Computer Science 331 — Winter 2016
Assignment #3

Instructions

This assignment concerns lecture material that was introduced in this course on or before Wednesday, March 9.

Please **read the entire assignment** before you begin work on any part of it.

This assignment is due by 11:59 pm on Wednesday, March 23.

There are a total of 100 marks available, plus an additional 20 bonus marks.

Questions

A *mapping* is a partial function $f : K \mapsto V$ from a set K of "keys" to a set V of "values." Since this is a partial function, there is *at most* one value $f(k) \in V$ that is defined for each key $k \in K$. If the set of keys k such that $f(k)$ is defined is finite, and there is a useful "linear order" defined for the set K, then a binary search tree is one of several data structures that can be used to represent the mapping f. Note that a mapping is another way to describe the dictionary ADT described in class.

The Java Collections Framework contains an interface `SortedMap` describing such a mapping. The following questions will make use of the `SimpleSortedMap` interface (available on the Assignment 2 web page), a simplified version of `SortedMap`. The file `LinkedListMap.java` consists of a linked list based implementation of `SimpleSortedMap`, and `CountWords.java` contains a program the uses a `SimpleSortedMap` to identify the words and their frequencies from a text file.

The purpose of this assignment is to give you practice working with a binary search tree implementation of a mapping. In particular, you will:

- create an implementation of a binary search tree in which all algorithms are iterative (as opposed to recursive),
- create an Iterator object for your binary search tree class that implements an iterative version of an in-order traversal,
- work with the JCF `TreeMap` class, which contains an implementation of a red-black tree.

You will have to read background material on Java iterators and tree traversals. The file `traversals.pdf`, available on the Assignment 3 web page and the course Schedule page, contains a small amount of material on tree traversals. Further information on-line should be easy to find if you need it.

1. **(15 marks)** Prove that the following iterative search algorithm for a binary search tree is correct (i.e., it is partially correct and terminates). You must use a loop invariant (for partial correctness) and a loop variant (for termination).

```
$V$ search($T$, $key$)
PRECONDITION: {$T$ is a binary search tree with keys of type $E$ and associated values of type
  $V$; $key$ is of type $E$ }
  $v = null$
  $curr = T.root$
  while $curr \neq null$ and $v = null$ do
    if $key = curr.key$ then
      $v = curr.value$
    else if $key < curr.key$ then
      $curr = curr.left$
    else
      $curr = curr.right$
    end if
  end while
  if $v = null$ then
    throw KeyNotFoundException
  else
    return $v$
  end if
POSTCONDITION: {$T$ and $key$ are unchanged; if $key$ is a key in $T$ then the associated value
  is returned; if $key$ is not in $T$ then a KeyNotFoundException is thrown }
```

2. **(10 marks)** Give pseudocode for an *iterative* algorithm for insertion into a binary search tree.

3. **(10 marks)** Give pseudocode for an *iterative* algorithm for an in-order traversal of a binary search tree. Your algorithm should make use of a stack.

4. **(30 marks)** Implement a Java class called `BSTMap` that implements the `SimpleSortedMap` interface using a binary search tree. This class should satisfy the following:

   - The `search`, `insert`, `delete`, and `modify` functions must be implemented *iteratively*. You must *not* use a stack for any of these functions.
   - The `Iterator` returned by the `iterator` function must perform an in-order traversal of the binary search tree, accessing the keys in ascending order. You may use any implementation of the Stack ADT you like for your implementation. The `remove` operation, required by `Iterable`, does not need to be supported, and can just throw an `UnsupportedOperationException`.

   Be sure to document your class thoroughly using `javadoc` tags and assertions as appropriate. Both the quality of your implementation and documentation will be taken into account when grading this question. *5 of the 30 available marks for this question will be allocated to documentation.*

5. **(15 marks)** Implement a Java class called `RBTMap` that implements the `SimpleSortedMap` interface using a red-black tree. You should do this by using the `TreeMap` class from the Java

API, and using existing functions of `TreeMap` to implement the required functions from the interface. The Java Collections API documentation, available through the course web page, provides all the information you need to know about `TreeMap` to enable you to complete this question.

Be sure to document your class thoroughly using `javadoc` tags and assertions as appropriate. Both the quality of your implementation and documentation will be taken into account when grading this question. *5 of the 20 available marks for this question will be allocated to documentation.*

Note: The `CountWords.java` program will be used to test your `BSTMap` and `RBTMap` classes. This program accepts an optional 2nd command line parameter that allows you to choose which implementation of `SimpleSortedMap` is used. As long as your `BSTMap` and `RBTMap` correctly use this interface and provide a default constructor, the only required modification to `CountWords.java` should be uncommenting the lines in the program where `SimpleSortedMap` instance is created. The TAs will use the version of this program provided on the web site as-is, and it is your responsibility to make sure that your classes work properly with it.

6. **(15 marks)** Add a function `height` (this function may be recursive or iterative) to the `BSTMap` class that computes the height of the binary search tree. Write a program called `A3Q6.java` that computes some statistics on the height of m different binary search trees created by inserting n *distinct* random integers into the tree, where n and m are command-line parameters. Run your program for values of n = 100, 1000, 10000, and 100000 and $m = 100$, and present a table with the following information for each value of n :
   - the minimum height of all 100 random trees
   - the maximum height of all 100 random trees
   - the average height of all 100 random trees
   - an upper bound on the expected height of a random binary search tree of size n, as per the average case analysis presented in class
   - a worst-case upper bound on the maximum height of a random red-black tree of size n.
7. **(5 marks)** Is the data you obtained in the previous question what you would expect? Why or why not?

Note: Although you are not being asked to provide a test suite for this assignment, you are responsible for testing your classes as thoroughly as possible in order to make them as bug-free as possible. You should feel free to use JUnit or to add any auxiliary functions to your classes that will help with this process.

Bonus Questions

The following bonus question should only be attempted after the previous questions have been completed satisfactorily. While solutions to these questions are not required for full credit on this assignment, correct solutions can result in a grade of more than 100 %.

8. **(20 bonus marks)** Implement a Java class called `MyRBTMap` that implements the Java interface `SimpleSortedMap` using the red-black tree data structure from scratch, i.e., without using `TreeMap`. Modify the program `CountWords.java` to use your class as well.