**RANK-BALANCED TREE 作业四**

计科三班 1140310323 蒲毅

### 本篇论文的局限

本篇论文的所讨论的问题都是基于平衡二叉树的问题，包括了新的rank-balanced tree 的理论体系，以及提出wavl并证明其高效性。本文的局限也在于此，虽然这一理论分类了平衡二叉树使其定义上变得统一，复杂度分析上更简化，也稍微提高了平衡二叉树的效率，但始终没有解决平衡二叉树本质上的缺点，就是平衡树的高度最好情况下也是 log2n，其中n是平衡二叉树的节点的个数。平衡树的在计算机领域的使用主要是在数据库中索引的建立上，索引是庞大的，往往不能直接储存在内存中，所以引索也同样是储存在外存之中的。我们还知道外存的访问消耗的时间远远大于内存访问的时间消耗，而对于一个引索，读取磁盘的次数是和引索的高度成正比的，也正因为如此，现在的数据库中所使用的也不是红黑树而是B树B+树之类的多叉平衡树，因为相比之下平衡多叉树的高度远远小于平衡二叉树的高度，所以减少了外存访问的次数，增加了引索访问的效率。

### 基于本篇论文的想法

本篇论文用rank理论统一定义了常见的平衡二叉树，我们也可以用rank类似的定义平衡多叉树。比如三叉树的情况，需要注意的是因为多叉树有多个指向其孩子节点的指针，所以我们每个节点需要储存不止一个关键字来划分指向孩子的指针。

(1)：所有空节点的rank为0

(2)：叶子节点的rank为1

(3)：非叶子结点的rank为其所有子树的rank的最大值

(4)：所有节点的子树的rank差为0或1

(5)：子树数目不足3的节点尽量在树的底层

从如上的定义我们可以知道，所有叶子节点的高度差为0或1。每次查询的复杂度为log2n到log3n。并且相对于平衡二叉树，这种平衡多叉树有更低的高度。

插入操作：插入的节点会在树的底层，如果破坏了树的平衡性，我们自底向上的维护失去平衡的子树。

删除操作：删除节点可能会影响树的平衡性，如果树失去平衡时，我们从失去平衡的点向下维护树的平衡性。

在树失去平衡之后，多叉平衡树调整的复杂度远远高于二叉平衡树，下面简单讨论了简单的减少多叉平衡树调整平衡的出现次数。

### 多叉平衡树插入和删除的优化方案

对于一个平衡多叉树的插入和删除操作，相比于二叉树来说消耗是非常大的。

**删除操作**：

我们可以通过给标记节点设置标记的方法来对节点进行删除操作，这样就会避免频繁的调整树的平衡带来的时间消耗。

**插入操作：**

我们可以用删除操作同样的思想，就是在建树的时候，为树中加入一些空节点，这样在插入的时候我们就不需要频繁的改变树的高度，减少其带来的时间复杂度的消耗。如果发现当前子树中没有空节点可供使用时，尽量在其兄弟之间进行调整来得到课存放的空间，再考虑重新增加树的高度。

对于什么时候空节点过多需要调整树，建树时又应该留多少空节点则可以根据实际的需要来确定。

例如：预计的操作插入较多，而删除的可能性比较小，那么新建树时空节点的数量应该多一些，并且没有必要因为空节点过多而调整树；预计的改动可能较大，那么就应该确定一个需要缩小树高度的对应空节点的值。

通过这样的方法，我们就避免了二叉树磁盘访问过多，而多叉树调整平衡的时间消耗过大的问题。

虽然空节点过多时会带来额外的时间消耗，但是这点时间消耗是可以接受的，因为额外的时间消耗是对数的增长率增加的，并且对数的底数可以控制得非常大，所以额外的时间消耗是非常小的。