Homework #5

Question 1 (2 pt.)

The height of a particular node within a binary tree can be recursively defined as the maximum height of its child nodes plus one. The height of a leaf node is 0. The height of a binary tree is the height of its root node.

Extend the implementation of the binary search tree given in class `Tree` on the support material with the following functions:

- A private function `int getHeightAux(Node *node)` returning the height of a node.
- A public function `int getHeight()` returning the height of the tree.

Extend the main program in order to obtain the height of a tree and print it. Upload a ZIP file called `q1.zip` with all files needed for the project to compile and run correctly.

Question 2 (4 pt.)

Write two versions of a main program, called `main-increasing.cc` and `main-random.cc`, respectively. Both take an integer value `size` as a command-line argument. They create `size` objects of type `Integer` and insert them into a binary search tree, where the object acts both as the key and the data (that is, the entire object acts as its own key).

The main program called `main-increasing.cc` inserts integers in increasing order, while the main program called `main-random.cc` inserts random values.

a) (1 pt.) Measure the execution time of both programs as the number of elements increases, and plot the results.

b) (1 pt.) Measure the height of the tree at the end of the execution of each program, using function `Tree::GetHeight()`, as implemented in the previous question, and plot the results.

c) (1 pt.) Justify the difference between both, and the correlation between the tree height and the performance.

d) (1 pt.) Upload a ZIP file named `q2.zip` containing all files needed to compile and run the project, including both main programs written for this question.

Question 3 (2 pt.)

Extend the binary tree interface with a pair of sub-tree rotation functions with the following interfaces:

- `void RotateRight(Node *node)`

- `void RotateLeft(Node *node)`

The rotation functions should operate as described in the behavior of the red-black trees (see support material). The functions take a node as a rotation edge, bringing that node one level down the tree, either left or right.

Question 4 (2 pt.)

Modify the insertion algorithm of the binary search tree to implement a simple and partial balancing technique. If the distance between the new node and the root is greater than the total number of nodes divided by 2, a rotation is performed over the root. This rotation will be to the right if the node was inserted on the left of the root, and to the left if it was inserted on the right of the root.

a) (1 pt.) Modify the code in function Tree::Insert() to implement this technique. You can invoke the rotation functions implemented for the previous question. Upload a ZIP file named q4.zip including all files needed to compile and run the project.

b) (1 pt.) Run the first main program from Question 2 (`main-increasing.cc`) where all tree nodes are inserted in order, using the new implementation for the insertion algorithm, and observe the evolution of the execution time and the tree height as the number of nodes increases. What do you observe? How does the new implementation help balancing the tree? Justify your results. You don't need to obtain plots now, just add a description on your report.