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" \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

# FHQ\_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.

前面的介绍表示，他就是个treap，而fhq\_treap比普通treap好在不是通过旋转保持平衡。它有2个特殊操作，定义：

void split(Node \*r,int k,Node \*&tl,Node \*&tr)

把以r为根的树的按key的把比k小的节点组成的树分离，tl,tr是返回结果，分别代表较小和较大的两棵树，注意，拆分操作可以有不同规则，此规则最常用。

Node\* merge(Node \*a,Node \*b)

代表把以a和b为根节点的两棵树合并，返回合并后的根

下面介绍包括split和merge在内的fhq\_treap所有操作

### 分裂(split)

是fhq\_treap特殊操作，用递归思想，从根开始递归，设当前节点为r

如果r的key小于等于k:说明r本身和左边节点都小于k,把r赋值给tl,递归右边，修改tl的右边

如果r的key大于k:说明r本身和右边节点都大于k,把r赋值给tr,递归左边，修改tr的左边

分裂操作一般是按照k值分裂，tl是小于等于k的节点的树，tr是大于k的节点的树。但不同场合也可以用其他方法，可以写代码时传递比较函数的指针，来应付多种场景。

### 合并(merge)

Node\* merge(Node \*a,Node \*b)

合并是按照rnd合并，要求a,b本身就是2个treap,且a.key<b.key,

此时，我们只需要考虑rnd进行合并即可，因为按照特定方法合并总能保证结果满足BST树和堆的性质，方法是：

1. 如果a和b某个为NULL，就返回非空的那个
2. 如果a.rnd>b.rnd则以b为根，b的右儿子不变，b的左儿子可能发生改变，递归左边，去比较b.son[0]和a
3. 如果a.rnd<b.rnd则以a为根，a的左儿子不变，a的右儿子可能发生改变，递归右边，去比较a.son[1]和b

### 查找

可以按照普通BST树那样查找，也可以用fhq\_treap特有方法查找：

带查找的值是k，就是调用分裂的方法split(root,k,tl,tr)

寻找tl所有节点的最大值，就是要查找的k,

寻找tl所有节点的最大值的方法是从tl起一直向右走到尽头

### 插入

对于插入权值为k的节点，调用分裂的方法split(root,k,tl,tr)

此时一定满足：tl.k<=k<=tr.k

只需要先node=merge(tl,k)再root=merge(node,tr)即可

这里注意，有可能插入后使得树存在相同k值的节点，如果处理时不允许树有相同k值节点，可以加一层过滤，分割root后，先查找tl的最大值是否为k,不相等再插入。但过滤相等元素操作要额外耗时，如果插入频繁而查找不频繁可以允许树中有权值重复的节点

### 删除

对于删除权值为k的节点，调用分裂的方法split(root,k,x,z)

此时一定满足：x.k<=k<=z.k

方法一：

寻找x为根的树的最大值即可，如果不等于k,则说明不存在，删除失败。

如果等于k：由于是x最大值节点，必定没有右儿子，删除后连接左侧即可。

方法二：

调用分裂的方法split(x,k-1,x,y)

此时y的树的节点一定全是权值为k的节点，合并y的左右儿子，再删除y即可

为什么成立？因为在数据为离散整数的情况下，split(x,k-1,x,y)相当于把x中小于等于k-1和大于k-1的节点分离，而x又正好全部节点小于等于k，所以夹逼准则，y的树的节点权值一定全都等于k

### 查询k的前驱后继

直接以k为参数分裂即可：

如果查询前驱，就按照k值分裂，tl是小于k的节点的树，查找tl树的最大值节点即可；

如果查询后继，就按照k值分裂，tr是大于k的节点的树，查找tr树的最小值节点即可；

这里注意：查询前驱的分裂方式比较函数是[](int a,int b)->bool{return a<b}

而查询后继的分裂方式比较函数是[](int a,int b)->bool{return a<=b}

### 查询数字k的排名第几小

调用分裂的方法split(root,k-1,x,y)

当于把x中小于等于k-1和大于k-1的节点分离，在数据为离散整数的情况下，直接返回x为根的树的节点总数+1即可。

* 要注意特判x为NULL的情况，此时排名是1
* 分裂时可以变通写法：split(x,k,x,y,cmp),其中cmp写比较函数，把严格小于k的节点过滤到x