Version: 4-11-2019 10:10 PM

CS 2223 D19 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/d19/#policies.
- Due Date for this assignment is **2PM Monday April 22nd**. Homework received after 2PM receive 25% penalty. Homework received after 6PM 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

Note that Homework 4 is due on Monday April 22nd.

Version: 4-11-2019 10:10 PM

Q1. Balanced AVL Binary Tree (20 pts)

AVL trees use rotation to self-balance as keys are inserted and removed. However, it is possible to insert keys in a certain order to minimize – or even eliminate – any rotations.

For this assignment, copy the `algs.hw4.Question1` class into your `USERID.hw4` package. You will see that it inserts the values form 1 to 12 (in ascending order) into an empty AVL tree and prints out the number of rotations and the height of the final tree.

Q1.1 Remove Rotations (10 pts):

Come up with a different arrangement of these twelve numbers which, when inserted, constructs an AVL tree with no rotations.

```
        int[] values = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

List these values within the `WrittenQuestions.txt` file and modify `Question1.java` accordingly.

Q1.2 Model Number Of Rotations (10 pts):

Using `StdRandom.shuffle()` to rearrange the elements of an array, run a number of T=1,000 independent trials to compute the maximum number of rotations when inserting N randomly ordered integers from 0 to N-1 into an empty AVL tree. Also compute the maximum height of the resulting tree. Run your experiment for N=1, 3, 7, 15, up to 4095 (these are the powers of 2 minus one).

Your output should look like this:

```
N       MaxHt. MaxRot
1       0      0
3       1      2
7       3      8
15      4      18
…
```

Q1.3 Bonus Question (1pt):

Find an ordering that produces a tree whose height is **four** or prove that one can't exist.

Q1.4 Bonus Question (1pt):

Find an ordering that produces a tree whose height is **five** or prove that one can't exist.

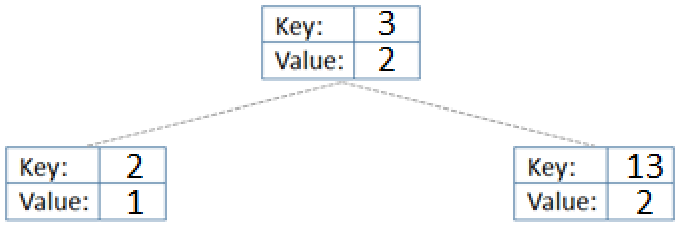
Q1.5 Bonus Question (1pt):

Come up with a formula f(N) that predicts the values computed in Q1.2 for both `MaxHt.` and `MaxRot` when N is one less than a power of 2.

Q1.6 Bonus Question (1pt): Modify your formula to work with arbitrary N.

Q2. Composite revisited (80 pts)

The AVL tree can store (key, value) pairs since it provides the behavior of a Symbol Table. You are to take advantage of this to, once again, implement a `Composite` data type which represents a positive integer greater than 1. According to the Fundamental Theorem of Arithmetic, every integer greater than 1 is either a prime number itself or can be represented as the product of prime numbers. For example, $3042 = 2x3x3x13x13$ which can be written more concisely as $3042 = 2x3^2x13^2$. Using an AVL tree representation, this value would be stored as follows.



```
package USERID.hw4;

public class Pair {
  final Key key;
  final Value value;

  public Pair (Key k, Value v) {
    this.key = k;
    this.value = v;
  }
}

public class Composite {
  /**
    * Keep track of the AVL tree of factor/exponents based at this root.
    * Each key is a BigInteger factor; each value is a power of that factor. */
  AVL<BigInteger, Integer> tree = new AVL<BigInteger, Integer>();

  // operations that you have to complete
  public Composite(BigInteger val) { ... }
  public String toString() { ... }
  public boolean equals(Object o) { ... }
  public BigInteger value() { ... }
  public boolean isPrime() { ... }
  public boolean divisibleBy(BigInteger factor) { ... }
  public Composite multiply(Composite comp) { ... }
  public Composite gcd(Composite comp) { ... }
  public Composite lcm(Composite comp) { ... }
}
```

Note: I have simplified this assignment by removing the "add" operation. In addition, a `Composite` value cannot be 1. That is, it must truly represent a value > 1.

Version: 4-11-2019 10:10 PM

Copy `algs.hw4.Composite` into USERID.hw4 and complete its implementation, using the AVL tree for storing all (key, value) pairs of (factor, power). More documentation is found in the sample file.

The AVL tree has the following `pairs()` method which will prove useful. This returns all key values (in order) as stored in the AVL tree within a `Pair` object.

```java
/**
 * Returns all keys in the symbol table as an <tt>Iterable</tt> of pairs so
 * we don't lose the values.
 * To iterate over all of the keys in the symbol table named <tt>st</tt>,
 * use the foreach notation: <tt>for (Pair p : st.pairs())</tt>.
 *
 * @return all keys in the symbol table
 */
public Iterable<Pair<Key,Value>> pairs() {
  if (root == null) { return empty; }
  return pairs(min(), max());
}
```

Note that `Pair` objects – as well as `Composite` objects – are immutable.

We will validate the output against the set of test cases in **`TestComposite`** that we develop for the grading. Individual breakdown of points is found on the rubric. There is a **`PerformanceTest`** included which reveals the total time to compute $(n!)^2$ for standard values of n (to run this performance test, copy it to your `USERID.hw4` package and execute).

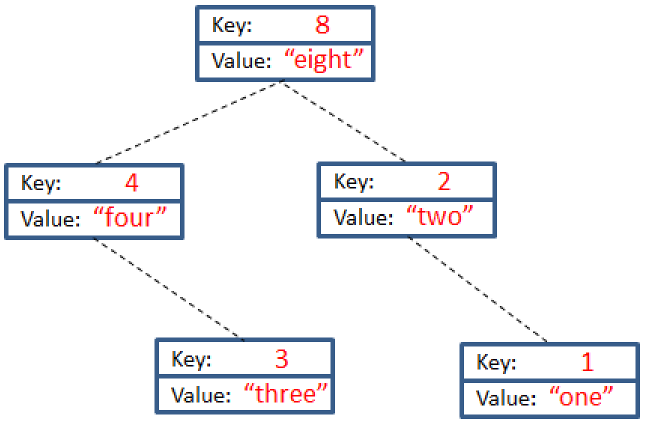
Version: 4-11-2019 10:10 PM

Q3. Bonus Question (1 point)

The binary tree structure can be reused to implement a binary heap and thus implement a priority queue.

```
// For a Node in a BinaryMaxHeap, it is certain only that the key is greater
// than the keys in either the left or the right child (should they exist).
  class Node {
     Key    key;
     Value  value;
     Node   left, right;  // left and right subtrees
     Node (Key k, Value v) { this.key = k; this.value = v; N = 1;}
     int N;
     public String toString() { return "[key=" + key + ", value=" + value + "]"; }
   }
```

A Node in a `BinaryMaxHeap` (whose class you should copy into your USERID.hw4 to complete) represents a (key, value) pair where the key is greater than (or equal) to the keys in either the left child or the right child (if they exist). The following is a sample binary max heap:



As you can see, we are no longer requiring the “heap shape property” as we did before when a heap was stored in an array. All that matters is that, for each node, its key is larger then the keys in either of its two children.

Complete the implementation of `BinaryMaxHeap` which offers the interface that we defined earlier for a priority queue, and then the `BonusQuestion` will properly execute and show an output as follows:

```
****************************************
***************************************
**************************************
*************************************
************************************
***********************************
**********************************
*********************************
********************************
*******************************
******************************
*****************************
****************************
***************************
**************************
*************************
************************
***********************
**********************
*********************
********************
*******************
******************
*****************
****************
***************
**************
*************
************
***********
**********
*********
********
*******
******
*****
****
***
**
*
```