Additional Binary Search Tree Operations & Red Black Trees

Due: Friday, 16th of May

Summary

In the last assignment, you folks implemented Search, Inorder traversal, Successor, Min. and Max. for a Binary Search Tree. In this assignment, we'll build upon what you did earlier and transition into coding a red-black tree.

Do not worry; the algorithm's we've covered in class are fairly easy to translate into code. What I'd encourage you to do, especially while coding the red-black tree, is to try to understand what cases 1,2 and 3 actually mean (e.g. case 1 basically pushes red-node violations further up the tree).

You have the following tasks:

Steps: Write a Red-Black Tree class based off of the BST class you defined earlier.

a. If you haven't already, make the BSTNode class public. From that derive a class RBNode class (i.e. in terms of syntax RBNode : BSTNode).
b. Derive a class RBTree from the BST class (i.e. RBTree:BST). You would learn that you don't have to write separate functions for things like the InOrder traversal you wrote for the BST class. It also hammers home the point that a RB-Tree is a special kind of BST.
c. Write functions for **left rotate** and **right rotate** for the BST class, use them in the red-black tree class.
d. Write functions to **Insert** a node and **Fix-up** RBTree (if you do true object oriented style, you'll define an Insert function in the BST class itself, and then call it from a RBTree object because RBTree inherits from BST)
e. Write a function to return the ***black height*** of a node. (it'd take the input key from the visualizer and return the result in a MessageBox)
f. One thing to watch out for, is what do you do with the constructor of the RBTree or the InOrder walk that was tied to the visualizer in assignment 2? In the previous assignment, you hard coded a tree (either in the constructor, or some of you did that in the main), but this time around, the ***number of nodes is not fixed***. You should be able to start from an ***empty*** tree and work your way up to a perfect red black tree.
   To that end, in this assignment, you are allowed to use the built in List,

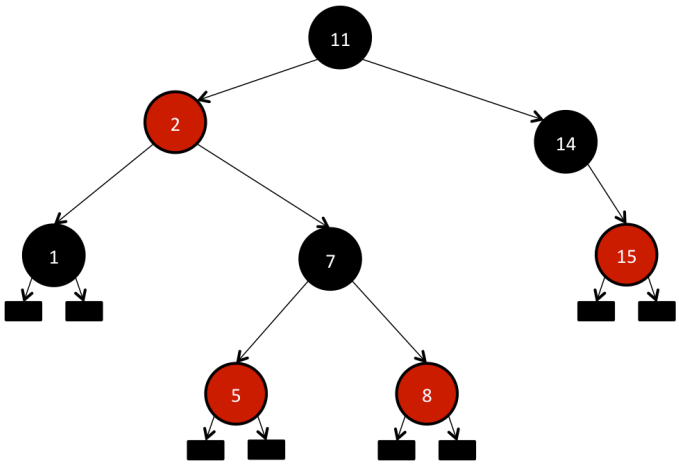
ArrayList or any other C# built in classes. Just so you know, you can define a list of RBNodes to which you can add nodes very easily (the syntax would look similar to List<RBNode> myRBTree, for more examples, check out this link)
However, if you you choose to use the Dynamic Array class you wrote yourself, that's awesome! We just didn't want you to worry if you aren't confident about the earlier assignment.

g. Test it by using the front end to check the color of the nodes (red or black) AND doing an InOrder traversal to ensure that it is still returning a sorted list ***after you have inserted a new node***. I recommend using the following brushes for red and black, so that Aleck and I don't go nuts looking at a plethora of shades that you folks have come up with ☺

h. Hook up the insert and search functions to the visualizer (you should already have the Search hooked up from assignment 3). Again, by default, the visualizer should show the inorder walk of the RBTree (so sorted list) the same way you showed queues in assignment 2 (blocks/circles with content inside them). Drawing circles instead of rectangles is recommended (but not mandatory). By the way, the visualizer now has 2 buttons, Search, and Insert. The names tell you what they are supposed to do ☺.

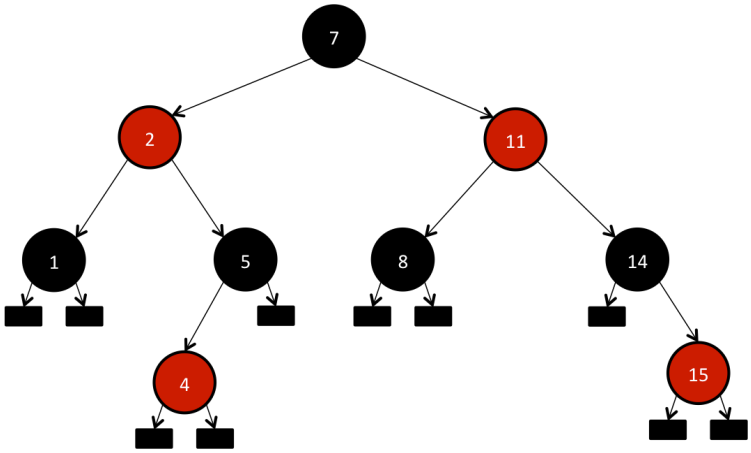
Other strategies that might help:

a. Declare a separate constructor that builds out the tree from slide# 9 of today's lecture (i.e. the Lecture_11_Insertion_Red_Black_Trees.pdf), also pasted below:



b. Run the program to double check that your inorder traversal works with this tree.

c. Insert 4 into the tree and check from the front-end/visualizer that the colors and the Inorder traversal works after 4 is inserted into the tree. The output should match the colors of the tree given below:



d. If your code passes the test of inserting 4, the next step would be to start from a blank tree, and insert the nodes 7,2, 11, 1, 5, 8,14 and 15 to see if you get the same configuration as the tree you started out before inserting 4.

e. By going through this exercise, you would have tested your rotation as well as insertion code.

Turning the assignment

- In Visual Studio, chose "Clean Solution" from the Build menu. This will remove all the binary files from the solution, leaving only the source code. That will make your assignment smaller and easier to upload.
- Close Visual Studio
- Right click on the folder icon for your assignment and choose the "Compressed (zipped) folder" selection from the Send To menu. o Note:Many of you are using commercial compression software such as WinRar. You may use these programs to make a zip file, if you prefer. However, you must specifically turn you program in as a ZIP file. Do not turn in other file formats such as RAR, TAR, TGZ, etc.
- Return to this assignment on blackboard and upload your zip file as your submission.