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

# 树堆Treap

是平衡的BST树，怎么保持平衡，通过额外维护一个数rnd， 使树保持BST树性质同时，rnd满足堆的性质，即每个节点的rnd都要大于等于它父节点的rnd,rnd是一个随机数，随机值取保证了，树的节点树越多，退化成一个链的几率越小，treap可以近似的看成是平衡树，实际上不是像AVL那样严格的平衡树

## 查找算法：

同BST树

## 插入结点的算法：

先同BST树一样插入,之后进行递归的单旋转，目的是把不满足堆性质的新点旋转的满足堆性质，复杂度logn

## 删除结点的算法：

这个过程不和BST树那样，因为要满足堆的性质，是插入的逆过程，把要删除的点转到叶子结点，再直接删除

## 怎么旋转？

类似AVL树的方式，但是目的不同，AVL是为了尽量平衡，而这个是为了使不满足堆性质的rnd满足堆性质，

只有一类单旋转   
zig型单旋转 ，相反方向的也类似

