COSC1285/2123: Algorithms & Analysis
Laboratory 7

Topic

AVL tree.

Objective

Students who complete this lab should:

- Improve their understanding of rotations and rebalancing in AVL trees.

Introduction

In lectures, we have studied AVL trees and how to do insertion and rotations to maintain the balance of the tree. To help you gain a greater understanding, in this laboratory exercise, you will implement two of these rotations to rebalance the tree, then test your implementation on sample input to convince yourselves that the rotations do maintain tree balance.

Provided Code

Similar to the previous laboratory on BST, `AVLDemo.java` is an interactive command line program that can execute commands to build and print an AVL tree.

The following files are provided:

file	description
`AVLTree.java`	Skeleton code that implements much of the expected functionality of an AVL tree. You are to implement two of the four rotations and complete the insert functionality.
`AVLDemo.java`	Implements an interactive command line program to build and print an AVL tree.

Compile the program using the following command:

```
javac *.java
```

Running your code

The provided skeleton code allows you to dynamically modify the AVL TREE using a interactive command shell. The following commands are available:

- insert <element> – if element is not a duplicate, then creates a node containing the element and insert into the tree. This may cause the tree to rebalance.
- height – prints out the height of the tree.
- print_ascii – prints out tree in ascii.
- quit/end – terminates the program.

As an example, here is a sample output from typing in the insert, print_ascii and height commands:

```
$ java AVLDemo
insert 10
insert 4
insert 3
insert 15
insert 12
insert 20
insert 7
insert 8
insert 14
insert 11
print_ascii

        12
       / \
      /   \
     /     \
    /       \
   /         \
   8         15
  / \       / \
 /   \     /   \
 4    10   14   20
/ \    \
3 7    11

height

Height = 3

end
```

Task

In AVLDemo.java, we have implemented two rotations, *left* (method `leftRotation()`) and *right-left* (method `rightLeftRotation()` rotations and when they are used (method `insert()`). Study these, and use them to help you implement the remaining two rotations, in addition to completing the implementation of the insert method() to rebalance the tree when the inserted element was inserted into the left subtree.

- Right rotation (method `rightRotation()`).

- Left-right rotation (method `leftRightRotation()`).
- Implement the case where insert key is less than current root node's key (see method `insert()`).

Testing your code

Use the interactive shell to insert different values into an AVL tree, to convince yourself that the AVL tree is working and help you study how it works. We have provided you with one sample test file `test01.in` and the corresponding expected output `test01.exp` to test your program.

First run your program reading the inputs from stdin and write the output to file using the following command:

```
java AVLDemo < test01.in > test01.out
```

Then compare your output to the expected output:

```
diff test01.exp test01.out
```