AVL Tree

林敬翊
信计 3210300367

2022/10/28

1 设计思路

因为我们的树是使用 AvlTree，所以可以通过自身的调节确保他们平衡以进行查找，所以复杂度为 $O(k + logn)$

分析，理论上我们改动了 insert 函数的情况，每层的时间复杂度为 $O(1)$，总的时间复杂度不超过 insert 自身的时间复杂度。

我们最后采用 PrintElement 操作，并通过打印时间计算时间复杂度。

2 AVL 理论

(1) 查找代价：AVL 查找效率最好，最坏情况都是 $O(logN)$ 数量级的。

(2) 插入代价：AVL 插入操作的代价仍然在 $O(logN)$ 级别上 (插入结点需要首先查找插入的位置)。

(3) 删除代价：AVL 每一次删除操作最多需要 $O(logN)$ 次旋转。因此，删除操作的时间复杂度为 $O(logN) + O(logN) = O(2logN)$

AVL 效率总结: 查找的时间复杂度维持在 $O(logN)$，不会出现最差情况

AVL 树在执行每个插入操作时，其时间复杂度在 $O(logN)$ 左右。

AVL 树在执行删除时代价稍大，执行每个删除操作的时间复杂度需要 $O(2logN)$。

3 数值结果分析

经过运算我们可以得到下图

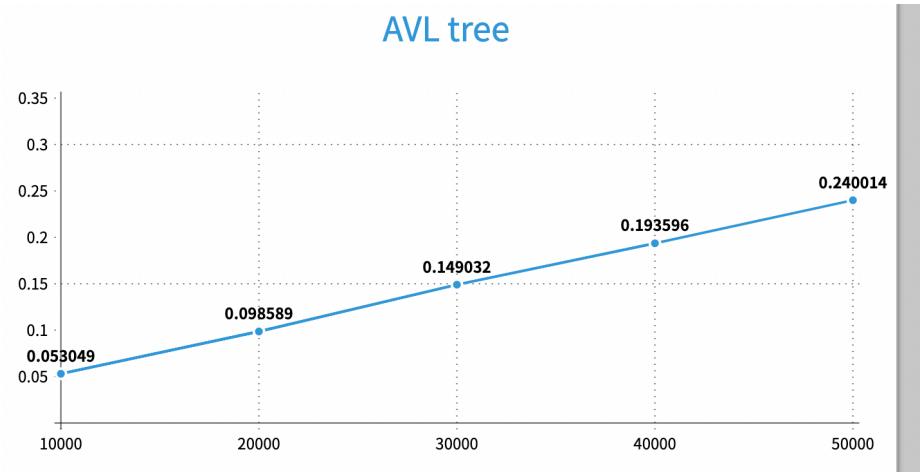

图 1: 测试结果

由测试结果可以发现函数在运行的时候是符合一次函数的关系，所以可知函数的时间复杂度收到 k 控制，为 $O(k)$