**RANK-BALANCED TREE 论文概述**

计科三班 1140310323 蒲毅

解决的问题:

二叉搜索树是一个能够在最坏log(n) 的复杂度进行插入，查找以及删除操作的数据结构，在计算机科学中有着非常广泛的应用。在AVL被发明之后，有着很多的平衡树的实现方式。他们的侧重点要么是实现上的简洁，要么是效率上的高效，或者说两者全部。在多种平衡树的实现形式上，最出名和常用的就是AVL 和 RBH(red-black tree)，本文定义了一种新的平衡树的实现方式WAVL(weak AVL)。对于AVL，他的最大高度是logφn ≈ 1.44log2n(对于一个含有n个数据节点，φ为黄金比例)，但是AVL在删除操作时，可能需要一个对数时间复杂度的调整来使AVL重新平衡；RBH的最大高度虽然是2log2n，但是对于一次删除操作，只需要一个常数时间复杂度的消耗来使RBH重新平衡。所以本文介绍的WAVL在只有插入的情况下拥有和AVL相同的高度上限，在有插入和删除的情况下不超过RBH的高度限制，并且删除操作是常数消耗，甚至操作次数少于RBH的删除的操作次数。所以在总的效率上优于AVL和RBH。

采用的思想和算法的描述：

在二叉查找树中的节点分为内部点(包含数据的节点)和外部点(不包含数据的节点，叶子结点)。

我们知道在AVL中，平衡的标志是对于所有的树中的节点的平衡因子小于2，这个平衡因子定义为当前结点左子树的高度和右子树的高度之差。但在WAVL中，我们不用高度来定义平衡因子，而用rank来定义平衡因子

1. 所有的外部点(不包含数据的节点)的rank为0
2. 一个rank为r的非根节点的父亲节点的rank 为r+1或r+2
3. 有两个外部点为孩子的内部点的rank为1

值得注意的是这个rank是节点高度的近似值。

查找操作：

和普通的二叉查找树没有区别。

插入操作：

当需要插入元素k时，查找到需要插入k的位置(为一个外部点)，然后用一个拥有两个外部点为孩子的内部点代替这个节点。然后自底向上的跟新当前结点父亲节点的rank值使之大于跟新之后的当前结点的rank值，直到满足以下三个条件的任意之一：

设当前结点rank为r

1. 跟新的根节点
2. 跟新到一个节点，他的父亲节点的rank为r+2
3. 跟新到一个节点，他的父亲节点的rank为r+1，但是父亲节点的另一个孩子节点的rank为r-1，在这种情况下，总可以通过最多两次旋转操作将树重新调整到平衡

删除操作：

如果需要删除的节点有两个为内部点的孩子，那么可以用过平衡树删除的常用手段将需要伤处的的节点置换到一个拥有一个外部点孩子和一个内部点孩子的节点处，再删除它，此时直接将他的孩子与他的父亲相连即可。然后再如同插入操作的调整至底向上的调整路径上节点的rank值，直到满足以上的三个条件的任意之一。

算法的复杂度的分析：

WAVL在没有删除操作的情况下，插入节点如果使树失去平衡，跟新的路径最大长度为logφn ≈ 1.44log2n(即树的最大高度为logφn ≈ 1.44log2n)；如果有删除操作存在，那么跟新的路径最大长度为2log2n 。并且在发现失衡之后，可以在做多两次旋转操作将树重新调整到平衡。所以WAVL拥有AVL以及RBH共同的优点。

应用：

平衡树在计算机的领域中有着广泛的应用，比如c++ stl 里面的set 和 map 都是用的RBH，如果能够优化平衡二叉树的性能，那么很多程序或者软件的效率都能够得到提升。