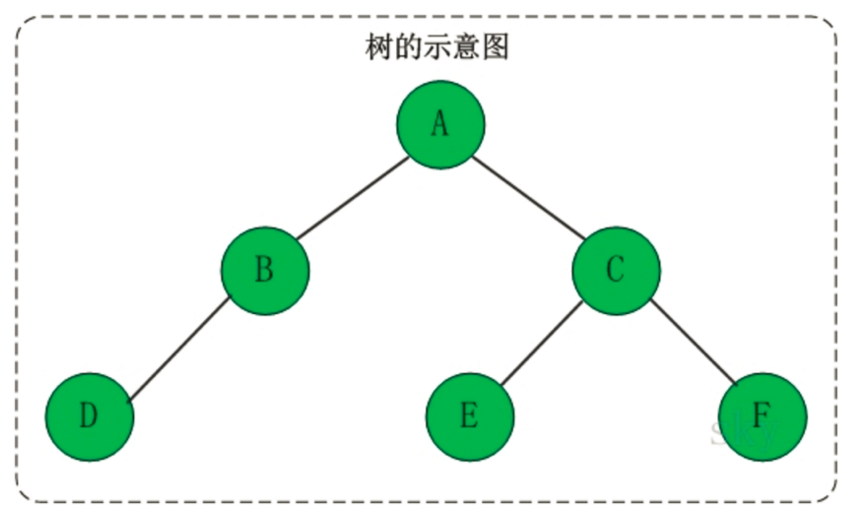
二叉树算法

# 树的定义

树是一种数据结构，它是由n（n>=1）个有限结点组成一个具有层次关系的集合。



树具有的特点有：

### 每个结点**有零个或多个子结点**；

### 没有父节点的结点称为**根节点**；

### 每一个非根结点**有且只有一个父节点**；

### 除了根结点外，每个子结点可以分为多个不相交的子树。

树的基本术语有：

若一个结点有子树，那么该结点称为**子树根的“双亲”**，子树的根称为**该结点的“孩子”**。有相同双亲的结点互为**“兄弟”**。一个结点的所有子树上的任何结点都是该结点的**后裔**。从根结点到某个结点的路径上的所有结点都是该结点的**祖先**。

**结点的度**：结点拥有的子树的数目；

**叶子结点**：度为0的结点；

**分支结点**：度不为0的结点；

**树的度**：树中结点的最大的度；

**层次**：根结点的层次为1，其余结点的层次等于该结点的双亲结点的层次加1；

**树的高度**：树中结点的最大层次；

**森林**：0个或多个不相交的树组成。对森林加上一个根，森林即成为树；删去根，树即成为森林。

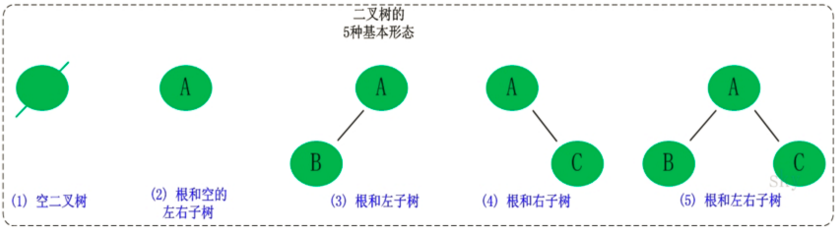
# 二叉树BinTree

## 二叉树的定义

**二叉树是每个结点最多有两个子树的树结构**。

它有五种基本形态：二叉树可以是空集；根可以有空的左子树或右子树；或者左、右子树皆为空。

二叉树节点：**BinTreeNode**



## 二叉树的性质

性质1：二叉树第i层上的结点数目最多为**2i-1**(i>=1)

性质2：深度为k的二叉树至多有2k-1个结点（k>=1）

**证明：20+21+…+2k-1 = 1\*(1-2k)/(1-2)= 2k-1**。

性质3：包含n个结点的二叉树的高度至少为**(log2n)+1**

性质4：在任意一棵二叉树中，若终端结点的个数为n0，度为2的结点数为n2，则n0=n2+1

## 性质4的证明

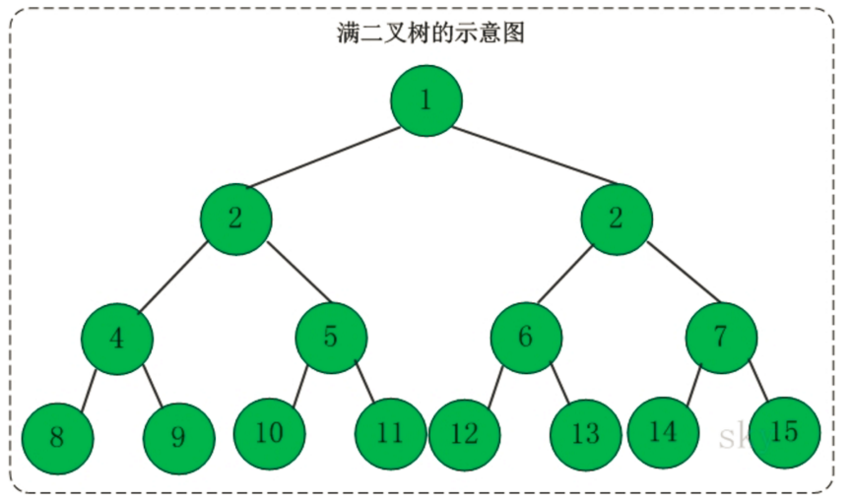
**性质4**：在任意一棵二叉树中，若终端结点的个数为n0，度为2的结点数为n2，则n0=n2+1 **证明**：因为二叉树中所有结点的度数均不大于2，不妨设n0表示度为0的结点个数，n1表示度为1的结点个数，n2表示度为2的结点个数。三类结点加起来为总结点个数，于是便可得到：**n=n0+n1+n2  (1)** 由度之间的关系可得第二个等式：n=n0\*0+n1\*1+n2\*2+1即**n=n1+2\*n2+1  (2)** 将（1）（2）组合在一起可得到**n0=n2+1**。

# 常见的二叉树

## 满二叉树

定义：高度为h，并且由2h-1个结点组成的二叉树，称为**满二叉树**。

除最后一层外，每一层上的所有节点都有两个子节点，最后一层都是叶子节点。

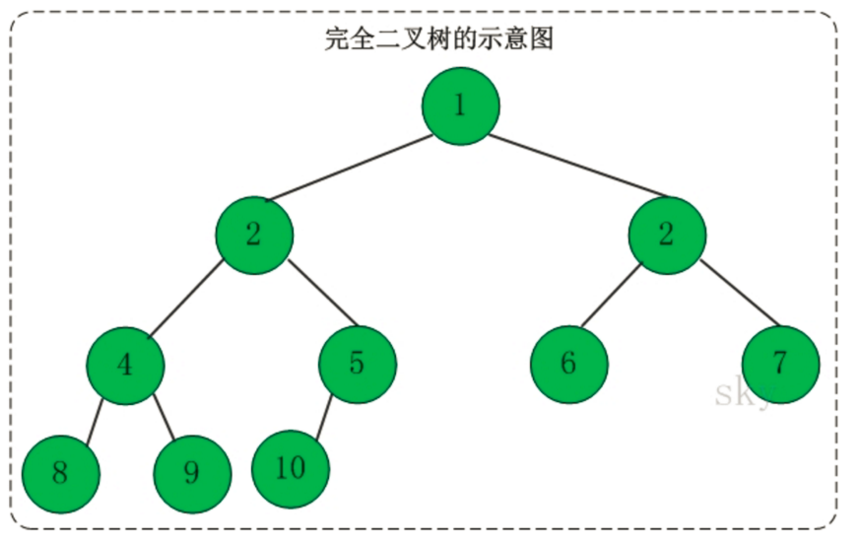


## 完全二叉树

定义：**一棵二叉树中，只有最下面两层结点的度可以小于2，并且最下层的叶结点集中在靠左的若干位置上**，这样的二叉树称为**完全二叉树**。

特点：**叶子结点只能出现在最下层和次下层，且最下层的叶子结点集中在树的左部**。显然，一棵满二叉树必定是一棵完全二叉树，而完全二叉树未必是满二叉树。

若二叉树的高度是h，除第h层之外，其他（1~h-1）层的节点数都达到了最大个数，并且第h层的节点都连续的集中在最左边。想到点什么没？实际上，完全二叉树和堆联系比较紧密哈



**面试题**：如果一个完全二叉树的结点总数为768个，求叶子结点的个数。

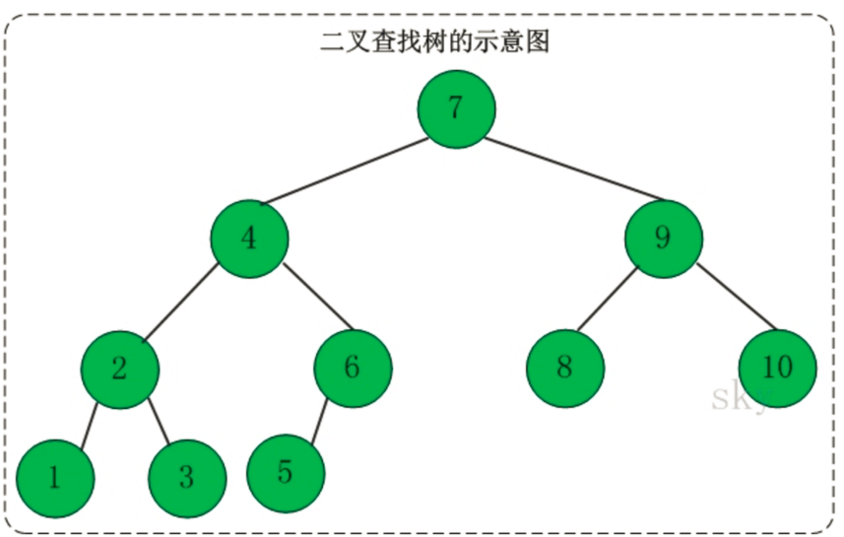
由二叉树的性质知：n0=n2+1，将之带入768=n0+n1+n2中得：768=n1+2n2+1，因为完全二叉树度为1的结点个数要么为0，要么为1，那么就把n1=0或者1都代入公式中，很容易发现n1=1才符合条件。所以算出来n2=383，所以叶子结点个数n0=n2+1=384。

**总结规律**：如果一棵完全二叉树的结点总数为n，那么叶子结点等于n/2（当n为偶数时）或者(n+1)/2（当n为奇数时）。

## 二叉查找树

定义：二叉查找树又被称为二叉搜索树。

设x为二叉查找树中的一个结点，x结点包含关键字key，结点x的key值计为key[x]。如果y是x的左子树中的一个结点，则key[y]<=key[x]；如果y是x的右子树的一个结点，则key[y]>=key[x]



在二叉查找树中：

（1）若任意结点的左子树不空，则左子树上所有结点的值**均小于**它的根结点的值。

（2）任意结点的右子树不空，则右子树上所有结点的值**均大于**它的根结点的值。

（3）任意结点的左、右子树也分别为二叉查找树。

（4）没有键值相等的结点。

## 红黑树

红黑树：红黑树是每个节点都带颜色的树，节点颜色或是红色或是黑色，**红黑树是一种查找树**。红黑树有一个重要的性质，从根节点到叶子节点的最长的路径不多于最短的路径的长度的两倍。对于红黑树，插入，删除，查找的复杂度都是O（log N）。

## 哈夫曼树

又称为最优数，这是一种**带权路径长度最短的树**。哈夫曼编码就是哈夫曼树的应用。

## 平衡二叉树

平衡二叉树：所谓平衡二叉树指的是，左右两个子树的高度差的绝对值不超过 1。

# 二叉树的三种遍历

二叉树遍历：

所谓遍历(Traversal)是指沿着某条搜索路线，依次对树中每个结点均做一次且仅做一次访问。访问结点所做的操作依赖于具体的应用问题。

遍历是二叉树上最重要的运算之一，是二叉树上进行其它运算之基础。

二叉树的遍历分为以下三种：

根据访问结点操作发生位置命名：

### **NLR：前序遍历(Preorder Traversal 亦称（先序遍历））**

——**访问根结点的操作**发生在遍历其左右子树之前；遍历顺序规则为【根左右】。

### LNR：中序遍历(Inorder Traversal)

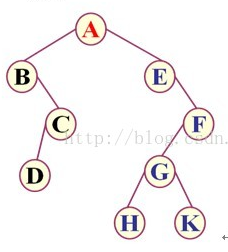
——**访问根结点的操作**发生在遍历其左右子树之中（间）；遍历顺序规则为【左根右】

### LRN：后序遍历(Postorder Traversal)

——**访问根结点的操作**发生在遍历其左右子树之后；遍历顺序规则为【左右根】

什么是【根左右】？就是先遍历根，再遍历左结点，最后遍历右结点。

注意：由于被访问的结点必是某子树的根，所以**N(Node）、L(Left subtree）和R(Right subtree）又可解释为根、根的左子树和根的右子树**。NLR、LNR和LRN分别又称为先根遍历、中根遍历和后根遍历。



先序遍历：ABCDEFGHK

中序遍历：BDCAEHGKF

后序遍历：DCBHKGFEA

**方法**：从根节点root开始，根据三种遍历原则，进行遍历，不断的更新根节点，进行遍历。

(**其实真的很简单**)

以中序遍历为例：

中序遍历的规则是【左根右】，我们从**root节点A**看起；

此时**A是根节点**，遍历A的左子树；

A的左子树存在，找到B，此时B看做根节点，遍历B的左子树；

B的左子树不存在，返回B，根据【左根右】的遍历规则，记录B，遍历B的右子树；

B的右子树存在，找到C，**此时C看做根节点**，遍历C的左子树；

C的左子树存在，找到D，由于D是叶子节点，无左子树，记录D，无右子树，返回C，根据【左根右】的遍历规则，记录C，遍历C的右子树；

C的右子树不存在，返回B，B的右子树遍历完，返回A；

至此，A的左子树遍历完毕，根据【左根右】的遍历规则，记录A，遍历A的右子树；

A的右子树存在，找到E，此时E看做根节点，遍历E的左子树；

E的左子树不存在，返回E，根据【左根右】的遍历规则，记录E，遍历E的右子树；

E的右子树存在，找到F，此时**F看做根节点**，遍历F的左子树；

F的左子树存在，找到G，此时**G看做根节点**，遍历G的左子树；

G的左子树存在，找到H，由于H是叶子节点，无左子树，记录H，无右子树，返回G，根据【左根右】的遍历规则，记录G，遍历G的右子树；

G的右子树存在，找到K，由于K是叶子节点，无左子树，记录K，无右子树，返回G，根据【左根右】的遍历规则，记录F，遍历F的右子树；

F的右子树不存在，返回F，E的右子树遍历完毕，返回A；

至此，A的右子树也遍历完毕；

最终我们得到上图的中序遍历为BDCAEHGKF，无非是按照遍历规则来的；