

二叉排序树

- 对于任意的根节点
 - 左子树的值 $<$ 根节点的值
 - 右子树的值 $>$ 根节点的值
- 类似中序遍历的遍历方式，因此用中序遍历遍历出来一定是有序的

插入

与根节点比较，判断要插入到左子树还是右子树

删除


1. 删除叶子节点
 - 直接删除，让父节点指针指向空地址
2. 删除出度为1的节点
 - 出度为1的节点只有一个子节点
 - 交换子节点和根节点的位置
 - 删除根节点(此时根节点变成叶子节点)
3. 删除出度为2的节点
 - 当前节点既有左子树又有右子树
 - 当前节点的前驱(中序遍历)是左子树中的最大值,后继是右子树中的最小值
 - 前驱是左子树中最右边的节点，因此一定没有右子树，同理，后继也一定没有左子树
 - 用前驱和根节点交换，将问题转化为删除度为1的节点

AVL树(平衡二叉排序树)

- 左右子树的高度差不会超过1，也就是说树不会退化成链表
- 其它与二叉排序树性质相同

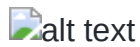
左旋与右旋

左旋

 alt text

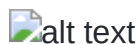
- K3的左子树的值比K1的值大
- 左旋后将K3的左子树变为K1的右子树

右旋



- K2的右子树的值比K1的值小
- 右旋后将K2的右子树变为K1的左子树

失衡类型



LL型

- 站在K1节点看左子树比右子树更高
- K1节点左子树的左子树比右子树更高
- 用一个右旋

LR型

- 站在K1节点看左子树比右子树更高
- K1节点左子树的右子树比左子树更高
- 先左旋再右旋

RL型

- 站在K1节点看右子树比左子树更高
- K1节点右子树的左子树比右子树更高
- 先右旋再左旋

RR型

- 站在K1节点看右子树比左子树更高
- K1节点左子树的右子树比左子树更高
- 用一个左旋


红黑树(平衡二叉排序树)

平衡条件


1. 每个节点非黑即红
 2. 根节点为黑色
 3. 叶节点(NIL)是黑色,是虚拟空节点, 而不是叶子节点
 4. 如果一个节点是红色, 那他的两个子节点都是黑色的
 5. 从根节点出发到叶子节点(虚拟空节点)路径上, 黑色节点数量相同!!!
- 红黑树中, 最长路径是最短路径的两倍
 - 新插入的节点是红色节点(插入黑色必定失衡)
-


只能以确定颜色的节点去进行平衡调整!!!

插入

 alt text


- 20与18发生冲突时, 叔父节点是红色
 - 此时把1改成黑, 15改成红, 20改成黑(黑红红->红黑黑)
-

 alt text


- 叔父节点是黑色
 - 按照红色节点的位置进行分类, 即LL,LR,RL,RR(AVL树), 如图为LL型
 - 根据分得的种类进行左旋或右旋, 再对根节点及其左右节点使用红色上浮或红色下沉的方法调整颜色, 如图  alt text **调整完后, 根节点要设置成黑色**(根节点颜色的变化不改变路径上黑节点数量)
-


删除

- 直接删除度为0的红色节点(不存在度为1的红色节点)
 - 度为1的黑色节点的子孩子只可能是红色
 - 红提升变黑, 删原黑
 - 删除度为0的黑色节点
 - 将对应路径上的NIL标记为双重黑
 - 删除度为2的节点可以转化为删除度为1的节点或删除度为0的节点的情况
-


 alt text

- 双重黑节点的兄弟节点是黑色，且其两个子节点都为黑色
 - 双重黑节点的父节点加上一重黑色(是红则变黑，是黑则变双重黑)。即若父节点是黑色，则需要继续进行调整，因为此时这颗子树无法调整至平衡
 - 双重黑节点及其兄弟节点各减去一重黑色
-


 alt text

 alt text


- 双重黑节点的兄弟节点是黑色，且其右节点是红色(RR)
 - 抓住双重黑节点的父节点进行左旋
 - 原根节点和双重黑节点的兄弟节点的右节点改为黑色
 - 左旋后的新根节点变为原根节点的颜色
-

 alt text

- 此种情况可以抓住兄弟节点进行右旋，转变为RR情况

 alt text

- 右旋后的新根节点变成黑色
 - 原根结点变成红色
 - 接着变成RR情况
-

- 双重黑节点的兄弟节点是红色  alt text
- 将红色节点调整到根节点，并将其变成黑色
- 将原根节点变成红色(因为其两个子节点都是确定的黑色)
- 双重黑节点的兄弟节点是黑色，是前面所提到的情况

B-树

一颗m阶B树，需要满足下列特性：


1. 树中每个节点，最多含有m棵子树
2. 若根节点不是叶子节点，则至少有2棵子树

3. 除根节点之外的所有非终端节点至少有 $m/2$ 棵子树
4. 如果一个节点有 $n-1$ 个关键字，则该结点有 n 个分支，且这 $n-1$ 个关键字按照递增顺序排列
5. 每个节点的结构为： $(n, A_0, K_1, K_2, A_2, \dots, K_n, A_n)$
6. 非根节点中关键字个数 n ，满足： $\lceil m/2 \rceil - 1 \leq n \leq m - 1$
7. 所有叶子节点在同一层

插入

将元素插入到终端节点处 当关键字超过上限，需要进行插入调整 **核心操作：节点分裂**

插入调整

 alt text

- 将关键字数量超过上限的节点从中间提取出一个关键字提升为父节点
- 其余的关键字为父节点的子树