Assignment 1: AVL & Splay Trees

COMP2014J: Data Structures and Algorithms 2
Weight: **10% of final grade**
Due Date: 08:00 Monday May 6th 2019
Document Version: 1.0

Introduction

This assignment is intended to give you experience implementing, AVL and Splay trees. It is also a good exercise to gain experience about how object references work in Java.

Source code that you can start from has been posted to Moodle in the file `Assignment1-Source.zip`. This also contains the Javadoc API documentation for the classes that have been provided (in the "doc" folder). Import this project into Eclipse in the usual way.

Tasks

The main tasks for this assignment are:

- Implement the key methods for an AVL Tree.
- Implement the key methods for a Splay Tree.
- Develop a strategy to test if your implementations are correct.
- Improve the efficiency of the AVL Tree implementation.

Implementation of AVL Tree Methods

The source code contains a partial implementation of an AVL Tree in a file called `AVLTree.java` in the `dsa.impl` package. Your work in this section **must** be in this class.

You must implement the following methods:

- `private void restructure( INode<T> x )` – trinode restructuring (the three nodes are x, its parent and its grandparent).

  Hint: You can cast to an `INode<T>` to a `BTNode` in the same way as you did in Worksheet 4.

- `public void insert( T value )` – insert a value into the AVL tree.
- `public void remove( T value )` – remove a value from the AVL tree.
- `public boolean contains(T value)` – check to see if a value is contained in the AVL tree. Returns `true` if the value is in the tree, or `false` if not.

If you wish, you may create other methods that help you to complete the task (e.g. `rightRotate(INode<T> n)`, `leftRotate(INode<T> n)`, etc.).

Implementation of Splay Tree Methods

The source code contains a partial implementation of a Splay Tree in a file called `SplayTree.java` in the `dsa.impl` package. Your work in this section **must** be in this class.

You must implement the following methods:

- `private void splay( INode<T> n )` – splay a node in the tree.
- `public void insert( T value )` – insert a value into the splay tree.
- `public void remove( T value )` – remove a value from the splay tree.
- `public boolean contains(T value)` – check to see if a value is contained in the splay tree. Returns `true` if the value is in the tree, or `false` if not.

Testing the Tree Implementations

It is important to check whether your implementations are correct. A good way to do this is to use your tree to perform some operations, and then check if the outcome is correct. This is best done using a program, rather than doing it manually every time.

In the `Main` class, write code that will automatically perform some operations on your tree implementations, to check if they are correct. Here are some suggestions:

- A simple test: perform some operations on the trees, then print the output using `TreePrinter` and manually compare it to what you expect the output to be.
- A more complex test: A Binary Search Tree (BST) implementation has been provided. Write a method to compare the structure and contents of your AVL/Splay tree with a BST that represents the correct output.
- More complex again: Create some text files that represent operations to be performed on different types of trees (e.g. I25 to insert 25 into the tree, R13 to remove 13, etc.). Write code to read these files and perform the operations on the trees, then compare the outputs.

In all of the above cases, you need to know what the correct output of your implementation should be. The operations you perform should test all the different types of restructuring that are possible (e.g. for a Splay Tree they should cause zig, zig-zig and zig-zag splays to both sides, and at the root and deeper in the tree).

Improving AVL Tree Efficiency

In this implementation, the height of each node must be recalculated every time it is needed, which in practice makes both the insert(…) and remove(…) methods O(*n*) operations, where *n* is the number of nodes in the tree.

Adjust the implementation of the AVL tree so that each node stores its own height, and these are updated only when necessary (**Hint**: updating the heights of nodes should be no worse than O(*h*) complexity following an insert(…) or remove(…) operation, where *h* is the height of the tree).

For this task, you must not change the public API of the `AVLTree` class. All your code must be inside the `AVLTree.java` file.

Submission

This is a **pair programming assignment**. Therefore all code must be written by you and/or your partner. Refer to the UCD Plagiarism Policy for more (http://www.ucd.ie/t4cms/RevisedPlagiarismProtocol.pdf).

- All code should be well formatted and well commented to describe what it is trying to do.
- If you write code outside the `Main.java`, `SplayTree.java` and `AVLTree.java` files, it will not be noticed when grading. Write code only in these files.
- Submit a single .zip file to Moodle.
    - This should Include **only** the following files: `Main.java`, `SplayTree.java` and `AVLTree.java`
- Every file submitted must include a comment to say who the two authors of the code are.
- It is not necessary for both team members to submit to Moodle. However, it is your responsibility to make sure that your work has been submitted. If in doubt, submit it yourself also!