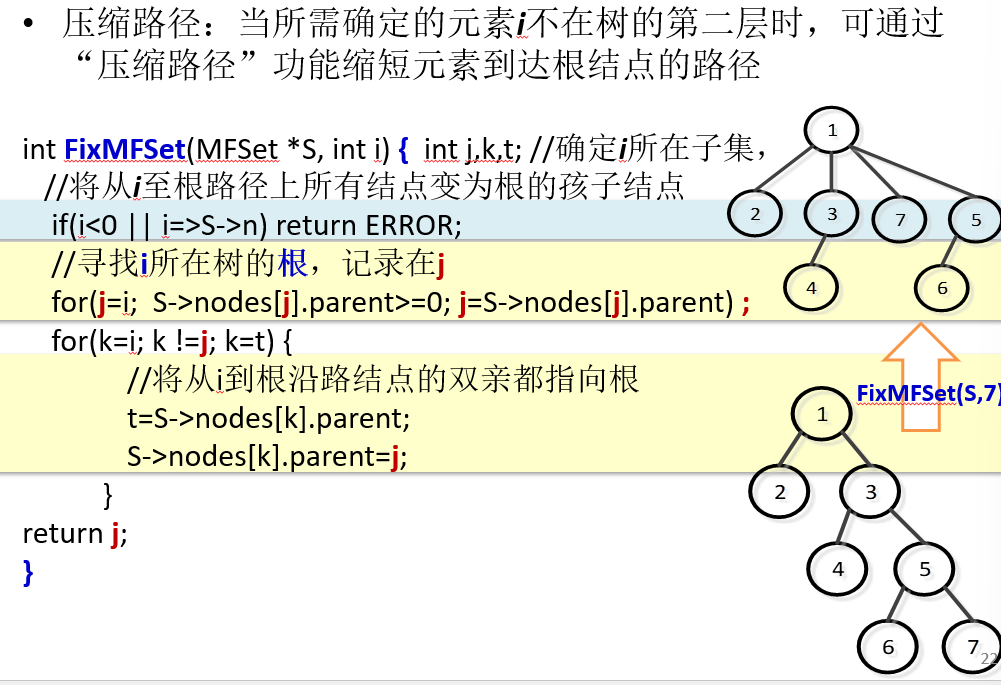
**广度优先遍历/层次遍历**：

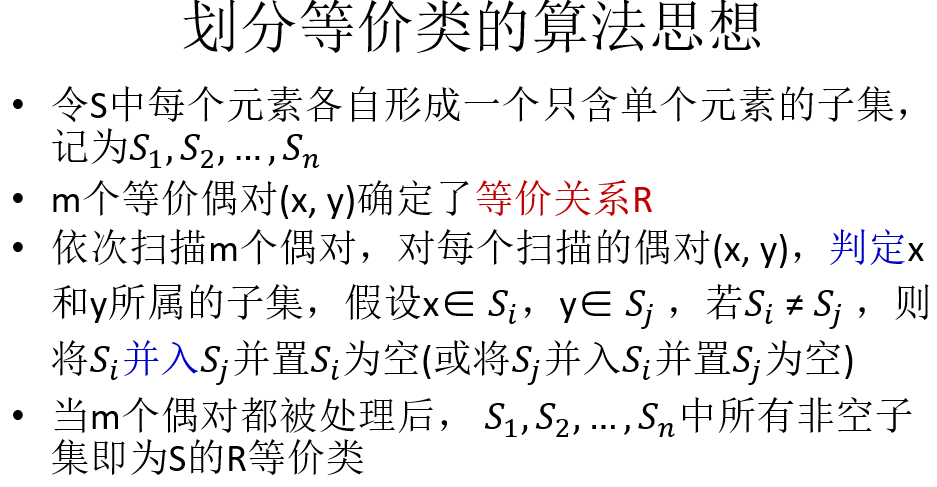
**6.3并查集及其应用**

**并查集(union-find set, disjoint set)**：一种集合，每个元素只属于一个子集合，不同子集合中的元素不相交



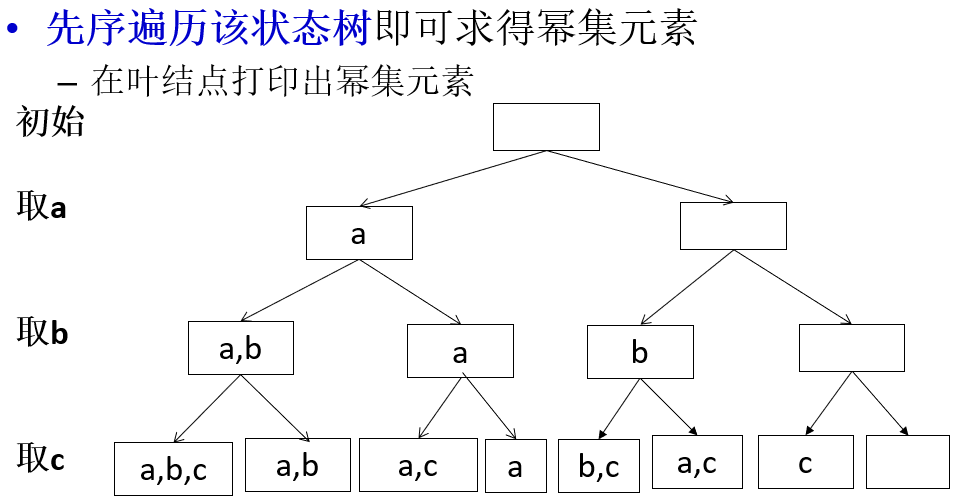




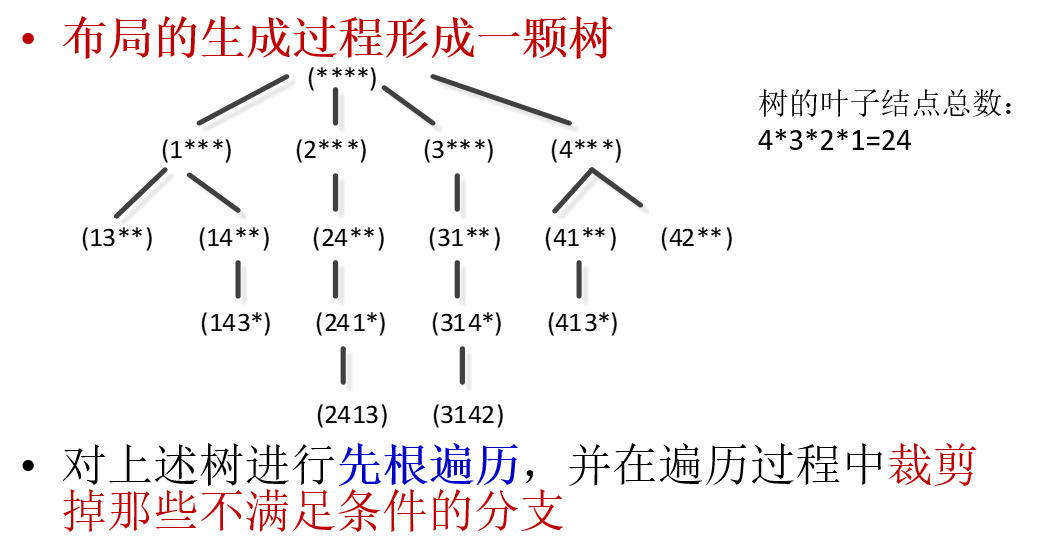


**6.4 n个元素的幂集**

**可以用一棵二叉树表示幂集生成的过程**



**6.5 四皇后问题**



**7.森林**

