CME 2001 Assignment-2 (Augmented AVL-Tree)

Due date: 20.12.2016, 23:59. Late submissions aren't allowed.

Control date: Control date will be announced later. Research assistances will control your assignments in their offices. You will have 5 minutes to show your assignments.

Parallelism Control: The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in lectures.

Please do not forget to bring your laptops and reports while coming to the assignment control!

Description: You are expected to implement a balanced binary search tree (specifically an AVL-Tree) which performs operations and queries displayed below. Then you are required to improve it by augmenting the tree structure to make some queries faster. Finally, compare them according to the computing time of queries. Project will be coded in Java.

Task - 1
Design and code AVL-Tree data structure with the following capabilities:

-INSERT: insert an input item with a given key (an integer number)
-GETSUMSMALLER: get summation of items smaller than a given input key
-GETMAX: get maximum of all items
-GETMIN: get minimum of all items
-GETSUM: get summation of all items in tree
-PRINT: display all items of the tree by applying in-order tree walk.

Task - 2
The program should run the methods in the following order for testing.

-Insert all elements
-Print the time elapsed for the insertion of all items
-Print the result of GETSUMSMALLER for the item with value 1000
-Print the maximum value
-Print the minimum value
-Print the summation of all items (GETSUM)
-Print the time elapsed during GETSUM

Task - 3
Improve the AVL-Tree class by augmenting the structure so that each node holds an additional field: the sum of all items (keys) that are smaller than the current node item. Note that you need to update this value on each insert operation.

The standard output of your program should appear as follows:

```
------ AVL-Tree ------
All items were inserted.
The time elapsed for the insertion of all items is 2,47563383 nanoseconds
The result of GETSUMSMALLER for the item with value 1000 is 23442
The maximum value of all items is 1987
The minimum value of all items is 2
The summation of all items is 221327
The time elapsed for GETSUM is 0,01563383 nanoseconds
```

```
------ Augmented AVL-Tree ------
All items were inserted.
The time elapsed for the insertion of all items is 3,15623873 nanoseconds
The result of GETSUMSMALLER for the item with value 1000 is 23442
The maximum value of all items is 1987
The minimum value of all items is 2
The summation of all items is 221327
The time elapsed for GETSUM is 0,00062383 nanoseconds
```

Task - 4

Compare the running times (in nanoseconds) of the INSERT and GETSUM methods for AVL-Tree and Augmented AVL-Tree with sizes of 1000, 10000, 100000 items; fill Table 1 accordingly. **Use System.nanoTime() to calculate the time elapsed**. You need to generate the items randomly (Generate random numbers between 0 and 2000). You should use the same input for testing both AVL-Tree and Augmented AVL-Tree for each size (i.e., 1000, 10000, 100000). If you get memory errors while inserting large amount of numbers (e.g., size of 100000), you can rescale input sizes e.g., 500, 5000, 50000.

		AVL-Tree			**Augmented AVL-Tree**		
	N	1,000	10,000	100,000	1,000	10,000	100,000
Running Time	**INSERT**						
	GETSUM						

Table 1. Comparison table

Task - 5

Prepare a scientific report in the light of the results you obtained from the program and compare the variations in the running time of the methods. Your report should include the Comparison table (Table 1) for running times. Then, compare worst-case running time complexity of the Augmented AVL-Tree with the original AVL-Tree for INSERT and GETSUM methods. Finally, comment on if augmented AVL-Tree data structure is a reasonable decision or not.

Grading Policy

Task	**Percentage**
AVL-Tree	% 40
Augmented AVL-Tree	% 20
Running Time	% 20
Report	% 20

Table 2. Grading Policy