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

# 伸展树Splay

是平衡的BST树，怎么个平衡法，它是通过查找某元素后，对树旋转，使得树达到平衡，AVL树是在每次插入都要旋转

## 查找算法：

同BST树一样查找，但是查找后要旋转操作，不停的转，直到把查询的点转到根

## 插入结点的算法：

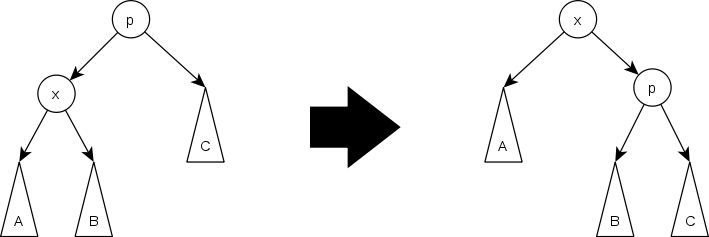
同BST树

## 删除结点的算法：

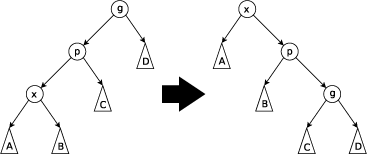
同BST树

## 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，而这个是为了把查询的节点搬到根节点，分为三类旋转，相反方向类似，没有给出图片。旋转的三类分别是单转，同向双转和异向双旋转。操作的时候尽可能用双旋转，效率高，单旋转是专门为了应对，当前要转的节点距离根节点距离1的情况(距离1指两个节点恰好父子关系)。  
1.zig型单旋转，只用于查询的点在根节点下方：



2.zigzig型同向双旋转：



3.zigzag型异向双旋转：

