|  |  |
| --- | --- |
| A picture of a winding road and trees  Assıgnment 2  Augmented AVL Tree | Abstract  AVL (Adel’son-Vel’skii and Landis) tree is a binary search tree such that for any node in the tree, the height of the left and right sub-trees can differ by at most 1. Augmented AVL tree is the updated version of AVL tree which the tree nodes holds the sum of all keys that are smaller than the current node key. In this report, the comparison of AVL Tree and Augmented AVL Tree insertion and GETSUM methods running time comparison has been done. Also, the worst-case running times for each methods has been compared for both AVL Tree’s. Finally, the efficient AVL Tree data structure was decided.  Gul Eda Aydemir 2015510013  CME 2001- DATA STRUCTURE |

Contents of Report

**Comparison 2**

a) Comparison Table Between AVL Tree vs Augmented AVL Tree2

b) Running Time Comparison Between AVL Tree vs Augmented AVL Tree2

c) Worst-Case Running Time Complexity Comparison Between AVL Tree vs Augmented AVL Tree3

**Screen Shots 4**

**Conclusion & References5**

Efficient Augmented AVL Tree & Reason5

References5

# COMPARISON

1. **Comparison Table Between AVL Tree vs Augmented AVL Tree**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | |  | **AVL-­‐‑Tree** | | **Augmented AVL-­‐‑Tree** | | |
| 100 | 1,000 | 10,000 | 100 | 1,000 | 10,000 |
| **Running**  **Time** | **INSERT** | 651745 | 8816523 | 1031639412 | 2759398 | 21846810 | 2144479614 |
| **GETSUM** | 68253 | 112900 | 515751 | 34383 | 37976 | 382836 |

*Table1. Comparison Table (~nanosecond)*

1. **Running Time Comparison Between AVL Tree vs Augmented AVL Tree**

Table 1 shows us the running time for both of insert and GETSUM methods between AVL Tree and Augmented AVL Tree for each node size. Therefore, as the running times shown above in Table 1, Augmented AVL Tree insertion time is bigger than the AVL Tree insertion for every size because of updating Augmented AVL tree for each insertion. On the other hand, Augmented AVL Tree GETSUM method time is smaller than AVL Tree for every size because the nodes are allready holds the sum of all items that are smaller than the current node item. If we get that value and add the key value to it we find the sum of all items very fast. We don’t need to walk around all the tree for finding the sum of all items. That’s why, the time required for GETSUM method is smaller in Augmented AVL Tree.

1. **Worst-Case Running Time Complexity Comparison Between AVL Tree vs Augmented AVL Tree**

INSERT AVL TREE: The rotation operations (left and right rotate) take constant time as only few pointers are being changed there. Updating the height and getting the balance factor also take constant time. So the time complexity of AVL insert remains same as BST insert which is O(h) where h is height of the tree. Since AVL tree is balanced, the height is O(Logn). So time complexity of AVL insert is O(Logn).

INSERT AUGMENTED AVL TREE:The rotation operations (left and right rotate) take constant time as only few pointers are being changed there. Updating the height and getting the balance factor also take constant time. Updating the smaller node in Augmented AVL tree, needs to travel the nodes by inorder traversal algorithm way. So it takes O(n) time. So the time complexity of Augmented AVL Tree is O(h+n) and because of Augmented AVL Tree is balanced, therefore the time complexity become O(logn+n).

GETSUM AVL TREE:The GETSUM method for AVL tree is the algorithm like the way of inorder traversal. So the time complexity is O(n).

GETSUM AUGMENTED AVL TREE: The GETSUM method for Augmented AVL tree is the simply the returning GETMAX’s node that which holds the sum of smaller than the current node. So we can find the sum by simply adding node key to this value. So the time complexity is required for this method is O(logn).

Thats why we can say that ;

Insert : AVL TREE O(logn) < AUGMENTED AVL TREE O(logn +n )

GetSum : AVL TREE O(n) > AUGMENTED AVL TREE O(logn)

# SCREEN SHOTS

# 

**////////////////////////////////////////**100

------ AVL-Tree ------

All items were inserted.

The time elapsed for the instertion of all items is 651745 nanoseconds

The result of GETSUMSMALLER for the item with value 100 is 5017

The minimum value of all items is:0

The maximum value of all items is:100

The summation of all items is: 5017

The time elapsed for GETSUM is 68253 nanoseconds

------ Augmented AVL-tree ------

All items were inserted.

The time elapsed for the instertion of all items is 2759398 nanoseconds

The result of GETSUMSMALLER for the item with value 100 is 4917

The minimum value of all items is:0

The maximum value of all items is:100

The summation of all items is: 5017

The time elapsed for GETSUM is 34383 nanoseconds

///////////////////////////////////////////////1000

------ AVL-Tree ------

All items were inserted.

The time elapsed for the instertion of all items is 8816523 nanoseconds

The result of GETSUMSMALLER for the item with value 1000 is 500238

The minimum value of all items is:0

The maximum value of all items is:1000

The summation of all items is: 500238

The time elapsed for GETSUM is 112900 nanoseconds

------ Augmented AVL-tree ------

All items were inserted.

The time elapsed for the instertion of all items is 21846810 nanoseconds

The result of GETSUMSMALLER for the item with value 1000 is 499238

The minimum value of all items is:0

The maximum value of all items is:1000

The summation of all items is: 500238

The time elapsed for GETSUM is 37976 nanoseconds

/////////////////////////////////////////////10000

------ AVL-Tree ------

All items were inserted.

The time elapsed for the instertion of all items is 1031639412 nanoseconds

The result of GETSUMSMALLER for the item with value 10000 is 49998226

The minimum value of all items is:0

The maximum value of all items is:10000

The summation of all items is: 49998226

The time elapsed for GETSUM is 515751 nanoseconds

------ Augmented AVL-tree ------

All items were inserted.

The time elapsed for the instertion of all items is 2144479614 nanoseconds

The result of GETSUMSMALLER for the item with value 10000 is 49988226

The minimum value of all items is:0

The maximum value of all items is:10000

The summation of all items is: 49998226

The time elapsed for GETSUM is 382836 nanoseconds

# CONCLUSION & REFERENCES

1. Efficient Augmented AVL Tree & Reason

Augmented AVL Tree is efficient to use if we need to get sum of all items for large size. On the other hand, Augmented AVL Tree is not efficient because of large time running for Insertion method.

Although, If we do not have to use the method of AVL tree, there is more efficient way to hold the sum of smaller items.

1. References

I got help from the internet during my homework.

https://rosettacode.org/wiki/AVL\_tree#Java

I got these methods from this site:

* Node find(int x, Node n);
* Node rotateLeft (Node n);
* Node rotateRight (Node n);
* Node rotateLeftThenRight(Node n);
* Node rotateRightThenLeft(Node n);
* Height(Node n);
* setBalance(Node n);
* printBalance(Node n);