CS 3024 Home Assignment 2 (30 + 5 or more bonus points)

Main task (30 points).

Sakai folder "Classes" contains files **binary_tree.h** with the definition of class **btree** for binary search tree with **int** keys and data items and **binary_tree.cpp** file running tests on it.

Your task is to add to the class **btree** new member functions:
int smallest_key() returns the smallest **key** stored in the binary search tree, or 0 if tree is empty;
int largest_key() returns the largest **key** stored in the binary search tree, or 0 if tree is empty;
double average_data() returns the average of all **data items** stored in the tree, or 0 if tree is empty. This way we can receive the fractional part of average as well.

In all cases, when the binary search tree is empty, the member function returns the default value 0 and prints a message.

For the **smallest_key()** and **largest_key()** methods don't use the "brute force" algorithm (complete traversal of the whole tree). It is inefficient, and binary search tree provides a better way to do it. Use algorithm similar to one implemented in **get_predecessor()** member function. On the other side, **average_data()** method requires a complete tree traversal in order to visit all nodes and collect data values. This may be similar to the algorithm implemented by **print()** method.

Add these new public member functions to the modified **your_name_binary_tree.h** file.
Don't forget to adjust the comments (header comment and comments attached to your code). Clearly mark new/modified parts of the original source code with comments of the form //***** **my update** ****

Using **binary_tree.cpp** as a sample, create a file **your_name_HA2.cpp** with a **`main()`** function that performs the following tests.

a) Test for an empty tree.

b) Test for a tree containing just one node.

c) Test for a tree with a small number of random nodes (5 - 8 nodes).

d) Test for a tree with a small number of nodes (5 - 8 nodes) entered in ascending order. This can be done with **for** loop generating keys in ascending order with random data items.

e) Test for a tree with a small number of nodes (5 - 8 nodes) entered in descending order. This can be done with **for** loop generating keys in descending order with random data items.

Each test case should appear in your **main()** function separately and should have the following structure.

- A comment explaining the test case.
- Declare a variable of type **btree**.
- Populate the tree with key and data using **insert()** method. For the case c) random keys may occasionally duplicate, and the tree will have less nodes neither originally expected.
- Print the contents of tree using the **print()** or **print_vis()** method.
- Show the results of calling each of your new methods.

Note. Don't copy tests for existing member functions from the binary_tree.cpp, although you may earn 1 bonus point for each test case demonstrating an error in some of already given member functions. If you decide to proceed with this aspect, please comment properly your test cases in the main() body explaining what kind of erroneous behavior your test case exposes.

Bonus 5 points.

A *path length* between nodes N1 and N2 in the tree is the number of nodes, including N1 and N2, on the path between N1 and N2. Note that in a binary tree there always exists a single path between any pair of nodes.

Add new public function members:

int min_depth() returns the smallest path length from the root to a leaf;

int max_depth() returns the largest path length from the root to a leaf;

For an empty tree T both min_depth() and max_depth should return 0.

For a tree containing a single node, both min_depth() and max_depth() return 1.

Provide tests for this new functionality in the main() function covering all meaningful cases.

Note. The difference between max_depth() and min_depth() in fact reflects the degree of binary tree's balance – the bigger is the difference, the worse is the balance.

The header comments in your source files are mandatory, and should conatin at least the following:

Your name;
Date;
File name;
Short description of the file contents (tasks, purpose, etc.);
Platform on which it has been designed (Mac OS, Windows, etc.).

Submit the source code of your solution (files **your_name_binary_tree.h** and **your_name_HA2.cpp**) electronically to maugusto@nps.edu

The deadline is **Wednesday, November 29, 2017, midnight.**