

## CS 2223 D18 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/d18/#policies](http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d18/#policies).
- Due Date for this assignment is 9PM Monday April 23<sup>rd</sup>. Homework received after 9PM receive zero credit.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate 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 where USERID is your CCC user id.
- Submission information is found at the end of this document.

### Primary Instructions

## Q1. Balanced Binary Tree (45 pts)

There is a simplified AVL implementation, `algs.hw4.AVL`, which is provided to you in the git repository; this version does not allow keys to be deleted from the tree. It has a `keys()` method that returns a `Queue` containing all values in the AVL tree in sorted order (that is, by using repeated dequeue operations, you can process all keys in sorted order).

(a) [10 pts] Complete the `merge` method in this AVL class, which inserts into an AVL tree all of the keys found in the other AVL tree. Note that 'other' should remain unaffected by this method.

```
public void merge(AVL<Key> other) {
}

```

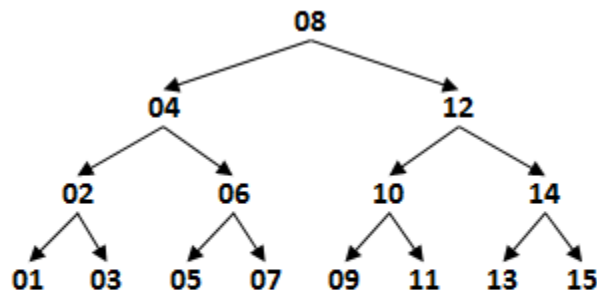
(b) [15 pts] Demonstrate that you have a working `merge` method by modifying the `algs.hw4.Question1B` class. Once you have properly merged two AVL trees, then you will use

```
SeparateChainingHashST<Integer, Integer> table = new SeparateChainingHashST<>();

```

Use this symbol table to count the number of times a given key appears in the merged AVL tree. This symbol table uses an `Integer` as the key, and naturally the value will be an integer for the count. The purpose of this question is to write code that ensures all keys are present in the merged tree. See the class file for more information.

(c) [20 pts] Consider the following diagram, which shows a **full binary tree** of  $n=2^k - 1$  nodes with  $k=4$ . Observe that there are  $(n+1)/2$  leaf nodes and all interior nodes (that is, the ones that have a child) have exactly two children.



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 an AVL tree of size 15. Complete the constructor:

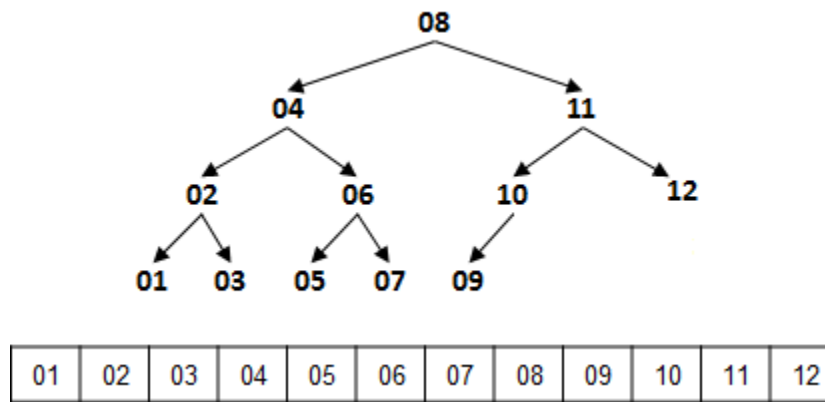
```
public AVL(Key[] other) {
    ...
    StdOut.println ("number of rotations:" + rotations);
}

```

which will construct the AST from a sorted array of  $n=2^k - 1$  values **without needing to perform any rotations**. You can validate that no rotations were performed, simply by printing the value of 'rotations' at the end of the constructor. Modify `algs.hw4.Question1C` directly.

*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...*

(c) **[1 pt] Bonus Question:** Have this constructor work for any  $N>0$ , not just  $2^k - 1$ . For this to work, you will aim to create a binary tree that looks like it maintains the 'shape heap property'. This tree will have height  $h$  and it should be a full binary tree for the first  $h$  levels, and then the final level would contain leaves, starting from the left-most possible leaf node. Here is an example:



Let  $h = \text{floor}(\log_2(N))$ . In the above example, this would be 3, thus the first  $h$  levels (depth 0, depth 1 and depth 2) are all complete. The remaining  $N - 2^h + 1 = 12 - 8 + 1 = 5$  nodes appear on depth 3 and are all leaves.

(d) **[1 pt] Bonus Bonus Question:** Write a static method that takes in two AVL trees and returns a new AVL tree that represents the combined values in both original trees (note: original trees are not changed in any way). The trick is that **you must perform no rotations**. [Hint: use (c) in some way]. If you want to see how much faster this is, than say, merging two AVL trees. Consider running some experiments based on question 1a.

## Q2. Evaluating Binary Trees (not BSTs) – 35 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
- `triangle` Returns the  $n^{\text{th}}$  Triangle number. Thus ( `triangle 6` ) equals  $6*7/2 = 21.0$
- `^` 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  $\pi$ . Thus ( `2 + pi` ) is equal to 5.141592653589793
- `-` Subtraction of two numbers. Thus ( `1 - 3` ) is equal to -2.0

(b) Integrate these classes into `algs.hw4.expr.Evaluate` which takes Dijkstra's two stack algorithm for evaluating InFix operators and constructs a Binary Tree representing the desired equation.

Modify the existing `Evaluate` file you see in package `algs.hw4.expr`. Feel free to modify any classes. Naturally you will be adding you own classes to address the new features above.

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

- ( ( 1 + ( sqrt 5 ) ) / 2 )
- ( 4 + ( ( triangle 12 ) \* pi ) )
- ( ( ( 3 \* 4 ) + ( 8 ^ 3 ) ) / ( ( 2 - 1 ) \* ( sqrt 7 ) ) )

(d) **BONUS QUESTION [1 pts.]** Review the instructions in `algs.hw4.expr.bonus.EvaluateBonus` and see if you can upgrade the code to handle n-ary 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

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

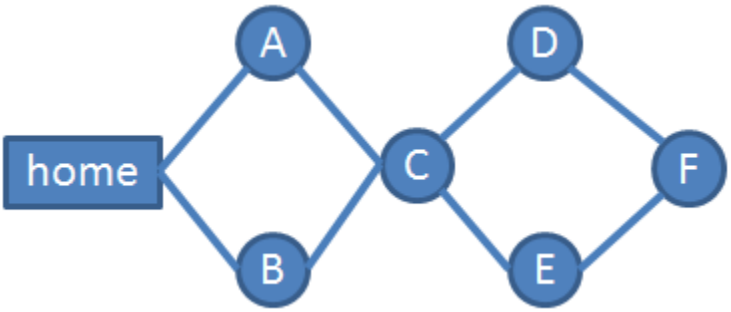
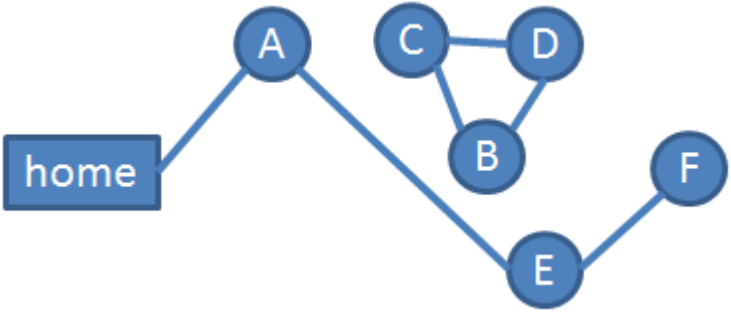
Note: the above formula doesn't apply the PEMDAS rule for order of operations; it simply applies them all from left to right. Instead it should be evaluated as ( 2 + ( 3\*4 ) - ( 7/2 ) ) which equals 10.5.

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

### Q3. DFS Exploration (20 points)

When planning a trip, you always want to know if you can reach any destination from a starting point. Write a program to read input representing a graph and answer this burning question!

The first line of input contains the available locations (other than 'home') separated by spaces. The next lines contain all edges that exist in the graph, with two vertex names on each line separated by spaces. The last line of the input contains a single "0" on a line by itself, which states that there are no more edges to be recorded.

Scenario	Sample Input & Sample Output
	<pre>A B C D E F home A home B B C C D C E E F A C D F 0 Can get everywhere</pre>
	<pre>A B C D E F home A B C B D C D E F A E 0 Can't get from 'home' to B Can't get from 'home' to C Can't get from 'home' to D</pre>

Your program should print out "Can get everywhere" if the graph is connected. If there exists any vertex that is not connected via some path from home, then you should print these vertices out (in any order) as shown in the text above.