**CME 2001 Data Structures and Algorithms**

Assignment 2 Report

**Introduction**

Purpose of this assignment is to improve Avl-Tree to Augmented Avl-Tree and to compare two of them with test results. Avl-Tree is a self-balancing binary search tree. We want to add some speciality to Avl-Tree by storing summation of smaller nodes in current node. Augmented Avl-Tree and their nodes have some additional attributes but they have also same methods with original Avl-Tree. You may get output from methods which are getSum(), getSumSmaller(), getMax(), getMin() and print(). Also, you can insert any integer to these trees.

**Comparison Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **N** | **AVL-Tree** | | | **Augmented AVL-Tree** | | |
| 1.000 | 10.000 | 100.000 | 1.000 | 10.000 | 100.000 |
| **Running Time** | Insert | 5802037 nanoseconds | 544385979 nanoseconds | 91398963670 nanoseconds | 8594905 nanoseconds | 992212886 nanoseconds | 165619913952 nanoseconds |
| GetSum | 139510 nanoseconds | 176003 nanoseconds | 1227084 nanoseconds | 12925 nanoseconds | 14826 nanoseconds | 19015 nanoseconds |

This comparison table have been created by using same arraylist (same inputs). In addition system.nanoTime() has been used to calculate time elapsed. Elements of arraylist have been generated randomly 0 from 600,000 and this arraylist has not same numbers. By this way, we can observe and comment properly about comparison of Avl-Tree and Augmented Avl-Tree.

**Comparing Worst-Case Running Time Complexities**

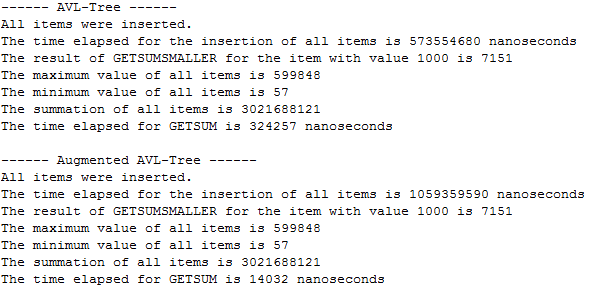
Worst-cases of insertion of the original Avl-Tree and Augmented Avl-Tree are respectively O(log n) and O(n \* log n) where n is the total number of nodes. Augmented Avl-Tree nodes stores summation of all smaller nodes. For this reason, every node must be updated on each insertion. It can be done with inorder treversal. An extra public attribute which add all items ascending and assign augmented nodes, is used to update augmented nodes and this attribute start from zero on every insertion. This make worst case O(n\*logn). So, original Avl-Tree make a difference in insert operation.

Worst-cases of getSum methods of the original Avl-Tree and Augmented Avl-Tree are respectively O(n) and O(log n) where n is total number of nodes. AVL-Tree add all items with inorder treversal but Augmented Avl-Tree does not need to visit all nodes. In Augmented Avl-Tree, only need to do is to use getMax() method. We should add biggest item’s augmented attribute and its key. So, we might obtain summation of all nodes with O(logn) time.

**Conculusion**

Original AVL-Tree and Augmented Avl-Tree have some advantage and disadvantage. It depends on to use them for what. Original AVL-Tree inserts all nodes faster than Augmented AVL-Tree. On the other hand, getSum() method of the Augmented Avl-Tree is much more faster than original AVL-Tree. I think, using Avl-Tree is more reasonable than Augmented since we might save time on insertion. It does not worth using Augment Avl-Tree, because we can gain further time from insertion operation in original Avl-Tree.

**Screenshot**



**References**

<http://blog.blackbam.at/2012/05/04/avl-tree-implementation-in-java/>

<https://en.wikipedia.org/wiki/AVL_tree>

Özgür Hepsağ

2014510043