CS 341 – Fall 2024
Assignment #5 – Color Me Dawgs
Due: 11/22/2024

Red-Black Trees are self-balancing Binary Trees. It achieves this balancing by coloring the nodes of the tree to ensure that insertions and deletions are handled in a balanced sense. The really nice benefit of a Red-Black Tree is that it provides a worst-case guarantee on insertion, deletion, and search times. This is great for real-time applications with strict requirements. On top of that by ensuring balance – they are ideal for searching and provide a significant benefit over using a Linked (or Doubly Linked) List. *Take that Assignment #4!*

For this assignment you are tasked with creating a Red-Black Tree. Blue IV (our faithful CEO) has heard about such a data structure and wants to use it to model athletic roster numbers (here we assume that no double numbers are possible). Blue IV wants to know what the root node of our tree is after all insertions and which nodes are RED and which nodes are BLACK after its all said and done. He has given us a text file that contains athletic roster numbers (integers) and our job is to read-in this file and build a tree from it. He only wants us to deal with insertions and searches for this project – no deletions (*thank goodness!*).

Just a reminder (discussed in lecture) a Red-Black Tree has the following properties:

- A Red-Black Tree must be a type of Binary Search Tree
- Every node has a color either RED or BLACK
- The root of the tree is always BLACK
- When a node is added to the tree it begins its life as a RED node
- There are no two adjacent RED nodes
    - A RED node cannot have a RED parent or a RED child
- Every path from a node to any of its descendant (down to the `nullptr`) has the same number of BLACK nodes

Phase I – 11/6/2024

For Phase I of this program you should begin by implementing a driver program that reads in SPACE delimited numerical data from a text file (`data.txt`). Print out the data found in this file onto the console/terminal window as shown below:

```
3 18 7 10 22 8 15 29
```

Once you have completed this task you are ready for Phase II.

Phase II – 11/8/2024

For Phase II, using the code we wrote in class – create a Binary Search Tree (`BinarySearchTree.cpp/.h` and `TreeNode.cpp/.h`) and insert the data you read-in in Phase I to construct the tree. Print out the root node value of the tree along with the height of the tree in the console/terminal window as shown given the input below:

```
3 18 7 10 22 8 15 29
```

Your output would contain the following:

```
Root: 3
```

Once you have completed this task you are ready for Phase III – now the fun begins!

Phase III – 11/12/2024

Now we get down to business – in this phase, you will be responsible for creating a new class `RedBlackTree.cpp/.h`. This class will need to inherit from your `BinarySearchTree`. In this phase you will need to implement the following methods:

- `virtual void insert(int data);`
    - This method will override the insert method from the `BinarySearchTree` class and provide the ability to add color to a node.
    - For this phase your root node should be `BLACK` while all other nodes should be inserted as `RED` nodes. See Red-Black Tree properties listed above for the rules.

In this phase you will need to modify your `TreeNode` class to include a new private attribute (and any associated accessor methods necessary) for a node's color.

- `Color color_;`
    - This is an Enumeration type in my instance – but you could make it a Class if you so desire – either/or will suffice.

Don't forget your `virtual` destructors – you don't want any memory leaks! Once you have completed this – print out the root node and its color into the console/terminal window as shown below given the following input:

```
3 18 7 10 22 8 15 29
```

Your output would contain the following:

```
Root: 7
Color: BLACK
```

If you have gotten this far – buckle up, we are headed to Phase IV!

Phase IV – 11/17/2024

For Phase IV we reach the pinnacle of difficulty – in this phase you are responsible for implementing the necessary code to insure that all insertions keep the tree in a constant state of balance through adherence to the Red-Black Tree properties. You will need to add the following methods to your `RedBlackTree` class:

- Private (`*&` = Pass-by-Pointer Reference):
  - `void rotateLeft(TreeNode *& root, TreeNode *& newNode);`
    - This allows us to perform a left rotational shift of our tree to ensure proper balance is maintained. It will be called internally from our `balanceColor` method.
  - `void rotateRight(TreeNode *& root, TreeNode *& newNode);`
    - This allows us to perform a right rotational shift of our tree to ensure proper balance is maintained. It will be called internally from our `balanceColor` method.
  - `void balanceColor(TreeNode *& root, TreeNode *& newNode);`
    - This method allows us to maintain proper balance within our tree and properly adjusts the color of each node, if necessary, to adhere to the rules of a Red-Black Tree. This will be a large and complex function – you will need to implement the Red-Black Tree rules that we discuss in lecture here.
- Public:
  - `virtual void insert(int data);`
    - We need to modify this method from Phase IV by allowing it to call the `balanceColor` method described above to insure self-balancing criteria.

To wrap up this phase – print the root node (after all insertions) out to the console/terminal window. If you are able to successfully complete this you are ready for Phase V. An example of the output that this phase should generate is shown given the input below:

```
3 18 7 10 22 8 15 29
```

Your output would contain the following:

```
Root: 7
```

Phase V – 11/22/2024

For this phase we will implement the necessary traversal strategies such that we can actually "visualize" our Red-Black Tree. We will explore two (2) different traversal strategies in this phase – inorder and preorder. Your job is to implement these two (2) strategies in two (2) methods that will print out the respective node colors. In Phase V you will need to add the following methods to your `RedBlackTree` class:

- Public:
  - `void printRedNodes(TreeNode * root);`
    - You will need to use an **Inorder Traversal** in this method to search for Red colored `TreeNodes`.
  - `void printBlackNodes(TreeNode * root);`
    - You will need to use a **Preorder Traversal** in this method to search for Black colored `TreeNodes`.

An example of sample output is as follows - given the following sample input file:

```
3 18 7 10 22 8 15 29
```

Your output would contain the following:

```
Red Nodes: 8  15  18  29
Black Nodes: 7  3  10  22
Root: 7
```

Congratulations you have made it!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with a partner (s). If you choose to work on this project with a partner(s) you need to email me letting me know of your "group." Each "group" member is required to maintain the code in their respective repository. Be sure to make note of specific contributions of each team member so I can assign grades accordingly. **Note:** You are NOT required to work with a partner.

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **RedBlackTree.cpp**
- **RedBlackTree.h**
- **BinaryTree.cpp**
- **BinaryTree.h**
- **TreeNode.cpp**
- **TreeNode.h**
- **Node.h (optional – if you choose not to use the Node class from Assignment #4)**
- **Node.cpp (optional – if you choose not to use the Node class from Assignment #4)**
- **data.txt**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.