Assignment 3, Part 1 of 2
Binary Search Tree Implementation

Instructor: Homeyra Pourmohammadali
MTE140 - Data Structures and Algorithms
Winter 2021 (Online)
University of Waterloo

Due: 11:00 PM, Wed, April 7th, 2021

Purpose of this assignment

In this assignment, you will practice your knowledge about **tree** by implementing a data type called **binary search tree**. The header file binary-search-tree.h, which is explained below, provides the structure of the BinarySearchTree class with declarations of member functions. Do not modify the signatures of any of these functions. You need to implement all of the public member functions listed in binary-search-tree.cpp. Note that the tree does not have to be balanced for this part of the assignment.

Instruction

Sign in to GitLab and verify that you have a project set up for your Assignment 3 (A3) at https://git.uwaterloo.ca/mte140-1211/a3/WATIAM_ID with the following files.

```
YOUR-WATIAM-ID
├── CMakeLists.txt
├── README.md
├── user.yml
├── binary-search-tree.cpp
├── binary-search-tree.h
├── avl-tree.cpp
├── avl-tree.h
└── test.cpp
```

For this part of assignment, you only need to modify binary-search-tree.cpp and user.yml. You can design your own test case and code in test.cpp. It is optional and we will not grade this file.

You can use the same procedures in Assignment 1.1 to pull, edit, build, commit, and push your repo.

Description

The details of the header file binary-search-tree.h are as follows:

DataType defines the kind of data that the tree will contain. Being public, it can be accessed directly as BinarySearchTree::DataType.

Member variables:

Node: This is a structure declaration. Node contains the following member function and variables:
Node(DataType newval): Sets the left and right children to NULL, and initializes val.
val: Value of the node.
left: Pointer to the left node.
right: Pointer to the right node.
avlBalance: As introduced in the class, the balance value of each node refers to the absolute value of difference in heights between the left and right subtrees of the node. You may implement updateNodeBalance function to update avlBalance. It is not necessary for Part 1 about binary search tree, but it may help you when you implement AVL tree in Part 2. Note that there are other methods to implement AVL tree properly without the need to use this variable, so it is optional.

root_: Pointer to the root node of the tree.

size_: Number of nodes in the tree.

Member functions:

BinarySearchTree(): Constructor to initialize an empty tree with no node.

~BinarySearchTree(): Deallocates the memory space allocated for the binary search tree nodes, if any.

int size() const: Returns the number of nodes in the tree.

DataType max() const: Returns the maximum value of a node in the tree among all the nodes. You can assume that this function will never be called on an empty tree.

DataType min() const: Returns the minimum value of a node in the tree among all the nodes. You can assume that this function will never be called on an empty tree.

unsigned int depth() const: Returns the maximum depth of the tree. A tree with only the root node has a depth of 0. You can assume that this function will never be called on an empty tree.

void print() const: You can print the tree in whatever order you prefer. However, methods such as in-order or level-order traversal could be the most useful for debugging. Note:

this function will not be used for grading purposes, so the printing format does not matter.

bool exists(DataType val) const: Returns true if a node with the value val exists in the tree; otherwise, it returns false.

Node* getRootNode(): Returns a pointer to the root node.

Node** getRootNodeAddress(): Returns the address of the root node pointer.

bool insert(DataType val): Inserts the value val into the tree as a new node. Returns false if val already exists in the tree, and true otherwise.

bool remove(DataType val): Removes the node with the value val from the tree. Returns true if successful, and false otherwise. Note: when the to-be-removed node has two child nodes, replace the value with the predecessor (rather than successor). This implementation will be different from the demo code used in the lecture, which used the successor.

void updateNodeBalance(Node* n): Optional. This function is not necessary for Part 1 about binary search tree, but it may help you when you implement AVL tree in Part 2. Given Node* n that is a pointer pointing to a node in the tree, this function calculates the balance value of Node* n and repeat this for all the ancestors of Node* n. Note that there are other methods to implement AVL tree properly without the need to implement this function, so it is optional.

Marking

We will try different inputs and check your output. We will only test your program with syntactically and semantically correct inputs.

Part 1 counts **60%** of Assignment 3, which is 60 points in total.

Your program runs and does not crash during the test: + 20
Passes Test Cases: + 5 each, in total of 40