Data Structures and Algorithms Spring 2019
Assignment 3

Due: 3/20/2019 on Canvas
(100 points)

Instructions

Submit your answers and code in Canvas. Include a brief README file explaining your code, especially if you implemented some of the suggestions for extra credit.

Question [100pt]

In this assignment you will implement AVL trees. You are responsible for implementing insertion and deletion, so you are also responsible for rotating subtrees to maintain the balance property, find the minimum, and a few additional auxiliary methods.

In `assignment3.zip`, we are providing you with the following code:

```
avl.cpp -> your AVL implementation, mostly dummy text
avl.hpp -> header file, you do not have to change it
```

avl.cpp already has a couple methods implemented that you **should not change**:
• a method to print trees, and
• a main method that reads test cases (sequences of operations) and executes them

A sample README file is also provided. You may find how to compile and test the code.

We will test your implementation with test cases that spell out operations (insert a number, delete a number, and print) to be executed on an (initially) empty AVL tree. For example,

```
insert 100
insert 30
insert 20
insert 50
print
delete 30
print
insert 990
insert 900
print
```

means that you should insert 100, 30, 20, 50 and print the resulting AVL tree. Then you should delete 30 and print the resulting AVL tree. Finally, you need to insert 990 and 900 and print the resulting AVL tree. The expected outcome is the following:

```
30
|l 20
|r 100
| |l 50
```

```
50
|l 20
|r 100

50
|l 20
|r 900
| |l 100
| |r 990
```

where the last tree has root 50 and two children: 20 and 900, and node 900 has two children: 100 and 990. Also, 50 is the left child of 100 in the first tree.

A few more notes:
• We will publish more complex test cases three days prior to the submission deadline. I recommend you run the examples we saw in class by yourself.
• Your insertion and deletion must run in O(log(n)), and you must update the height of nodes after each operation.

Opportunity for extra credits (up to 25 points):
Write a program that checks whether a binary search tree is an AVL. The input is an arbitrary binary search tree, and the output is binary, so, either true of false.