BST树，是二叉搜索树：

二叉查找树（Binary Search Tree），（又：[二叉搜索树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)，二叉排序树）它或者是一棵空树，或者是具有下列性质的[二叉树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91)： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为[二叉排序树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91)。

* 每个点的值d都不同，因此若插入时候某个点的值在树中存在，则插入失败
* 左边的d一定小于当前点，右边点一定大于当前点
* 中序遍历可得到有序序列

# AVL平衡树

是平衡的BST树，怎么个平衡法，通过每次插入删除节点，一旦发现不平衡了马上旋转操作让他平衡。要有变量记录平衡因子

## 查找算法：

同BST树，复杂度是稳定的logn插入结点的算法：

## 插入结点的算法：

总体过程同BST树那样，看当前节点now和要插入节点的值大小而决定向左还是向右；在插入完成时要经过多次旋转使树平衡，这个旋转时间是log(n)级别的，总复杂度是logn+(logn的旋转操作)

## 删除结点的算法：

总体和BST一样；但是由于删除使树不平衡，又要转转，在被删除的节点开始递归回溯的旋转树，类似插入操作。

总复杂度是logn+(logn的旋转操作)