**Lab Exercise: AVL Trees**

In this exercise you will implement an AVL tree node and the operations necessary for an AVL tree toolkit.

Note: This is an advanced exercise and it is not expected that you can complete it. You will need to think very thoroughly about how to implement balance checks and rotation operations.

**Exercise 1:**

Suggest a design for a class AVLNode which models a node in an AVL tree – what member variables would the class have? Hint: It is not necessary to calculate or keep track of the *height* of a tree, only the *balance* of each node. See also Exercise 3 below and the hint on the next page.

**Exercise 2:**

Design, implement and test *single* rotations of nodes in an AVL tree. Remember to check for non-existing nodes/children.

When you do, you must also update the *balance* of the rotated nodes. See the next page for hints.

**Exercise 3:**

Implement and test *double* rotations of nodes in an AVL tree. Hint: a double rotation consists of two single rotations. As before, remember to check for non-existing nodes/children and update balances.

**Exercise 4:**

Suggest member operations in an AVL tree class, AVLTree. For each method, state it’s pre- and postconditions. Hints: Are there any methods that you can reuse from – or be inspired by in – your design of the Binary Search Tree? How will you use your implementations of rotations from Exercises 2 and 3 above?

**Exercise 5 (advanced, optional):**

Implement the AVL tree operations in the following order: search(), insert() and remove(). Test your implementations of one method before you continue with the next ones.

**About rotations and balance updates**

When nodes are rotated, their balance may change, obviously. Therefore, the balance for each of the rotated nodes (not their subtrees) must be updated.

The formulae for the update of the balances are documented at e.g. <https://cs.stackexchange.com/questions/48861/balance-factor-changes-after-local-rotations-in-avl-tree>, from which the below is derived. The reader is strongly encouraged to check the link and follow the derivision of the formulae – it is actually quite intuitive.

To calculate the new balances after rotations, the below can be used. Note the following:

* The “balance” of a node is a signed integer for which negative values means left-heavy, 0 is fully balanced, positive values means right-heavy.
* bal(A) is the balance of node A before the rotation, bal(A)’ is the balance of node A after the rotation.
* In all cases, the balance of subtrees a, b, and c remain unchanged.

To calculate the new balances after a *single left* rotation, assume we have the following case:

A B

/ \ / \

/ \ / \

a B ==> A c

/ \ / \

/ \ / \

b c a b

The new balances for a single left rotation of the nodes A and B are given by the following equations:

bal(A)’ = bal(A) - 1 - max(bal(B), 0)

bal(B)’ = bal(B) - 1 + min(bal(A)’, 0)

Let's calculate new balances after a *single right* rotation:

A B

/ \ / \

/ \ / \

B a ==> c A

/ \ / \

/ \ / \

c b b a

The new balances for a single right rotation of the nodes A and B are given by the following equations:

bal(A)’ = bal(A) + 1 - min(bal(B), 0)

bal(B)’ = bal(B) + 1 + max(bal(A)’, 0)