Programming Assignment #3: Treaps

COP 3503, Fall 2013

Due: Sunday, October 13, 11:59 PM via Webcourses@UCF

Abstract

In this assignment, you will implement treaps. You will gain experience working with a probabilistic data structure and implementing tree rotations. You will also solidify your understanding of generics in Java by making your treaps capable of holding any type of Comparable object.

Don't be intimidated by this assignment (unless that's what it takes to get you to start early, in which case you should be **VERY INTIMIDATED!**). Because treaps rely heavily on BST insertions and deletions, you already have a great framework in place for this program from Assignment #2. All you have to do is tweak it to implement some heap-inducing rotations based on randomly generated integers.

Deliverables

Treap.java

1. Problem Statement

Implement the probabilistic Treap data structure. It is important that you implement the insertion and deletion operations as described in class. (In particular, be sure to read my note in Webcourses on how deletion is handled, which includes a discussion of how this differs from our traditional approach to BST deletion.) Your Treap class must be generic. Since there is an ordering property in treaps, you must also restrict the type parameter (e.g., AnyType) to classes that implement Comparable.

2. Method and Class Requirements

Implement the following methods in a class named Treap. You may replace `AnyType` with something else, if you wish (e.g., `T`), as long as the function names and return types stay the same.

```
public void add(AnyType data);
```

Add *data* to the treap. Do not allow insertion of duplicate values. You will have to generate a random priority for this node. Find an efficient way to ensure that no duplicate priorities are generated, which uses no more than **O(n)** extra memory.

```
public void add(AnyType data, int priority);
```

Insert *data* into the treap (as above), but do not generate a random node priority for the new node. Instead, let the programmer (me) pass the priority as a parameter. I'll use this to test your code with my own sequence of not-so-random priorities.

```
public void remove(AnyType data);
```

Delete *data* from the treap (if it is present).

```
public boolean contains(AnyType data);
```

Return `true` if the treap contains *data*, `false` otherwise.

```
public int size();
```

Return the number of elements in the treap **in O(1) time**.

```
public int height();
```

Return the height of the treap. (An O(n) solution is fine here.)

```
public static double difficultyRating()
```

Return a double on the range 1.0 (ridiculously easy) to 5.0 (insanely difficult).

```
public static double hoursSpent()
```

Return an estimate (greater than zero) of the number of hours you spent on this assignment.

3. Program Output

I have included a basic framework in Traversals.java that includes in-order, pre-order, and post-order traversal methods. Please set up your Treap class so that it is compatible with calls to these methods. They are designed to produce output that conforms to the exact format I will be looking for in testing. Note that you will have to create a Node class in Treap.java that is compatible with these methods.

4. Compilation Requirements: No Compile-Time Warnings

With respect to generics, your source code must not produce any warnings when compiled. Please do not give your code to classmates and ask them to check whether their compilers generate compile-time warnings. Remember, sharing code in this course is out of bounds for assignments.

5. Grading Criteria and Miscellaneous Requirements

The *tentative* scoring breakdown (not set in stone) for this programming assignment is:

60%	program compiles and passes test cases (using both versions of the `add()` method)
20%	program compiles without warnings, method signatures are correct, and generics with Comparable are properly implemented
10%	`difficultyRating()` and `hoursSpent()` return doubles in the specified ranges
10%	adequate comments and whitespace

Programs that do not compile will receive zero credit. Please be sure to submit your `.java` file, not a `.class` file (and certainly not a `.doc` or `.pdf` file). Your best bet is to submit your program in advance of the deadline, then download the source code from Webcourses, re-compile, and re-test your code in order to ensure that you uploaded the correct version of your source code.

Your program should not print anything to the screen. Extraneous output is disruptive to the TAs' grading process and will result in severe point deductions. Please do not print to the screen.

Please do not create a java `package`. Articulating a `package` in your source code could be disruptive to the grading process and will result in severe point deductions.

Name your source file, class(es), and method(s) correctly. Minor errors in spelling and/or capitalization could be hugely disruptive to the grading process and may result in severe point deductions. Please double check your work!

Test your code thoroughly. Please be sure to create your own test cases and thoroughly test your code.

Start early! Work hard! Ask questions! Good luck!