Programming Assignment (PA) #3

Red-Black Tree

For this assignment you will implement a red-black tree. You will implement the RedBlackTree class from “RedBlackTree.h” in your source file “RedBlackTree.cpp”.

RedBlackTree.h contains an already defined Node class. You will use this node class to build your RedBlackTree, the Node class follows:

```
// Node class
// Contains already defined methods
// You may add helper methods if needed
class Node {
public:
    int val;
    int color;
    Node *parent;
    Node *left;
    Node *right;

    Node(int val) : val(val) {
        parent = left = right = nullptr;
        color = RED;
    }
    int getColor() { return color; }
    Node* getLeftChild() { return left; }
    Node* getRightChild() { return right; }
    Node* getParent() { return parent; }
    int getValue() { return val; };
};
```

A red-black tree is a binary search tree with nodes colored red and black in a way that satisfies the following properties:

Root Property: The root is black.
External Property: Every external node is black.
Red Property: The children of a red node are black.
Depth Property: All external nodes (leaves) have the same black depth, defined as the number of proper ancestors that are black.

For the add operation, you need to remedy a potential double red violation; For the remove operation, you need to remedy a potential double black violation. Please refer to the textbook and its code snippets to understand how this is done.

For the Red-Black tree class, you will implement four methods: add, remove, contains, and get. The first two methods modify the tree and might trigger rotations. The third method checks if the tree contains a value. The last method searches for a value within the tree and returns the node containing it. The RedBlackTree class follows:

```
// RedBlackTree class
class RedBlackTree {
private:
    Node* root;

public:
    // You must implement the following five methods

    // Default constructor for RedBlackTree
    RedBlackTree();

    //Inserts a value into the tree and performs the necessary balancing
    void add(int value);

    //Removes a value from the tree if present and performs the necessary balancing
    void remove(int value);

    //Returns true if the tree contains the specified value
    bool contains(int value);

    //Returns Node containing specified value
    Node* get(int value);

    // Helper Methods
    // You may remove these methods or add more if needed
    void leftRotate(Node* x);
    void rightRotate(Node* x);
    void fixRedRed(Node* x);
    void fixBlackBlack(Node* x);
};
```

Additional Details

- You may add code to the RedBlackTree.h header file if needed.
- Each method must be as efficient as possible. That is, a O(n) is unacceptable if the method could be written with O(log n) complexity.

- All source code files must have your name and class account number at the beginning of the file.
- Your code must not print anything.
- Your code should never crash, but must handle any/all error conditions gracefully.
- You must write code according to the header provided. You may add methods to the implementations if needed.
- Your code may generate unchecked warnings when compiled, but it must compile and run correctly on gradescope to receive any credit.

Testing Your Code

- A testing file “TestDriver.cpp” is provided.
- Use the makefile to compile your code.
    - To compile your code run the command “make” from a terminal while in your project directory.
    - To execute your code run the command “./RedBlackTree” from a terminal while in your project directory.
    - If you are using VSCode, the makefile tools extension can make compilation and execution with the makefile easier.

Submitting Your Code

- You will be submitting your assignment on gradescope.
- Only submit your RedBlackTree.cpp and RedBlackTree.h file.
- Gradescope will autograde your assignment and give you a score.
- Your comments and the readability of your code will be manually graded.