CS2134 HOMEWORK 9
Spring 2016
Due* 11:00 pm on Nov 27, 2016

Be sure to include your name at the beginning of each file! Assignment 9 include a programming portion and a written part. The programming portion must compile and consist of a single file (hw09.cpp). The written portion should consist of a single file (hw09written) in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

You must hand in both files via NYU Classes.

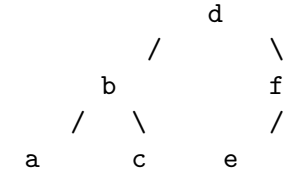
Programming Part

1. Add the following methods to the `BinarySearchTree` class.

   (a) Implement the method `contains` recursively.

   (b) Implement a method that takes two keys, `low` and `high`, and prints all the objects X that are in the range specified by `low` and `high`. (i.e. all keys in the range `[low, high]`.) Your program should run in `O(k + h)` time, where `k` is the number of keys printed and `h` is the height of the tree. Thus if `k` is small, you should be examining only a small part of the tree. Use a hidden recursive method and do not use an inorder traversal. Bound the running time of your algorithm using Big-Oh notation.

   (c) Ever feel contrarian? So much effort is spent creating a balanced BST. In this problem[1] you are to write a method called `stringy` that creates an unbalanced BST, a "stringy" BST, from a regular BST. If your tree contained

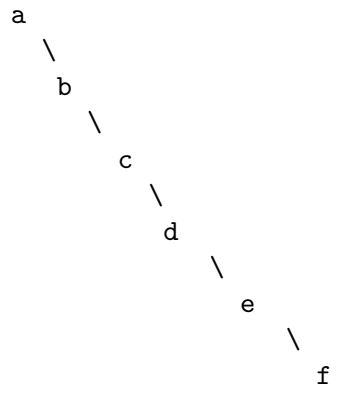


   after running `stringy`, all nodes in your tree should have their `left` member variable contain the `nullptr`. Thus the tree should have height $n - 1$.

*5% extra credit will be given if you turn this assignment in on Nov 26, 2016 at 11:00 p.m.

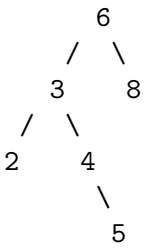
[1]This problem was suggested by Shahzaib.

In the example above, after calling `stringy` the tree would look like:



Your method must **not** create any new nodes[2]. I have removed the size method from the node class, so you do not need to update the size of each node.

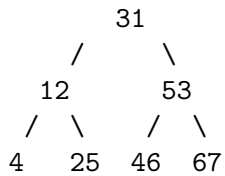
(d) Create a method called `average_node_depth` that computes the average depth of a node in the tree. e.g.



Then the average depth of a node is $(0+1+2+2+3+1)/6 = 9/6$ since there is one node at depth 1, two nodes at depth 2, one node at depth 3.[3]

2. Add a method to the `binary search tree` class that lists the nodes of a binary tree in level-order (It should first list the root, then the nodes at depth 1, then the nodes at depth 2, and so on). Your algorithm should have a worst case running time of $O(n)$, where n is the number of nodes in the tree. Explain how your algorithm fulfills this requirement. *Hint*: a `Queue<Node *>` might be helpful in solving this problem.[4]

e.g. For the following tree:



Printing the nodes of this tree in a level order would output: $31, 12, 53, 4, 25, 46, 67$.

[2]Please take the time to figure out how to solve this method on paper (using pictures) before you start coding.
[3]The average number of comparisons to find a item in this tree is 9/6+1.
[4]Level-order traversal of a tree is also called breadth-first traversal. I suggest you do **not** use recursion

3. (Extra Credit) You may do two of the following three extra extra credit problems. The points associated with each problem with be posted on Piazza.

   (a) Run empirical studies to estimate the average height of a node in a `binary search tree` by running 100 trial of inserting `n` random keys into an initially empty tree. For $n = 2^{10}, 2^{11}, 2^{12}$. For each n print the min depth of a node, the max depth of a node, and the average depth of a node.

   (b) Add[5] bidirectional iterators to the binary search tree class. To do this:

      i. add two extra pointers to the node class. Use these pointers to link each node to the next smallest and next largest node.[6]

      ii. add a tail node which is not part of the binary search tree[7]. Adding this node helps to make the code simpler

      iii. add the methods `begin( )` and `end( )` to the binary search tree class

   (c) See[8] `Shah's Memory.pdf`.

[5] This problem was suggested by Shahzaib.
[6] if you wish you can do this differently. We might ask you to explain your code.
[7] You can add a head node - but it is not needed.
[8] This problem is from Shahzaib.

Written Part

1. Write the pseudo code for:

   - programming problem 1. For each method, write the 3 to 8 steps needed to solve this problem.

2. For the programming problems in question 2, determine the running times of your implementations using Big-Oh notation.

3. This code is modified from the code we discussed in class. Does this code perform correctly? If not, describe all problems of this code and fix the code.

```
template <class Comparable>
BinaryNode<Comparable> * BinarySearchTree<Comparable>::findMin( Node * t ) const
{
    while( t->left != NULL && t != NULL )
         t = t->left;

    return t;
}
```

4. This code is modified from the code we discussed in class. Does this code perform correctly? If not, describe all problems of this code and fix the code.

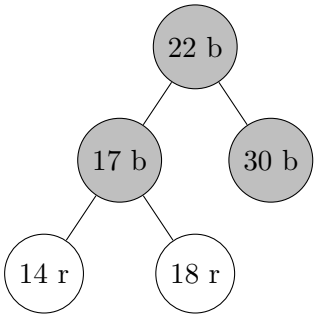
```
 void insert( const Comparable & x, Node *  t )
    {
        if( t == nullptr )
            t = new Node{ x, nullptr, nullptr, 1 };
        else if( x < t->element )
            insert( x, t->left );
        else if( t->element < x )
            insert( x, t->right );
        else
            ;  // Duplicate; do nothing
        t->size++;
    }
```

   For each of these questions, only print the number stored in the node. You do not need to print `r` or `b` which represent the *color* of the node.

5. Add `50,` then `44`, to this red-black tree (i.e. first insert `50` and perform any operations necessary (using only the algorithm presented in class) so the tree is again a red-black tree, then insert `44` and perform any necessary operations (using only the algorithm presented in class) so the tree is again a red-black tree.)

   Show the following:

   - Show the tree after inserting a value. If a violation occurred, state what the violation was (i.e. case 1, 2 or 3).
   - If the tree is not a red-black tree, use the algorithm we used in class to turn it into a red-black tree. **Show each step.**



6. **Do not turn in.** Please do this on your own. I will expect you to know how to do this for the exam. You can check your solution at https://www.cs.usfca.edu/ galles/visualization/RedBlack.html. Show the result of inserting `(15, 4, 2, 8, 16, 9, 5, 10, 17, 18)` into an initially empty red-black tree as described in class. Include the color of each node. Show the tree before and after each violation, and state which violation has occurred (i.e. case 1, case 2, or case 3).

7. Write the code needed to perform the method `rightRotateRecolor` for the `RedBlackNode` class. The method has the following prototype:

```
template <class Comparable>
void RedBlackTree<Comparable>::rightRotateRecolor( Node * & k2 )
```

8. When inserting an item into a Red-Black tree, what is the maximum number of pointer changes needed to adjust the tree? Explain your answer.