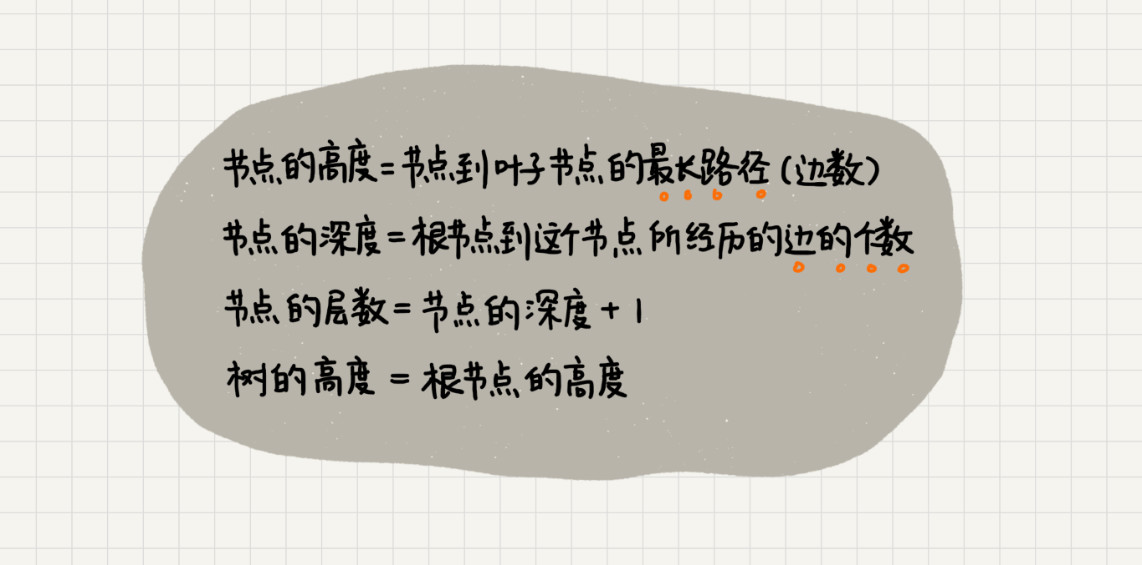
**一. 树**

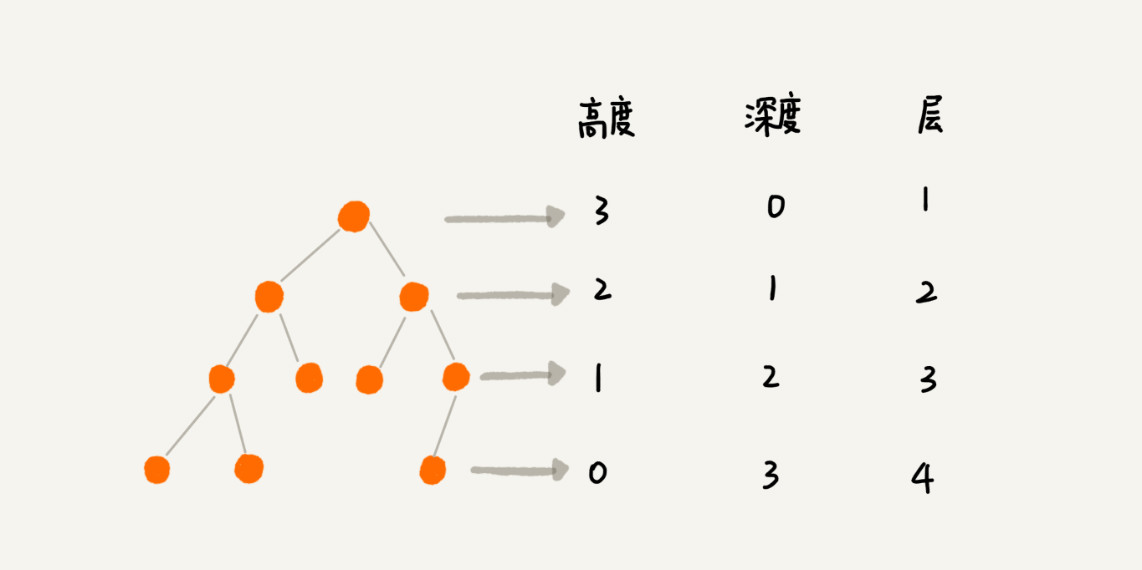
* 每个节点有零个或多个子节点
* 没有父节点的节点称为根节点
* 每一个非根结点有且只有一个父节点
* 除了根节点外，每个子节点可以分为多个不相交的字树

**二. 树的术语**

* 节点的度：一个节点含有的子树的个数
* 树的度：一棵树中，最大的节点的度
* 叶子节点：度为零的节点
* 父节点：若一个节点含有子节点，则这个节点成为其子节点的父节点
* 子节点：一个节点含有的子树的根节点称为该节点的子节点
* 兄弟节点：具有相同父节点的节点互称为兄弟节点
* 节点的祖先：从根到该节点所经分支上的所有的节点
* 子孙：以某节点为根的子树中任意节点都成为该节点的子孙
* 森林：由m(m>=0)棵不相交的树的结合成为森林

除此之外，关于“树”，还有三个比较相似的概念：高度（Height）、深度（Depth）、层（Level）。它们的定义是这样的：

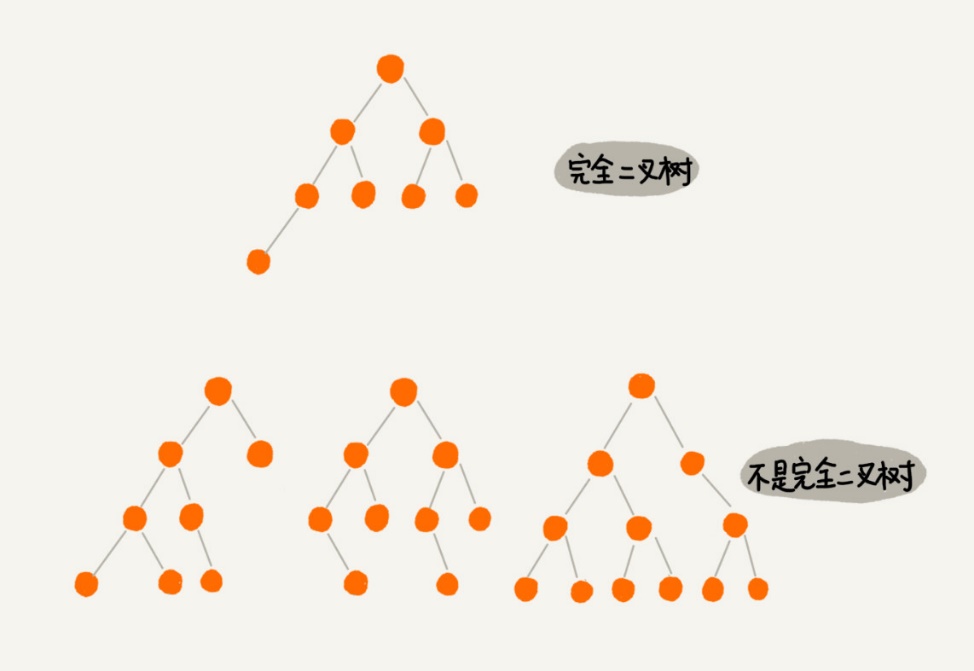




**三. 二叉树(Binary Tree)**

二叉树每个节点最多含有两个子树

1. 完全二叉树：除了最后一层以外，其余层的节点个数都要达到最大，最后一层的叶子节点从左向右连续紧密排列。



1. 满二叉树：叶子节点全部都在最底层，除了叶子节点以外，每个节点都有左右两个节点
2. 平衡二叉树(AVL树)：当且仅当任何节点的两棵子树高度差不大于1的二叉树
3. 二叉搜索树 Binary Search Tree: 在树中的任意一个节点，其左子树中的每个节点的值，都要小于这个节点的值，而右子树节点的值都大于这个节点的值。

Ps: 有序树：树中任意节点的子节点之间有顺序关系。包括二叉树，霍夫曼树（带权路径最短的二叉树，最优二叉树），B树（优化读写操作的自平衡二叉查找树，拥有多余两个子树）

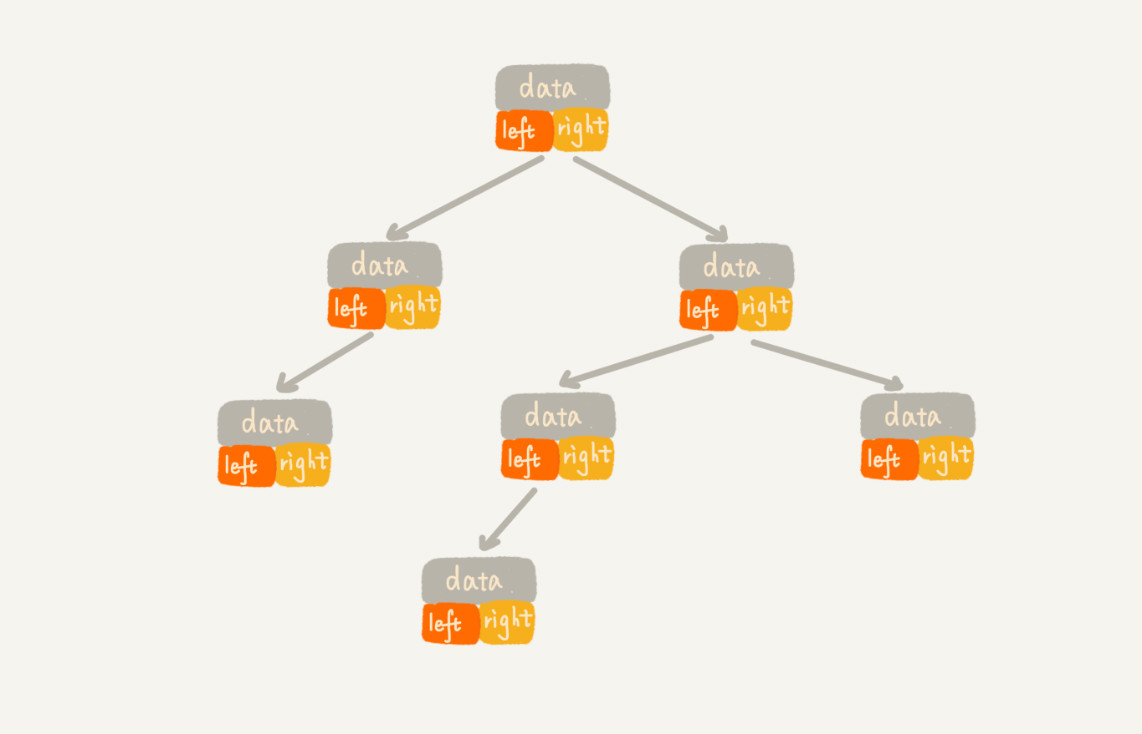
性质：

* 在二叉树第i层上至多有2^（i-1）个节点(i>0)
* 在最大层次为k的二叉树中至多有2^k – 1个节点(k>0)
* 具有n个节点的完全二叉树的向上取整有层
* 对于任意一棵二叉树，如果叶节点数为N0，而度数为2的节点总数为N2，则N0=N2+1

**四. 二叉树存储**

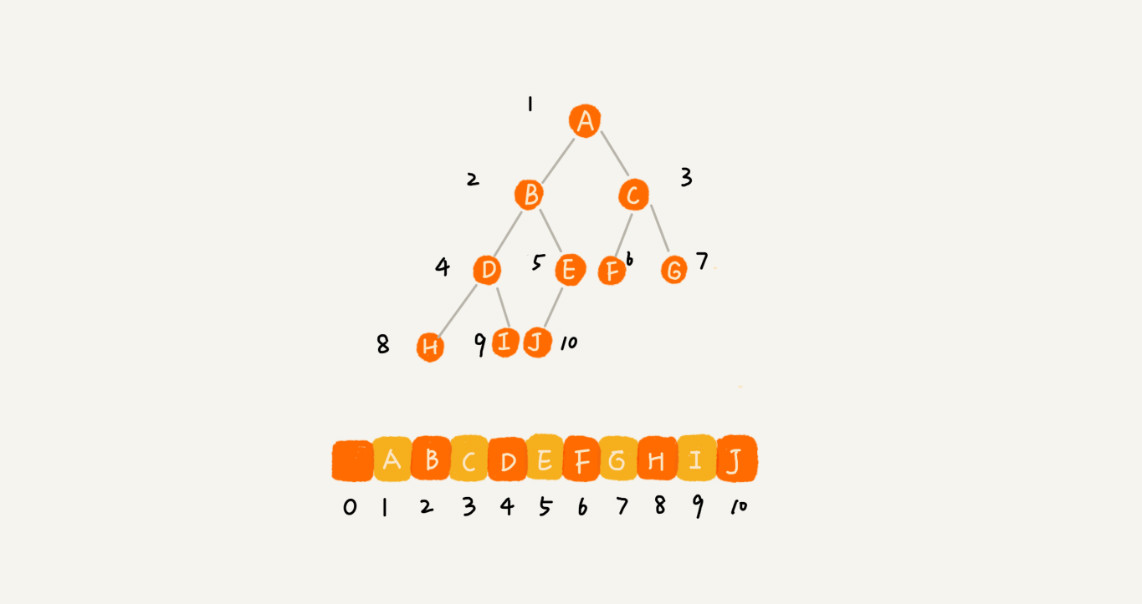
1. 链式存储法：

每个节点有三个字段，其中一个存储数据，另外两个是指向左右子节点的指针。我们只要拎住根节点，就可以通过左右子节点的指针，把整棵树都串起来。这种存储方式我们比较常用。大部分二叉树代码都是通过这种结构来实现的。



2. 顺序存储法:

基于数组的顺序存储法。我们把根节点存储在下标 i = 1 的位置。如果节点 X 存储在数组中下标为 i 的位置，下标为 2 \* i 的位置存储的就是左子节点，下标为 2 \* i + 1 的位置存储的就是右子节点。反过来，下标为 i/2 的位置存储就是它的父节点。

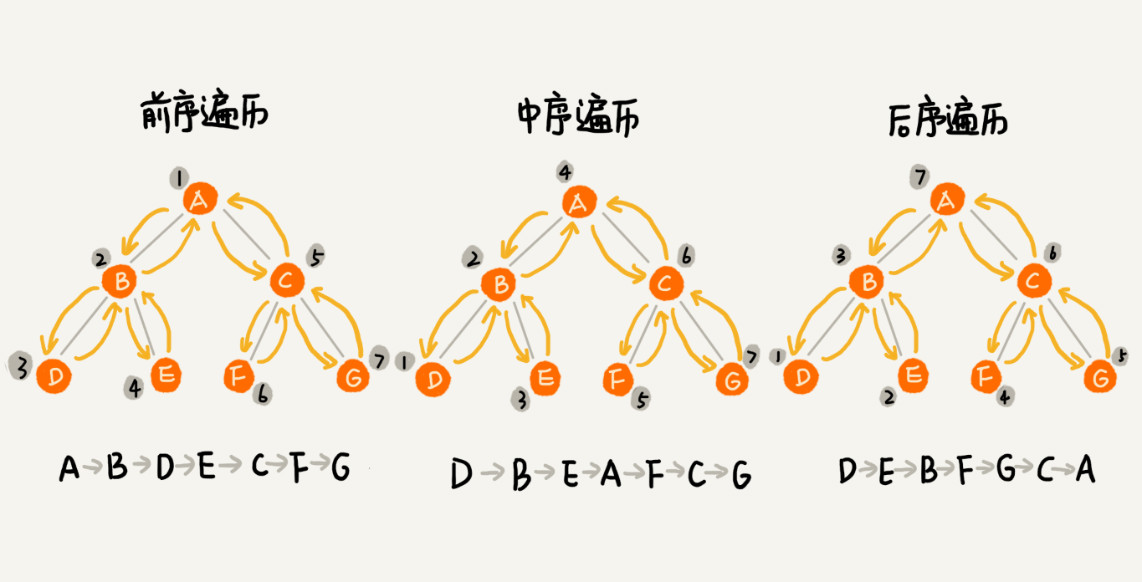


如果某棵二叉树是一棵完全二叉树，那用数组存储无疑是最节省内存的一种方式，仅浪费一个下标为0的存储位置。因为数组的存储方式并不需要像链式存储法那样，要存储额外的左右子节点的指针。这也是为什么完全二叉树要求最后一层的子节点都靠左的原因。

**五. 二叉树遍历**

经典的方法有三种，前序遍历、中序遍历和后序遍历。

* 前序遍历是指，对于树中的任意节点来说，先打印这个节点，然后再打印它的左子树，最后打印它的右子树。
* 中序遍历是指，对于树中的任意节点来说，先打印它的左子树，然后再打印它本身，最后打印它的右子树。
* 后序遍历是指，对于树中的任意节点来说，先打印它的左子树，然后再打印它的右子树，最后打印这个节点本身。



* 按层次遍历：借助队列思想，从上往下从左到右按层次遍历 ，O(n)。根节点先入队列，然后队列不空，取出对头元素，如果左子节点存在就入列队，否则什么也不做，右子节点同理。直到队列为空，则表示树层次遍历结束。树的层次遍历，其实也是一个广度优先的遍历算法。

Ps: 根据先序加中序或者后序加中序的结果可以确定一棵树

**六. 二叉查找树(Binary Search Tree)**

**七. 如何设计一个工业级的散列函数？**

**八. LRU 缓存淘汰算法**

**九. Redis 有序集合**

**十. Java LinkedHashMap**

**十一. 哈希算法**

***应用七：分布式存储***