# BST树(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" \t "_blank)，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" \t "_blank)： 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" \t "_blank)。

* 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

## 查找算法：

查找算法基础，插入删除也会用到，查找值是d0的点如何查？

设当前点now是根节点head,如果：

当前节点小于d0,向右找，

当前节点大于d0,向左找，

当前节点等于d0,说明找到了，

当前节点是空，穷途末路，说明不存在

## 插入结点的算法：

设当前点now是根节点head, 如果：

当前节点小于d0,向右找，

当前节点大于d0,向左找，

当前节点等于d0,根据BST树性质，不能有同样节点，插入错误

当前节点是空，可以插入，那就插入吧

## 删除结点的算法：

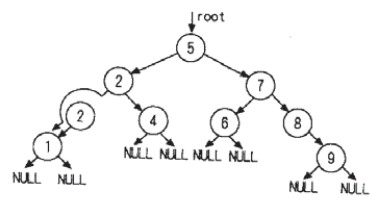
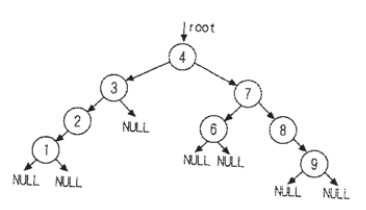
**情况1：**

对于左边或者右边有一个空的节点(或者左右都空)，只要直接删除再把下面连上即可

**情况2：**

如果两边都有节点：

寻找要删除的节点node的所有子节点中比它大一点(左子树最大值)，或者比它小一点的(右子树最小值)，和node交换，再删除那个节点(那个最值点一定是左右有一个是空的节点，也就是一定能变成情况1)。如下，删除根节点5

比如找比它小的点，只需要遍历node->left的每个最右的节点走，直到走不了。

# 扩展BST树：

增加属性cnt和sumcnt,cnt表示节点是d的个数，允许了树中存在相同值，sumcnt代表当前节点子树的数的个数（是每个子节点cnt的和而不是节点总个数）

struct Data{

int d=INF;

int cnt=1;//该节点出现次数

int sumcnt=0;//自己节点数的个数，会把相同的算进去

};

## 查找算法：

上面说了

## 插入结点的算法：

每次插入时，对于已经在树中存在的点得值d，让cnt++

并且不论怎样，每次插入后向上更新cumcnt的值

## 删除结点的算法：

每次删除时，对于目标某个节点的cnt大于1，让cnt—即可

并且不论怎样，每次删除后向上更新cumcnt的值

## 给定d查询它是第几小/大：

以第几小为例

调用查找算法返回now，设排名sum=1

如果给定d不存在返回-1，

若存在，向上一个个找，now在它父节点右边才计算

sum加上:now父节点cnt以及now父节点的左边的全部子节点cnt之和，

也就是：now.p.cnt+now.left.sumcnt+now.left.cnt

如此计算直到根节点，sum就是排名

## 查询排名第k小/大的数：

以第k小为例

树上每个节点的第几大可以通过一下情况确定：

* 根节点，k是左边所以节点cnt数+1
* 某个节点now是父的左边，

k是：往它上面，第一个是往右拐的点e的k值+e.cnt+now的左边所以节点cnt的和

* 某个节点now是父的右边，

k是：往它上面，父节点p的k值+p.cnt+now的左边所以节点cnt的和

## 给定数d，查询它的前驱后继：

### d在树上存在:

以**后继**为例

* 若右边不是空：沿着now.right一直往left找，找到空为止，那个节点是后继
* 若右边是空：那就向上找，若now.getFp()是左，就停止，否则继续向上找

### d在树上不存在:

以**后继**为例