CS2134 HOMEWORK 8
Spring 2016
Due[*] **5:00 pm** on Wed. April 13, 2016
Extra Credit due on April 18, 2016 at 5:00 pm

Be sure to include your name at the beginning of each file! Assignment 8 include a programming portion and a written part. The programming portion must compile and consist of a single file (hw08.cpp). The written portion should consist of a single file (hw08written) 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. In this part, you will store a list of correctly spelled words and a point value associated with each word into a variable of type `map<string, int>`. You will perform the following steps:

   - Enter the point value from a file called `Letter_point_value.txt` for each letter into a variable of type `vector<int>`. The point value associated with 'A' goes into position 0, and 'B' goes into position 1, etc.
   - The point value of a word is determined by the point value of each letter. To compute the point value of a word, add up the point values of each letter in the word. For example, the point value of "cat" is $4 + 1 + 1 = 6$, since 'C' has a point value of 4, 'A' has a point value of 1, and 'T' has a point value of 1. Upper and lower case letters have the same point value. (e.g. So "CAT" also has a point value of 6.) Create a function to compute the point value of a word.
   - The words you will store are in a file called `ENABLE.txt`. You will read in each word and store it and its associated point value in a variable of type `map<string,int>`.

   On your own[1]. You can write a program to help you play Words With Friends.

   Remember the recursive function from the extra credit problem in a previous assignment, where the user enters a string and you find all the combinations of the string.

   For each string created from the recursive function, you can test to see if it is in the ENABLE word list. If so, you print out the word and the points associated with the word. Use the `map<string,int>` you created in programming part 1.

[*]10% extra credit will be given if you turn this assignment in on Tuesday April 12, 2016 at 11:00 p.m.

[1]Do not turn this in

2. 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[2] you are to write a method called `stringy` that creates an unbalanced BST, a "stringy" BST, from a regular BST. If your tree contained

```
            d
        /       \
     b          f
   /   \       /
a       c    e
```

   after running `stringy`, all nodes in your tree should have their `left` member variable contain the `nullptr`. Thus the tree should have height $n - 1$. In the example above, after calling `stringy` the tree would look like:

```
a
  \
   b
     \
       c
         \
          d
             \
               e
                  \
                    f
```

   Your method must **<u>not</u>** create any new nodes[3]. 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.

```
      6
    /  \
   3    8
 /  \
2    4
      \
       5
```

   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.[4]

[2]This problem was suggested by Shahzaib.

[3]Please take the time to figure out how to solve this method on paper (using pictures) before you start coding.

[4]The average number of comparisons to find a item in this tree is 9/6+1.

3. 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.[5]

   e.g. For the following tree:

```
         31
       /     \
     12       53
    / \      / \
   4   25  46   67
```

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

4. (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, the max, and the average.

   (b) Add[6] 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.

      ii. add a header and tail node which are not part of the binary search tree. Adding these nodes helps to make the code simpler

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

   (c) See[7] `Shah's Memory.pdf`.

[5]Level-order traversal of a tree is also called breadth-first traversal. I suggest you do **not** use recursion
[6]This problem was suggested by Shahzaib.
[7]This problem is from Shahzaib.

Written Part

1. Write the pseudo code for:
   - programming problem 2. 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.

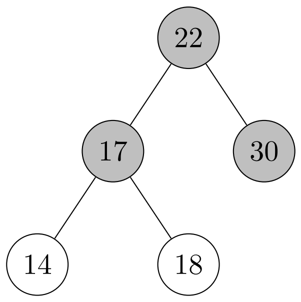
```
template <class Comparable>
BinaryNode<Comparable> * BinarySearchTree<Comparable>::findMin( Node * t ) const
{
    while( t->left != NULL && t != NULL )
         t = t->left;

    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++;
    }
```

5. For the following tree:



   (a) what is the output of an inorder traversal of the tree?
   (b) what is the output of an preorder traversal of the tree?
   (c) what is the output of an postorder traversal of the tree?