**RANK-BALANCED TREE 作业2**

计科三班 1140310323 蒲毅

平衡二叉树在对数据结构的查找，插入，删除中有着平均非常好的复杂度，而对于不同种类的平衡二叉树又有一些差别，有的是从时间复杂度上的差别，有的是在功能上的差别。

**常见的平衡二叉树：**

红黑树

红黑树是一种自平衡二叉查找树，是在计算机科学中用到的一种数据结构，典型的用途是实现关联数组。它是在1972年由Rudolf Bayer发明的，他称之为"对称二叉B树"，它现代的名字是在 Leo J. Guibas 和 Robert Sedgewick 于1978年写的一篇论文中获得的。它是复杂的，但它的操作有着良好的最坏情况运行时间，并且在实践中是高效的: 它可以在O(log n)时间内做查找，插入和删除，这里的n是树中元素的数目。

AVL

AVL是最先发明的自平衡二叉查找树算法。在AVL中任何节点的两个儿子子树的高度最大差别为一，所以它也被称为高度平衡树，n个结点的AVL树最大深度约1.44log2n。查找、插入和删除在平均和最坏情况下都是O（log n）。增加和删除可能需要通过一次或多次树旋转来重新平衡这个树。

伸展树

伸展树（Splay Tree）是一种二叉排序树，它能在O(log n)内完成插入、查找和删除操作。它由Daniel Sleator和Robert Tarjan创造。它的优势在于不需要记录用于平衡树的[冗余信息](http://baike.baidu.com/view/1695291.htm)。在伸展树上的一般操作都基于伸展操作。

SBT

Size Balanced Tree（简称SBT）是一自平衡二叉查找树，是在计算机科学中用到的一种数据结构。它是由中国广东[中山纪念中学](http://baike.baidu.com/view/93561.htm)的陈启峰发明的。陈启峰于2006年底完成论文《Size Balanced Tree》，并在2007年的[全国青少年信息学奥林匹克竞赛](http://baike.baidu.com/view/553911.htm)冬令营中发表。由于SBT的拼写很容易找到中文谐音，它常被中国的信息学竞赛选手和[ACM/ICPC](http://baike.baidu.com/view/94274.htm)选手们戏称为“傻[B树](http://baike.baidu.com/view/298408.htm)”、“Super BT”等。相比红黑树、AVL树等自平衡二叉查找树，SBT更易于实现。据陈启峰在论文中称，SBT是“目前为止速度最快的高级二叉搜索树”。SBT能在O(log n)的时间内完成所有[二叉搜索树](http://baike.baidu.com/view/389453.htm)（BST）的相关操作，而与普通二叉搜索树相比，SBT仅仅加入了简洁的核心操作Maintain。由于SBT赖以保持平衡的是size域而不是其他“无用”的域，它可以很方便地实现动态顺序统计中的[select](http://baike.baidu.com/view/621719.htm)和[rank](http://baike.baidu.com/view/1604393.htm)操作。

树堆（Treap）

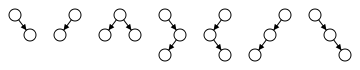
是有一个随机附加域满足[堆](https://zh.wikipedia.org/wiki/%E5%A0%86)的性质的[二叉搜索树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)，其结构相当于以随机数据插入的[二叉搜索树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)。其基本操作的期望[时间复杂度](https://zh.wikipedia.org/wiki/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6)为O(log n)。相对于其他的[平衡二叉搜索树](https://zh.wikipedia.org/wiki/%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A8%B9)，Treap的特点是实现简单，且能基本实现随机平衡的结构。

Treap=[Tree](https://zh.wikipedia.org/wiki/%E6%A0%91_(%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84))+[Heap](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E5%A0%86)。Treap本身是一棵[二叉搜索树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)，它的左子树和右子树也分别是一个Treap，和一般的[二叉搜索树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)不同的是，Treap纪录一个额外的数据，就是优先级。Treap在以关键码构成[二叉搜索树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%90%9C%E7%B4%A2%E6%A0%91)的同时，还满足[堆](https://zh.wikipedia.org/wiki/%E5%A0%86)的性质。Treap维护堆性质的方法用到了旋转，只需要两种旋转，编程复杂度比[Splay](https://zh.wikipedia.org/wiki/Splay)要小一些。

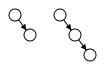
AA树

AA 树在[计算机科学](https://zh.wikipedia.org/wiki/%E9%9B%BB%E8%85%A6%E7%A7%91%E5%AD%B8)一种形式的[自平衡二叉查找树](https://zh.wikipedia.org/wiki/%E8%87%AA%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%9F%A5%E6%89%BE%E6%A0%91)用于高效存储和检索序数据。 AA 树的名称是由它的发明者[Arne Andersson](https://zh.wikipedia.org/w/index.php?title=Arne_Andersson&action=edit&redlink=1)而来。

AA树是[红黑树](https://zh.wikipedia.org/wiki/%E7%B4%85%E9%BB%91%E6%A8%B9)的一种变种，是Arne Andersson教授在1993年年在他的论文"Balanced search trees made simple"中介绍，设计的目的是减少[红黑树](https://zh.wikipedia.org/wiki/%E7%B4%85%E9%BB%91%E6%A8%B9)考虑的不同情况，区别于红黑树的是，AA树的红节点只能作为右叶子。换句话说，没有红节点可以是一个左子儿。这导致代替[2-3-4树](https://zh.wikipedia.org/wiki/2-3-4%E6%A8%B9)，从而大大简化了维护[2-3树](https://zh.wikipedia.org/wiki/2-3%E6%A0%91)的模拟。维护[红黑树](https://zh.wikipedia.org/wiki/%E7%B4%85%E9%BB%91%E6%A8%B9)的平衡需要考虑7种不同的情况:

[](https://zh.wikipedia.org/wiki/File:Red_Black_Shape_Cases.svg)

因为AA树有严格的条件(红节点只能为右节点)，故只需考虑2种情形:

[](https://zh.wikipedia.org/wiki/File:AA_Tree_Shape_Cases.svg)

平衡一颗[红黑树](https://zh.wikipedia.org/wiki/%E7%B4%85%E9%BB%91%E6%A8%B9)需要记录其颜色，而AA树是在每个节点记录其"level"这相当于红黑树节点的黑高度

1. 所有叶节点的level都是1
2. 每个左孩子的level恰好为其父亲的level减一
3. 每个右孩子的level等于其父亲的level或为其父亲的level减一
4. 每个右孙子的level严格小于其祖父节点的level
5. 每一个level大于1的节点有两个子节点

两个level相同的点之间的边水平边，也就是[红黑树](https://zh.wikipedia.org/wiki/%E7%B4%85%E9%BB%91%E6%A8%B9)上的红边。往右的水平边是允许的，但不可连续(红黑树性质)；不能有向左的水平边(AA树性质)。因为AA树的条件比红黑树严格，所以重新平衡一颗AA树会比重新平衡一颗红黑树容易。

插入和删除会让AA树变的不平衡(即违反它的性质)。恢复平衡只需两种操作:"skew"和"split". Skew是一个右旋转使得子树中向左的水平边变成向右的水平边；Split是一个左旋并增加子树根节点的level(请看范例)使得连续向右的水平边消失。平衡插入和删除操作的实现是由skew及split决定是否旋转，而不是在主程式中判断。