

## 5.1树的基本概念

### 树的定义

树是  $n$  ( $n \geq 0$ ) 个节点的有限集。当  $n=0$  时,称为空树

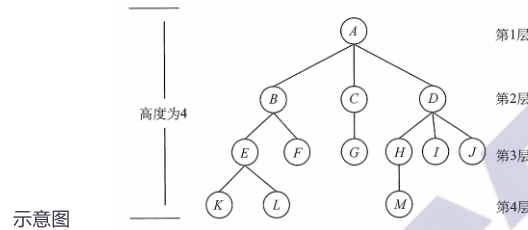
在任意一棵非空树中应满足 有且仅有一个特定的称为根的结点

当  $n > 1$  时,其余节点可分为  $m$  ( $m > 0$ ) 个互不相交的有限集  $T_1, T_2, \dots, T_m$ , 其中每个集合本身又是一棵树,并且称为根的子树

树是一种逻辑结构,也是一种分层结构

树的根结点没有前驱,除根结点外的所有结点有且只有一个前驱

树中所有结点可以有零个或多个后继



示意图

### 基本术语

考虑结点K K的祖先: 根A到结点K的唯一路径上的任意结点

子孙: 结点B是结点K的祖先,而结点K是结点B的子孙

双亲与孩子: 路径上最接近结点K的结点E称为K的双亲, 而K为结点E的孩子

兄弟: 有相同双亲的结点称为兄弟,如结点K和结点L有相同的双亲E,即K和L为兄弟

结点的度: 树中一个结点的孩子个数, 树中结点的最大度数称为树的度

分支结点: 度大于0的结点

叶子结点 (又称终端结点): 度为0 (没有子女结点) 的结点

结点的层次: 从树根开始定义, 根结点为第1层, 它的子结点为第2层, 以此类推

结点的深度: 从根结点开始自顶向下逐层累加的

结点的高度: 从叶结点开始自底向上逐层累加的

树的高度 (或深度): 树中结点的最大层数

有序树和无序树: 树中结点的各子树从左到右是有次序的, 不能互换, 称该树为有序树, 否则称为无序树

路径和路径长度: 树中两个结点之间的路径是由这两个结点之间所经过的结点序列构成的, 而路径长度是路径上所经过的边的个数

森林: 森林是  $n$  棵互不相交的树的集合

只要把树的根结点删去就成了森林

给  $n$  棵独立的树加上一个结点, 并把这  $n$  棵树作为该结点的子树, 则森林就变成了树

树中的结点数等于所有结点的度数加1

度为  $m$  的树中第  $i$  层上至多有  $m^{i-1}$  个结点 ( $i$  大于等于1)

高度为  $h$  的  $m$  叉树至少有  $h$  个结点。

高度为  $h$  的  $m$  叉树至多有  $(m^h - 1)/(m - 1)$  个结点

高度为  $h$ 、度为  $m$  的树至少有  $h + m - 1$  个结点

### 树的性质

具有  $n$  个结点的  $m$  叉树的最小高度为  $\lceil \log_m(n(m-1) + 1) \rceil$

任意结点的度  $\leq m$  (最多  $m$  个孩子)

$m$  叉树——每个结点最多只能有  $m$  个孩子的树

允许所有结点的度都  $< m$

可以是空树

任意结点的度  $\leq m$  (最多  $m$  个孩子)

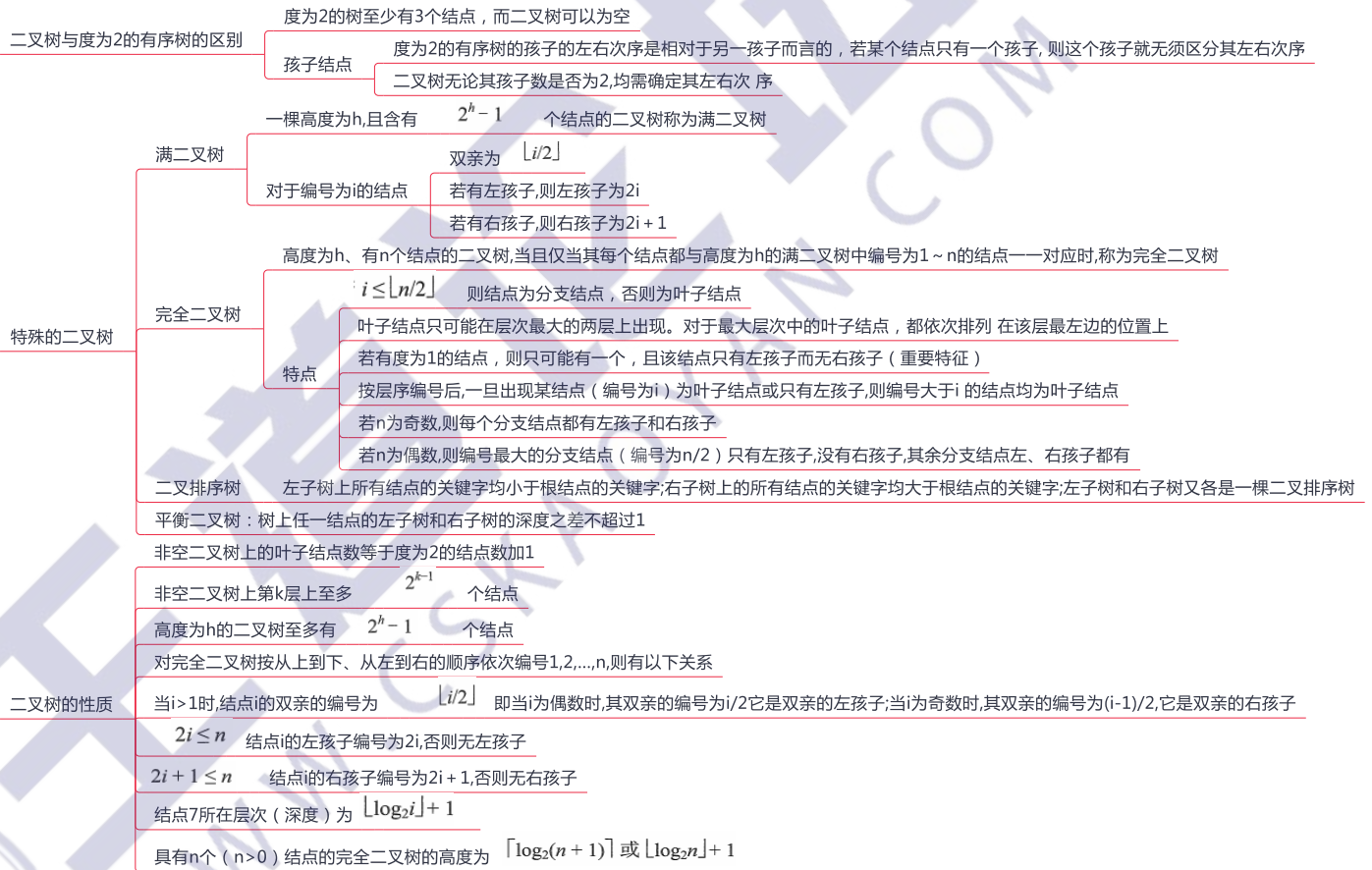
度为  $m$  的树 至少有一个结点的度  $= m$  (有  $m$  个孩子)

一定是非空树, 至少有  $m+1$  个结点

## 5.2 二叉树的概念

### 二叉树的定义及其主要特性

二叉树的定义：二叉树是另一种树形结构，其特点是每个结点至多只有两棵子树，并且二叉树的子树有左右之分，其次序不能任意颠倒



### 二叉树的存储结构

#### 顺序存储结构

用一组地址连续的存储单元依次自上而下、自左至右存储完全二叉树上的结点元素，即将完全二叉树上编号为 $i$ 的结点元素存储在一维数组下标为 $i - 1$ 的分量中

完全二叉树和满二叉树采用顺序存储比较合适

一般的二叉树，为了能反映二叉树中结点之间的逻辑关系，只能添加并不存在的空结点，让其每个结点与完全二叉树上的结点相对照，再存储到一维数组的相应分量中

由于顺序存储的空间利用率较低，因此二叉树一般都采用链式存储结构，用链表结点来存储二叉树中的每个结点

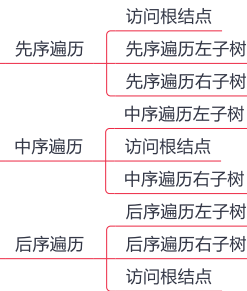
#### 链式存储结构

```
typedef struct BiTNode{
    ElemType data;           //数据域
    struct BiTNode *lchild, *rchild; //左、右孩子指针
}BiTNode, *BiTree;
```

在含有 $n$ 个结点的二叉链表中，含有 $n + 1$ 个空链域

## 5.3 二叉树的遍历和线索二叉树

### 二叉树的遍历



时间复杂度 $O(n)$

递归算法和非递归算法的转换：利用栈进行实现

### 层次遍历

利用队列实现

先将二叉树根结点入队，然后出队，访问出队结点

实现

若它有左子树，则将左子树根结点入队

若它有右子树，则将右子树根结点入队

然后出队，访问出队结点……如此反复，直至队列为空

### 由遍历序列构造二叉树

二叉树的先序（后序）序列和中序序列可以唯一地确定一棵二叉树

层序遍历与中序（后序）可以确定一颗二叉树

线索二叉树的基本概念

遍历二叉树是以一定的规则将二叉树中的结点排列成一个线性序列，从而得到几种遍历序列

使得该序列中的每个结点（第一个和最后一个结点除外）都有一个直接前驱和直接后继

规定

若无左子树，令lchild指向前驱结点

若无右子树，令rchild指向其后继结点

基本概念

结点结构

lchild	ltag	data	rtag	rchild
--------	------	------	------	--------

标志域的含义

ltag

0

lchild域指示结点的左孩子

1

lchild域指示结点的前驱

rtag

0

rchild域指示结点的右孩子

1

rchild域指示结点的后继

概念

二叉树的线索化是将二叉链表中的空指针改为指向前驱或后继的线索

而前驱或后继的信息只有在遍历时才能得到，因此线索化的实质就是遍历一次二叉树

### 线索二叉树

中序线索二叉树的构造

中序线索二叉树的建立

附设指针 pre 指向刚刚访问过的结点，指针p指向正在访问的结点，即pre指向p的前驱

在中序遍历的过程中

检查p的左指针是否为空，若为空就将它指向pre

检查pre的右指针是否为空，若为空就将它指向p

中序线索二叉树的遍历

若其右标志为"1"，则右链为线索，指示其后继，否则遍历右子树中第一个访问的结点（右子树中最左下的结点）为其后继

先序线索二叉树

先序序列为ABCDF，然后依次判断每个结点的左右链域，如果为空则将其改造为线索

结点A、B均有左右孩子；结点C无左孩子，将左链域指向前驱B，无右孩子，将右链域指向后继D

结点D无左孩子，将左链域指向前驱C，无右孩子，将右链域指向后继F

结点F无左孩子，将左链域指向前驱D，无右孩子，也无后继故置空

后序序列为CDBFA，结点C无左孩子，也无前驱故置空，无右孩子，将右链域指向后继D

后序线索二叉树

结点D无左孩子，将左链域指向前驱C，无右孩子，将右链域指向后继B

结点F无左孩子，将左链域指向前驱B，无右孩子，将右链域指向后继A

## 5.4树、森林

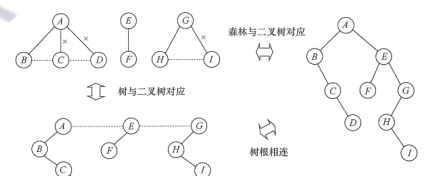
### 树的存储结构

- 双亲表示法
  - 采用一组连续空间来存储每个结点,同时在每个结点中增设一个伪指针,指示其双亲结点在数组中的位置
  - 根结点下标为0,其伪指针域为-1
  - 优点 利用了每个结点(根结点除外)只有唯一双亲的性质,可以很快得到每个结点的双亲结点
  - 缺点 但求结点的孩子时需要遍历整个结构
- 孩子表示法
  - 将每个结点的孩子结点都用单链表链接起来形成一个线性结构,此时n个结点就有n个孩子链表(叶子结点孩子链表为空表)
  - 优点 寻找子女的操作非常直接
  - 缺点 寻找双亲的操作需要遍历n个结点中孩子链表指针域所指向的n个孩子链表
- 孩子兄弟表示法
  - 以二叉链表作为树的存储结构,孩子兄弟表示法使每个结点包括三部分内容:结点值、指向结点第一个孩子结点的指针,及指向结点下一个兄弟结点的指针
  - 优点 这种存储表示法比较灵活,其最大的优点是可以方便地实现树转换为二叉树的操作,易于查找结点的孩子等
  - 缺点 从当前结点查找其双亲结点比较麻烦

若为每个结点增设一个parent域指向其父结点,则查找结点的父结点也很方便

### 树、森林与二叉树的转换

- 树转换为二叉树的规则 每个结点左指针指向它的第一个孩子,右指针指向它在树中的相邻右兄弟
- 树转换成二叉树的画法
  - 在兄弟结点之间加一连线
  - 对每个结点,只保留它与第一个孩子的连线,而与其他孩子的连线全部抹掉
  - 以树根为轴心,顺时针旋转45度
- 森林转换成二叉树的画法
  - 将森林中的每棵树转换成相应的二叉树
  - 每棵树的根也可视为兄弟关系,在每棵树的根之间加一根连线
  - 以第一棵树的根为轴心顺时针旋转45°
- 二叉树转换为森林的规则
  - 若二叉树非空,则二叉树的根及其左子树为第一棵树的二叉树形式,故将根的右链断开
  - 二叉树根的右子树又可视为一个由除第一棵树外的森林转换后的二叉树
  - 应用同样的方法,直到最后只剩一棵没有右子树的二叉树为止,最后再将每棵二叉树依次转换成树,就得到了原森林



森林与二叉树的对应关系

### 树和森林的遍历

- 先根遍历
  - 若树非空,先访问根结点,再依次遍历根结点的每棵子树,遍历子树时仍遵循先根后子树的规则
  - 其遍历序列与这棵树相应二叉树的先序序列相同
- 后根遍历
  - 若树非空,先依次遍历根结点的每棵子树,再访问根结点,遍历子树时仍遵循先子树后根的规则
  - 其遍历序列与这棵树相应二叉树的中序序列相同
- 先序遍历森林
  - 访问森林中第一棵树的根结点
  - 先序遍历第一棵树中根结点的子树森林
  - 先序遍历除去第一棵树之后剩余的树构成的森林
- 中序遍历森林
  - 中序遍历森林中第一棵树的根结点的子树森林
  - 访问第一棵树的根结点
  - 中序遍历除去第一棵树之后剩余的树构成的森林

树和森林的遍历与二叉树遍历的对应关系

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历


### 树的应用——并查集

- 支持操作
  - 1) Union ( S,Root1,Root2 ):把集合S中的子集合Root2并入子集合Root1。要求Root1 和Root2互不相交,否则不执行合并
  - 2) Find(S,x):查找集合S中元素x所在的子集,并返回该子集的名字
  - 3) Initial(S ):将集合S中的每个元素都初始化为只有一个单元素的子集合
- 具体实现
  - 用树(森林)的双亲表示作为并查集的存储结构,每个子集合以一棵树表示
  - 所有表示子集合的树,构成表示全集合的森林,存放在双亲表示数组内
  - 通常用数组元素的下标代表元素名,用根结点的下标代表子集合名,根结点的双亲结点为负数



## 5.5 树与二叉树的应用（上）

### 二叉排序树 (BST)

二叉排序树的定义	特性	1) 若左子树非空,则左子树上所有结点的值均小于根结点的值 2) 若右子树非空,则右子树上所有结点的值均大于根结点的值 3) 左、右子树也分别是一棵二叉排序树	<b>左子树结点值 &lt; 根结点值 &lt; 右子树结点值</b>
二叉排序树的查找	二叉排序树的查找是从根结点开始,沿某个分支逐层向下比较的过程,若二叉排序树非空,先将给定值与根结点的关键字比较	若相等,则查找成功 若不等,如果小于根结点的关键字,则在根结点的左子树上查找 否则在根结点的右子树上查找	<b>递归</b>
二叉排序树的插入	插入结点的过程	若原二叉排序树为空,则直接插入结点 若关键字k小于根结点值,则插入到左子树 若关键字k大于根结点值,则插入到右子树	<b>插入的结点一定是一个新添加的叶结点,且是查找失败时的查找路径上访问的最后一个结点的左孩子或右孩子</b>
二叉排序树的构造	生成示意图	设查找的关键字序列为{45,24,53,45,12,24},则生成的二叉排序树 	
二叉排序树的删除		若被删除结点z是叶结点,则直接删除,不会破坏二叉排序树的性质 若结点z只有一棵左子树或右子树,则让z的子树成为z父结点的子树,替代z的位置 若结点z有左、右两棵子树,则令z的直接后继(或直接前驱)替代z,然后从二叉排序树中删去这个直接后继(或直接前驱),这样就转换成了第一或第二种情况	
二叉排序树的查找效率分析	平均执行时间为		

## 5.5 树与二叉树的应用（中）

### 平衡二叉树

**平衡二叉树的定义** 在插入和删除二叉树结点时, 要保证任意结点的左、右子树高度差的绝对值不超过 1, 将这样的二叉树称为平衡二叉树  
平衡因子: 左子树与右子树的高度差为该结点的平衡因子 平衡因子的值只可能是 -1、0 或 1

首先检查其插入路径上的结点是否因为此次操作而导致了不平衡

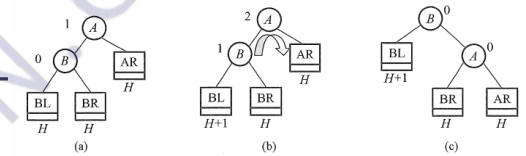
当在二叉排序树中插入 (或删除) 一个结点时

若导致了不平衡, 则先找到插入路径上离插入结点最近的平衡因子的绝对值大于 1 的结点 A, 再对以 A 为根的子树, 在保持二叉排序树特性的前提下, 调整各结点的位置关系, 使之重新达到平衡

平衡二叉树的插入

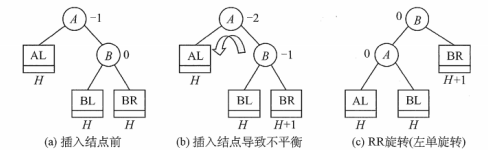
LL 平衡旋转 (右单旋转)

由于在结点 A 的左孩子 (L) 的左子树 (L) 上插入了新结点  
A 的平衡因子由 1 增至 2, 导致以 A 为根的子树失去平衡, 需要一次向右的旋转操作



RR 平衡旋转 (左单旋转)

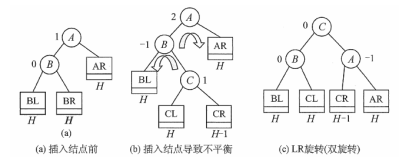
由于在结点 A 的右孩子 (R) 的右子树 (R) 上插入了新结点  
A 的平衡因子由 -1 减至 -2, 导致以 A 为根的子树失去平衡, 需要一次向左的旋转操作



规律归纳

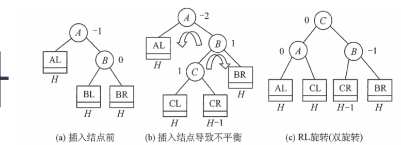
LR 平衡旋转 (先左右后右双旋转)

由于在 A 的左孩子 (L) 的右子树 (R) 上插入新结点  
A 的平衡因子由 1 增至 2, 导致以 A 为根的子树失去平衡, 需要进行两次旋转操作, 先左旋转后右旋转



RL 平衡旋转 (先右后左双旋转)

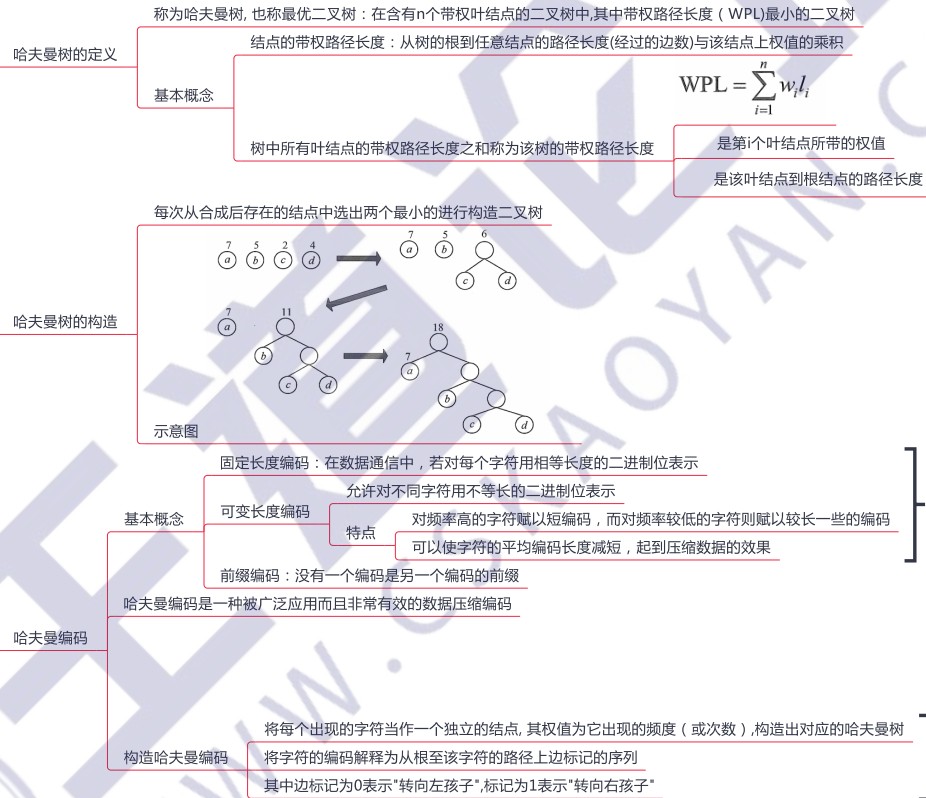
由于在 A 的右孩子 (R) 的左子树 (L) 上插入新结点  
A 的平衡因子由 -1 减至 -2, 导致以 A 为根的子树失去平衡, 需要进行两次旋转操作, 先右旋转后左旋转



平衡二叉树的查找 平均查找长度

## 5.5树与二叉树的应用（下）

### 哈夫曼树和哈夫曼编码



可变长度编码比固定长度编码要好

示意图

各字符编码为  
a:0  
b:101  
c:100  
d:111  
e:1101  
f:1100

