# 二叉树

## 树

### 结点

是数据结构中的基础，是构成复杂数据结构的基本组成单位。

结点拥有的子树数目称为**结点的度**。

### 树的概念

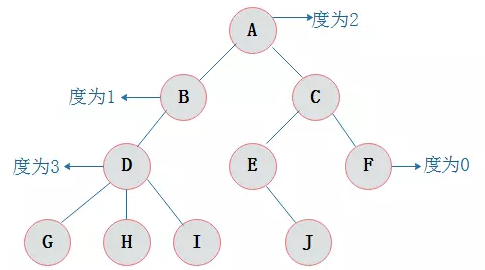
树是n（n>=0)个结点的有限集。n=0时称为空树。在任意一颗非空树中：  
1）有且仅有一个特定的称为根（Root）的结点；  
2）当n>1时，其余结点可分为m(m>0)个互不相交的有限集T1、T2、......、Tn，其中每一个集合本身又是一棵树，并且称为根的子树。

此外，树的定义还需要强调以下两点：  
1）n>0时根结点是唯一的，不可能存在多个根结点，数据结构中的树只能有一个根结点。  
2）m>0时，子树的个数没有限制，但它们一定是互不相交的。

树中结点的最大层次数称为**树的深度或高度。**

### 结点关系

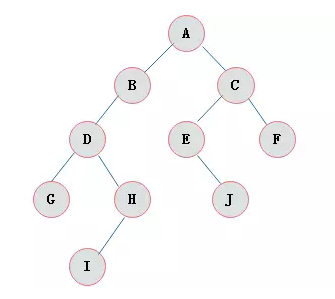
结点子树的根结点为该结点的**孩子结点**。相应该结点称为孩子结点的**双亲结点**。  
图2.2中，A为B的双亲结点，B为A的孩子结点。  
同一个双亲结点的孩子结点之间互称**兄弟结点**。  
图2.2中，结点B与结点C互为兄弟结点。



## 二叉树

### 定义

**二叉树**是n(n>=0)个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树组成。  
图3.1展示了一棵普通二叉树：



### 特点

由二叉树定义以及图示分析得出二叉树有以下特点：  
1）每个结点最多有两颗子树，所以二叉树中不存在度大于2的结点。  
2）左子树和右子树是有顺序的，次序不能任意颠倒。  
3）即使树中某结点只有一棵子树，也要区分它是左子树还是右子树。

### 性质

1）在二叉树的第i层上最多有2i-1 个节点 。（i>=1）  
2）二叉树中如果深度为k,那么最多有2k-1个节点。(k>=1）  
3）n0=n2+1 n0表示度数为0的节点数，n2表示度数为2的节点数。

解析：n个结点共有2n个指针，同时又n-1个指针指向对应的结点（除根节点外其他结点必定被父节点所指向），所以就有n+1个指针未指向任何结点，所以有:

2\*n0 + n1 = n+1;n1+2\*n2=n-1

根据上述两个式子可得出n0和n2的关系。  
4）在完全二叉树中，具有n个节点的完全二叉树的深度为[log2n]+1，其中[log2n]是向下取整。  
5）若对含 n 个结点的完全二叉树从上到下且从左至右进行 1 至 n 的编号，则对完全二叉树中任意一个编号为 i 的结点有如下特性：

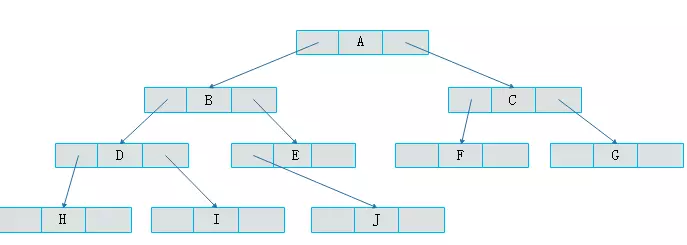
a 若 i=1，则该结点是二叉树的根，无双亲, 否则，编号为 [i/2] 的结点为其双亲结点;  
b 若 2i>n，则该结点无左孩子， 否则，编号为 2i 的结点为其左孩子结点；  
c 若 2i+1>n，则该结点无右孩子结点， 否则，编号为2i+1 的结点为其右孩子结点。

### 数据结构

二叉树采用链式存储，由二叉树定义可知，二叉树的每个结点最多有两个孩子。因此，可以将结点数据结构定义为一个数据和两个指针域。表示方式如图3.11所示：



二叉树的结果如下图：



### 二叉树遍历

**二叉树的遍历**是指从二叉树的根结点出发，按照某种次序依次访问二叉树中的所有结点，使得每个结点被访问一次，且仅被访问一次。  
二叉树的访问次序可以分为四种：

前序遍历

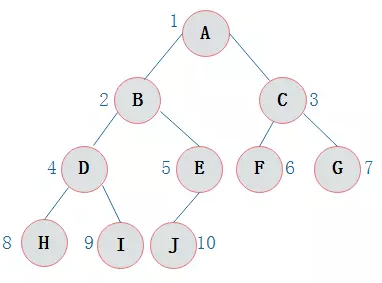
中序遍历

后序遍历

层序遍历

#### 前序遍历

**前序遍历**通俗的说就是从二叉树的根结点出发，当第一次到达结点时就输出结点数据，按照先向左在向右的方向访问。



上图所示二叉树遍历过程：从根结点出发，则第一次到达结点A，故输出A;继续向左访问，第一次访问结点B，故输出B；按照同样规则，输出D，输出H；当到达叶子结点H，返回到D，此时已经是第二次到达D，故不在输出D，进而向D右子树访问，D右子树不为空，则访问至I，第一次到达I，则输出I；I为叶子结点，则返回到D，D左右子树已经访问完毕，则返回到B，进而到B右子树，第一次到达E，故输出E；向E左子树，故输出J；按照同样的访问规则，继续输出C、F、G；

则3.13所示二叉树的前序遍历输出为：  
**ABDHIEJCFG**

#### 中序遍历

**中序遍历**就是从二叉树的根结点出发，当第二次到达结点时就输出结点数据，按照先向左在向右的方向访问。

遍历过程：从根结点出发，则第一次到达结点A，不输出A，继续向左访问，第一次访问结点B，不输出B；继续到达D，H；到达H，H左子树为空，则返回到H，此时第二次访问H，故输出H；H右子树为空，则返回至D，此时第二次到达D，故输出D；由D返回至B，第二次到达B，故输出B；按照同样规则继续访问，输出J、E、A、F、C、G；

中序遍历输出为：

HDIBEJAFCG

#### 后序遍历

**后序遍历**就是从二叉树的根结点出发，当第三次到达结点时就输出结点数据，按照先向左在向右的方向访问。

图3.13所示二叉树后序访问如下：从根结点出发，则第一次到达结点A，不输出A，继续向左访问，第一次访问结点B，不输出B；继续到达D，H；到达H，H左子树为空，则返回到H，此时第二次访问H，不输出H；H右子树为空，则返回至H，此时第三次到达H，故输出H；由H返回至D，第二次到达D，不输出D；继续访问至I，I左右子树均为空，故第三次访问I时，输出I；返回至D，此时第三次到达D，故输出D；按照同样规则继续访问，输出J、E、B、F、G、C，A；

则图3.13所示二叉树的后序遍历输出为：

**HIDJEBFGCA**

虽然二叉树的遍历过程看似繁琐，但是由于二叉树是一种递归定义的结构，故采用递归方式遍历二叉树的代码十分简单。

#### 遍历结论

1. 已知前序遍历序列和中序遍历序列，确定一棵二叉树。

前序遍历第一个输出结点为根结点，中序遍历根节点左边的在左边，右边的在右边。

1. 已知后序遍历序列和中序遍历序列，确定一棵二叉树
2. 已知前序遍历序列和后序遍历序列，不可以唯一确定一棵二叉树

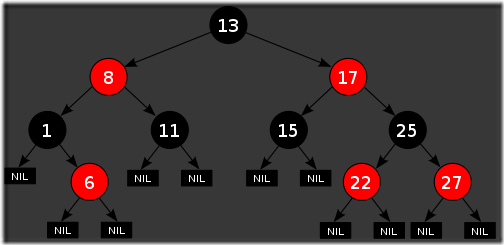
# 红黑树

## 简介

 R-B Tree，全称是Red-Black Tree，又称为“红黑树”，它一种特殊的二叉查找树。红黑树的每个节点上都有存储位表示节点的颜色，可以是红(Red)或黑(Black)。

**红黑树的特性:**

（1）每个节点或者是黑色，或者是红色。  
（2）根节点是黑色。  
（3）每个叶子节点（NIL）是黑色。  
（4）如果一个节点是红色的，则它的子节点必须是黑色的。  
（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。



**注意：**

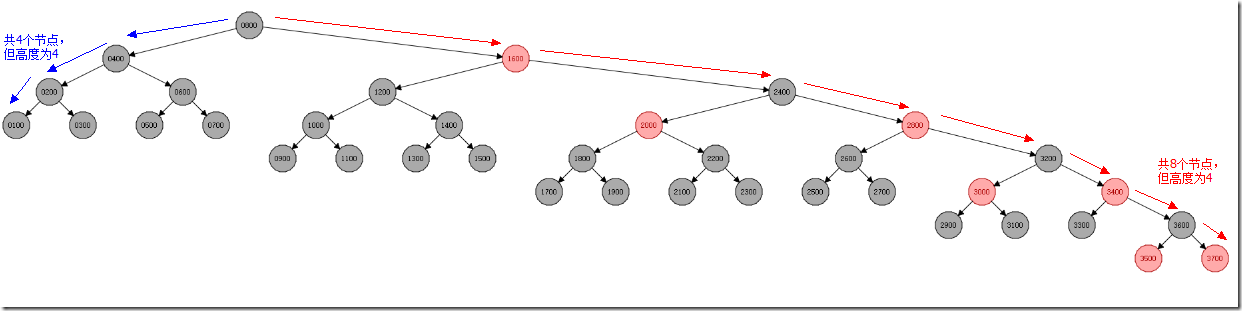
这些约束使红黑树具有这样一个关键属性：从根节点到最远的叶子节点的路径长与到最近的叶子节点的路径长度相差不会超过1倍。 因为红黑树是近似平衡的。

**解释**

释一下为什么有这样好的效果。注意性质4和性质5。假设一个红黑树T，其到叶节点的最短路径肯定全部是黑色节点（共B个），最长路径肯定有相同个黑色节点（性质5：黑色节点的数量是相等），另外会多几个红色节点。性质4（红色节点必须有两个黑色儿子节点）能保证不会再现两个连续的红色节点。所以最长的路径长度应该是2B个节点，其中B个红色，B个黑色。

最短的路径中全部是黑色节点，最长的路径中既有黑色又有红色节点。

因为这两个路径中黑色节点个数是一样的，而且不会出现两个连续的红色节点，所以最长的路径可能会出现红黑相间的节点。也就是说，树中任意两条路径中的节点数相差不会超过一倍。



## 时间复杂度及应用

红黑树的应用比较广泛，主要是用它来存储有序的数据，它的时间复杂度是O(lgn)，效率非常之高。例如，Java集合中的[TreeSet](http://www.cnblogs.com/skywang12345/p/3311268.html)和[TreeMap](http://www.cnblogs.com/skywang12345/p/3310928.html)，C++ STL中的set、map，以及Linux虚拟内存的管理，都是通过红黑树去实现的。