

根和子根之间的连线为“分支”。

结点分支的个数定义为“结点的度”，如结点B的度为2，D的度为3。

树中所有结点度的最大值定义为“树的度”。

度为0的结点为“叶子”或“终端结点”。

所有度不为0的结点被称作“分支结点”。

基本定义：

森林为m(m>=0)棵互不相交的树的集合。

树的深度定义为树中叶子结点所在最大层次数。

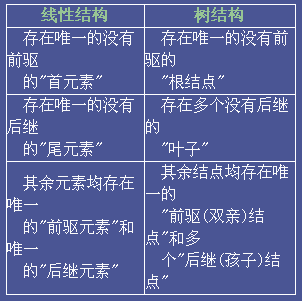
称根结点为子树根的“双亲”。

称子树根为根结点的“孩子”。

根的所有子树根互为“兄弟”。

有序树、无序树。如果树中每棵子树从左向右的排列拥有一定的顺序，不得互换，则成为有序树，否则称为无序树。

树和线性结构对照：



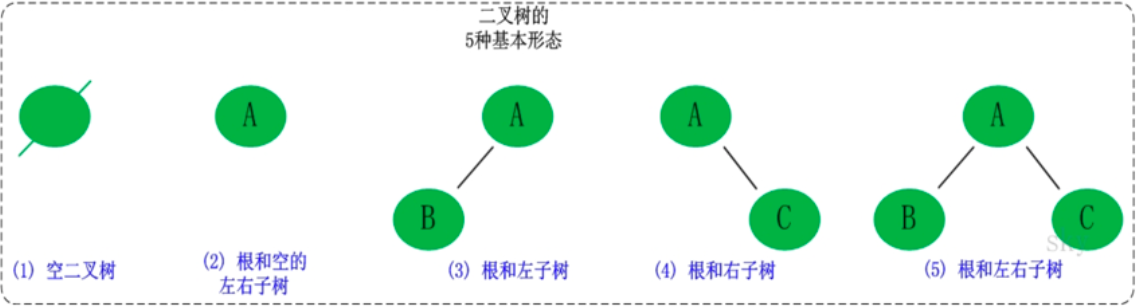
#### 二叉树的概念

二叉树，又是一种特殊的树，二叉树的任意一个结点的度都不大于2。

二叉树也可以用递归的形式定义。即是n（n>=0）个结点的有限集合。

* n=0时，空二叉树；
* n>0时，有一个根结点和互不相交的，分别称为左子树和右子树的二叉树组成。

二叉树的5种形态：



遍历方式：二叉树的遍历方式有三种，中序遍历，先序遍历，后序遍历：

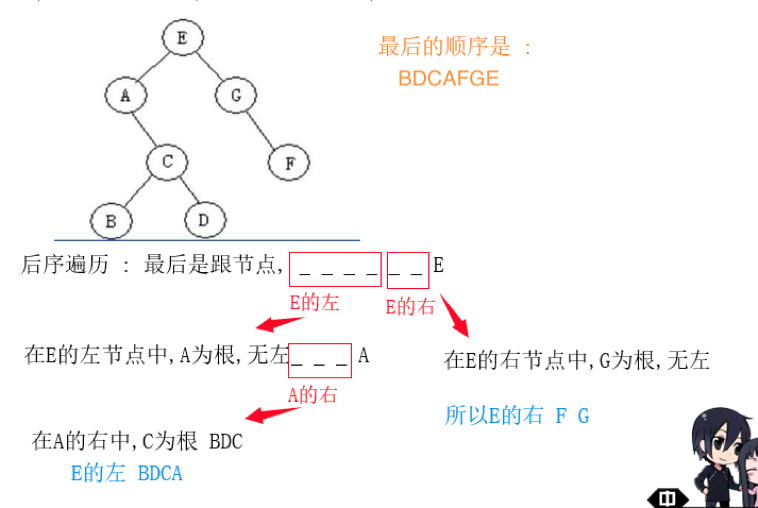
1. 中序遍历：左结点 -> 根结点 -> 右结点；



1. 前序遍历：根结点 -> 左结点 -> 右结点；



1. 后序遍历：左结点 -> 右结点 -> 根结点；



1. 层次遍历：按照一层一层的遍历



例：已知一棵二叉树，如果先序遍历的节点顺序是：ADCEFGHB，中序遍历是：CDFEGHAB，则后序遍历结果为？

首先，先序遍历最早遍历父亲结点，也就意味着A是根结点，而中序遍历最晚遍历右子树，所以推测出A的右边也就是B，是A的右子树。于是，我们将两个去除AB，也就是DCEFGH，CDFEGH，推测出父节点左子树的结果，同样的方法可以确定D是左子树的父亲结点，C是左子树的唯一一个左子树，从而去除。其实就是一个递归的思想，不断地确认父结点，以及拆分左子树，和右子树，完成树的重构。



#### 二叉查找树（BST）的概念

每个结点都含有一个Comparable的键（以及对应的值）。每个结点的键都大于左子树中任意结点的键而小于右子树中任意结点的键。



同时，左子树和右子树也为二叉BST。二叉查找树中序遍历后，会得到一个有序的元素集合。上图中序遍历后为{1,2,3,4,6,8}

优点：良好的查找性能，查找次数最多为树的深度，这也就意味着二叉查找树的查找效率取决于树的深度，而满二叉树的深度为log(N+1)，所以此时它的查找时间复杂度为O(log2N)。

缺点：在极端情况下会大大降低查找性能，比如说，顺序插入结点时，二叉树就会像一个链表，查找的时间复杂度将为O(N)。

#### 二叉查找树的基本操作和Java实现

**insert**

插入时，要满足左小右大。

实现思路：递归地去遍历一棵树，如果大于结点就遍历结点的右子树，如果小于结点就遍历结点的左子树，当结点为空时插入。

#### 满二叉树和完全二叉树

