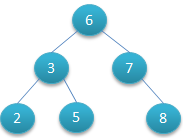
B-Tree是平衡多路查找树，B代表的是Balance，不是Binary。B+Tree是从最早的平衡二叉树演化来的。需要了解一些前提知识：二叉查找树Binary Search Tree、平衡二叉树AVL树。

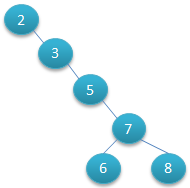
# Binary Search Tree

二叉查找树的性质是：左子树的键值小于根的键值，右子树的键值大于根的键值。



对该二叉树的节点进行查找发现深度为1的节点的查找次数为1，深度为2的查找次数为2，深度为n的节点的查找次数为n，因此其平均查找次数为 (1+2+2+3+3+3) / 6 = 2.3次。

但是二叉查找树可以任意构造，同样是2,3,5,6,7,8这6个数字也可以构造成下面的形式：

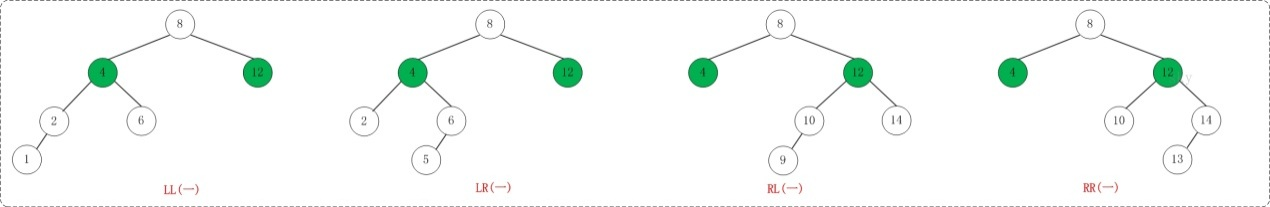


但是这棵二叉树的查询效率就低了。因此若想二叉树的查询效率尽可能高，需要这棵二叉树是平衡的，从而引出新的定义——平衡二叉树，也就是AVL树。

# AVL Tree

平衡二叉树是在符合二叉查找树的条件下，还满足任何节点的两个子树的高度最大差为1。

在AVL树中插入或者删除节点可能会导致AVL树失去平衡，需要通过一次旋转让AVL树恢复平衡。分为LL，RR，LR和RL。



LL（根节点左孩子的左孩子出了问题）：

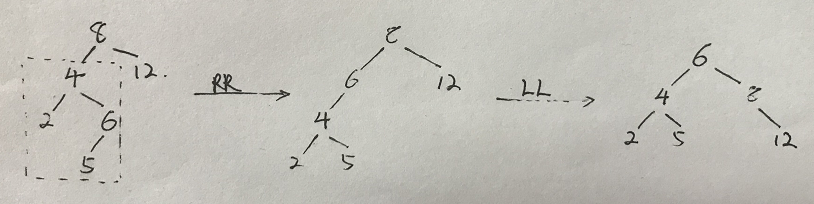
1. 根的左孩子作为新的根节点（4），即左孩子**右移**；
2. 新根节点的**右孩子**（6）作为原来根节点的左孩子。

RR（根节点右孩子的右孩子出了问题）：

1. 根的右孩子作为新的根节点（12），即右孩子**左移**；
2. 新根节点的**左孩子**（10）作为原来根节点的右孩子。

LR（根节点的左孩子的右孩子出了问题）

1. 对根节点的左孩子进行RR旋转；（根的左边部分）
2. 对根节点进行LL旋转。



RL（根节点的右孩子的左孩子出了问题）

1. 对根节点的右孩子进行LL旋转；
2. 对根节点进行RR旋转。

# B-Tree 平衡多路查找树

B-Tree是为磁盘等外存储设备设计的一种平衡查找树，许多数据库系统都一般使用B-Tree或者B-Tree的各种变形结构。比如在数据库系统中，B-Tree结构的数据可以让系统高效的找到数据所在的数据块，为了描述B-Tree，首先定义一条记录为一个二元组[key, data]，key为记录的的主键，data为一条记录除主键外的数据。一个**m阶**B-Tree有一下的特性：

1. 每个节点最多有m个孩子；
2. 除了根节点和叶子节点外，其它每个节点至少有Ceil(m/2)个孩子；
3. 若根节点不是叶子节点，则至少有2个孩子；
4. 所有叶子节点都在同一层，且不包含其它关键字信息；
5. 每个非终端节点包含n个关键字信息（P0,P1,…Pn, k1,…kn）；
6. 关键字的个数n满足：ceil(m/2)-1 <= n <= m-1 ；
7. ki(i=1,…n)为关键字，且关键字升序排序；
8. Pi(i=1,…n)为指向子树根节点的指针。P(i-1)指向的子树的所有节点关键字均小于ki，但都大于k(i-1)；

B-Tree中的每个节点根据实际情况可以包含大量的关键字信息和分支，下面是一个3阶的B-Tree。



# 参考

<https://www.cnblogs.com/vianzhang/p/7922426.html>