红黑数也是二叉查找树，它是普通二叉查找树的优化版。因为普通的二叉查找树有个最坏情况，既是每个节点只有一个子节点，树的深度等于节点的个数。这样查找的效率就是O（n），为了解决这种情况，就有了红黑树，它其实是平衡二叉查找树的一个实现。

这种平衡二叉查找树的一个思路就是尽量减少树的深度，就是不要出现上文说的那个最坏情况。怎么做呢？就是在新增一个节点或者删除一个节点时，进行一个修复树的操作。这里的话用红黑树实现，先了解下红黑树的性质。

红黑树的五个性质：  
1）每个结点要么是红的，要么是黑的。  
2）根结点是黑的。  
3）每个叶结点，即空结点（NIL）是黑的。  
4）如果一个结点是红的，那么它的俩个儿子都是黑的。  
5）对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点。

接下来的插入与删除操作后的树不得违背上面五条性质，违背了就要进行修复。始终保持这五条性质，就不会出现最坏情况。

上网看了一下，找了一些博文来看，感觉挺麻烦的。而且自己也没怎么看懂。就想自己实现一个不同的。

因为我是通过数组下标访问的，即下标 \* 2就是节点的左节点，再 + 1就是右节点。基于这个来实现平衡二叉查找树。

首先插入的时候，我会在数组后面新增。然后调整新增节点与父节点以及兄弟节点的关系，如果新增的当前位置在左节点，但是比父节点大，就与父节点调整位置。当前位置在右节点，但是比父节点小，也与父节点调整位置。调整位置后，再跟兄弟节点比较，比如说从左节点上来的，就跟右节点比较。从右节点上来的，就跟左节点比较。调整完后，重复刚才与父节点判断的步骤。判断完后，再与剩下的为比较过的节点判断（从右节点上来的就与左节点判断，从左节点上来的就与右节点判断），不合适就换位置，直到合适为止。这样先跟父节点比较，再跟兄弟节点比较，跟父节点比较，再跟兄弟节点比较。。。。。。。。。这样循环往复，直到合适为止。

删除的话，比较麻烦，也很耗时。最笨的方法就是把数组最后一个数覆盖那个要删除的数，然后再重新建立二叉查找树。删除的时间复杂度为O（n / 2）