自平衡二叉查找树

二叉查找树(BST)

在二叉树中查找一个节点的复杂度是O(n)，我们可以构建二叉查找树来改善查询时间。

特征

在二叉查找树中每一个节点的左子树中所有结点的值都小于该结点的值，右子树中所有结点的值都大于该结点的值。

查询

根据特征我们自然可以得出查找的方式。如果二叉树分布比较“像一棵树”我们可以接近理想状态的O(logn),但如果树退化成直线复杂度将达到最坏情况的O(n)。

插入节点

设新节点为k,每次考虑的节点为p,点的值为f(x)。
1.若p为空，则k直接放在这个位置。
2.若f(k)=f(p),则用户试图重复插入一个值，根据要求处理。
3.若f(k)<f(p),则k一定在p的左子树中，将p的左孩子视作p。反之则k一定在p的右子树中，将p的右孩子视作p。回到1。

删除节点

设该节点为p,左孩子为l,右孩子为r。 1.若无r,则l代替p;反之若无l,则r代替p。
2.r无左孩子,则r代替p,将l作为r的左孩子。
3.r有左孩子，则r的左子树中左下的最深的节点，即r的左子树中值最小的点，代替p。l。r不变。

自平衡二叉查找树

如果树中节点的数量为 n，则一棵满足O($log_2{n}$) 渐进运行时间的树的高度应接近于比$log_2{n}$小的最大整数。
对于一棵二叉查找树，往往不能事先知道所有的数据从而并不能保证树的形状合适，例如在不断输入数据并且不断查询的情况下。
如果我们能实时调整树使其满足最高效率的形状就好了。为此我们有自平衡二叉查找树。
有多种自平衡结构，如AVL树、红黑树、2-3树、2-3-4树、伸展树、B树等等。

红黑树

特征

首先是节点具有颜色：红、黑二色；其次是存在一种NIL节点，作为伪叶子节点连接着内点。 满足性质：

1. 根节点是黑色
2. NIL节点是黑色
3. 如果节点的颜色是红色，则其子节点均为黑色
4. 从任一节点到其后代任一叶子节点的路径上的黑色节点的数量相同

插入

按照BST的插入方式在计算出的位置插入一个红节点。

删除

AVL树

特征

对于每一个节点，其左右子树的高度差不超过1。

插入

按照二叉查找树

删除

未完待续..