Version: 11-25-2015 12:30 PM

CS 2223 B15 Term. Homework 4

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/b15/#policies .
- Due Date for this assignment is 2PM Friday December 4th. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM will receive zero credit.
- Submit your assignments electronically using the blackboard site for CS2223. Login to **my.wpi.edu** and go to CS2223 under “My Courses” then go to “Assignments” and submit your homework under “HW4”. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID.hw4 where USERID is your CCC user id (i.e., your email address without the @wpi.edu).

Primary Instructions

For this homework, you can eliminate the need to store an attribute N for each node. Also there no longer is an associated **`Value`**; rather we are only concerned with **`Keys`**. You should use a Node structure similar to the following:

```
public class BST<Key extends Comparable<Key>> {

  class Node {
    Key    key;
    Node   left, right;  // left and right subtrees

    public Node(Key key) {
      this.key = key;
    }

    public String toString() { return "[" + key + "]"; }
  }

 ...

}
```

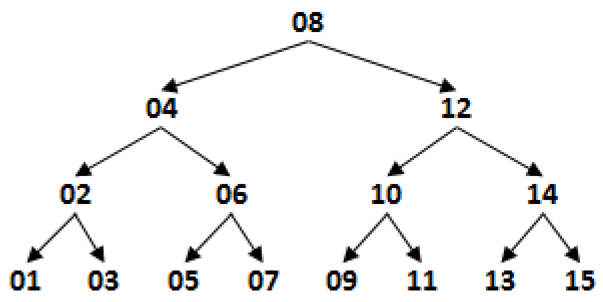
Make sure you submit a writtenAnswers.txt file which contains your written answers.

Make sure you submit the Java source files. Watch out for the subpackage ‘expression’. To be safe, zip up your entire top-level **hw4** directory, within which the expression subfolder exists as a child.

Version: 11-25-2015 12:30 PM

Q1. Balanced Binary Tree (25 pts)

(a) If you insert N values into a binary search tree in sorted order, the tree loses its efficiency because its operations are dependent on the height of the tree, which grows to be exactly N. Instead consider the following diagram.



01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

As you can see, these fifteen items are perfectly packed into a tree of size 15. You are to write a constructor for the BST class which takes a sorted array whose size is one less than a power of two.

Hint: Think about inserting the elements in a specific order, that is, level by level. Observe each of the numbers on the level are evenly spaced from each other…

(b) Demonstrate you succeeded by writing a method to dynamically compute the height of a binary tree, that is, the maximum distance from any leaf node to the root node. Pattern this recursive height method after the many examples shown in class; you will complete the following two methods:

```
public int height() { ... }
private int height(Node n) { ... }
```

(c) Write a benchmark program that demonstrates you can create such balanced trees for N=2^k-1 for k=2 to 10. To do this, generate the balanced trees and their computed height.

Q2. Delete Max (25 pts)

You have already seen in the Heap structure how **deleteMax** can be accomplished with no more than *2*Log N* compares, where N is the number of elements in the heap (p. 319). You can add a similar method to **BST**. One immediate question to ask is which one is more efficient on random data sets. Define efficiency by the least number of elements in the heap (or BST) visited by **deleteMax**.

(a) Modify the **BST** class to add a **deleteMax** operation as well as a max operation:

```
public Key max() { ... }
public void deleteMax() { ... }
Node deleteMax(Node parent) { ... }
```

Now, within these methods, count the number of ~~comparisons made~~ elements inspected, using a similar strategy to what you did with **MaxPQ**.

(b) Modify the **CompareBSTandHeap** class which runs a benchmark computation to determine which algorithm uses the most ~~comparisons~~element inspections. You will run T=512 randomized trials of constructing a **MaxPQ** and a **BST** from the same N values (where N varies from 8 to 1,048,576). That is, for each trial the values in the **MaxPQ** are the same values in the **BST** you construct.

In each trial, you will remove the maximum element from each structure until they are both empty (note: you must compare these maximum values against each other to make sure your code is properly working).

For each trial, you will compute the ratio HC/BC where HC is the number of element inspections~~comparisons~~ used by the Heap and BC is the number of element inspections ~~comparisons~~ used by the BST; then you must compute the **maxRatio** which reflects the largest such ratio HC/BC seen for all trials associated with a given N value. Finally, output a table that looks like the following, which shows the value N and the computed maxRatio for the given value of N.

```
8      17.0
16     4.8333335
32     5.965517
64     3.9237287

...
```

(c) Which of these structures inspects ~~uses~~ MORE elements ~~comparisons~~ in total to remove all N elements drawn from random data?

Note: computing the number of elements being inspected by the sink method in Heap is a bit challenging. Feel free to use the following:

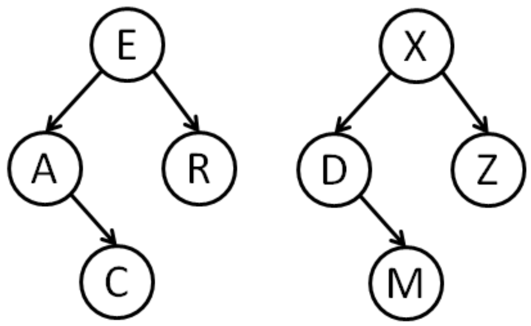
```
private void sink(int k) {
  boolean first = true;        // Make sure not to over-count
  while (2*k <= N) {
    if (first) { inspectedElementCount++; first = false; }
    int j = 2*k;
    if (j < N) { inspectedElementCount++; }
    if (j < N && less(j, j+1)) j++;
    inspectedElementCount++;
    if (!less(k, j)) break;
    exch(k, j);
    k = j;
  }
}
```

Q3. Binary Tree Methods (25 pts)

(a) Write a method that returns a complete copy of a BST, with the exact same structure and keys as the original BST. The signature of this method must be as follows (together with a helper method).

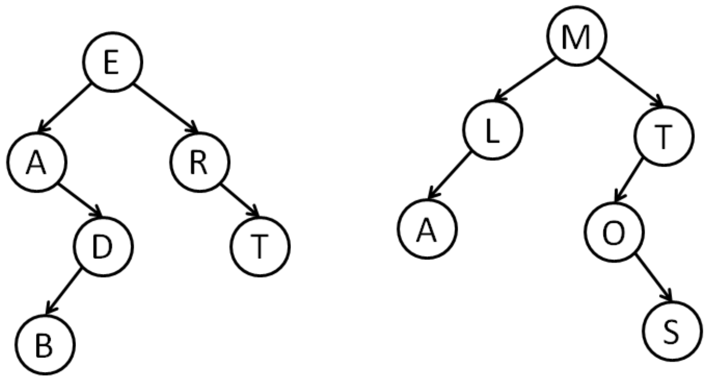
```
public BST<Key> copy () { … }
private Node copy (Node n) { … }
```

(b) Write a method that determines whether two BSTs have the **exact same structure** regardless of the keys stored by the BST. For example, the following two BSTs have the same structure:



```
public boolean identicalStructure(BST<Key> bst) { … }
private boolean identicalStructure (Node one, Node two) { … }
```

(c) Write a method that determines whether two BSTs have **structures that are mirror images of each other,** regardless of the keys stored by the BST. For example, the following two BSTs have mirror image structures.



```
public boolean mirrorImage(BST<Key> bst) { … }
private boolean mirrorImage (Node one, Node two) { … }
```

(d) Write a small demonstration program that uses the above trees to validate you have working **`identicalStructure`** and **`mirrorImage`** implementations.

Version: 11-25-2015 12:30 PM

Q4. Evaluating Binary Trees (not BSTs) – 25 points

The Binary Tree structure appears in countless places in computer science. In this question, you will see one such example. First, these are no longer Binary Search Trees (BSTs)! Each node in this Binary Tree still has two children (one left and one right) but it is up to the domain to decide how to structure the data. In this case, the tree represents a mathematical equation.

(a) Develop subclasses (appropriately extending **UnaryOperatorNode**, **BinaryOperatorNode** or **NoParameterOperatorNode**) that handle the following operations:

sqrt	Square Root of a number. Thus **`( sqrt 7 )`** is equal to `2.6457513110645907`
^	Exponent of a number by another. Thus **`( 3 ^ 4 )`** is equal to 81.0
*	Multiplication of two numbers . Thus **`( 5 * 7 )`** is equal to 35.0
/	Division of two numbers. Thus **`( 1 / 3 )`** is equal to `0.3333333333333333`
`pi`	Represents π. Thus **`( 2 + π )`** is equal to `5.141592653589793`
-	Subtraction of two numbers. Thus **`( 1 - 3 )`** is equal to `-2.0`

(b) Integrate these classes into Evaluate.java which takes Dijkstra's two stack algorithm for evaluating InFix operators and constructs a Binary Tree representing the desired equation.

Modify the existing **`Evaluate.java`** file you see in package **`algs.hw4.expression`**

(c) Test on the following expressions. State the output for each (both the value and the formatted expression).

- **`( 3 + ( sqrt 8 ) )`**
- **`( 4 + ( ( 7 * 2 ) + ( 6 * 3 ) ) )`**
- **`( ( ( 3 * 4 ) + ( 8 ^ 3 ) ) / ( ( 2 + 1 ) * ( sqrt 7 ) ) )`**

(d) **BONUS QUESTION [5 pts.]** Review the instructions in **`EvaluateBonus`** and see if you can upgrade the code to handle n-aray formulas based on binary operators. For example, you should handle:

- **`( 4 + 6 + 8 + 9 ) which computes to 27`**
- **`( 2 + 3 * 4 - 7 / 2 ) which computes to 6.5`**

(3) **BONUS QUESTION [1 pts.]** PEMDAS order of operations!

Note: the above formula doesn't apply the PEMDAS rule for order of operations; it simply applies them all from left to right. Instead it should be evaluated as **`(2 + (3*4) - (7/2))`** which equals 10.5.

I have left some hints in **`EvaluateBonus`** explaining how this could be accomplished. Good luck!