# LAB 7: BINARY SEARCH TREES AND AVL TREES

F27SG – SOFTWARE DEVELOPMENT 3 (5 MARKS)

The topics of this lab are **binary search trees** and **AVL trees**. Download Lab7.zip from Vision and import the project into Eclipse. Your task in the lab is to complete the implementation for this project. Read the source code and make sure you understand what is going on and what is missing. Before you start, you will need to download this and import it into Eclipse:

*File -> Import -> Existing Projects into Workspace*

Then select the project you downloaded. The project is organized as follows:

- The src directory contains all the source files
  - BinarySearchTree.java
  - AVLTree.java
- The test directory contains the unit tests for the project.

## 1. IMPLEMENT INORDER TREE TRAVERSAL (2 POINTS)

In lecture 12, we did an exercise where you had to give the pre-order, post-order and in-order traversals of a Binary Search Tree. In this task, you should implement a method that performs **in-order traversal** by printing out the number in the correct order. We have provided a method in the **BinarySearchTree.java**

```
public void inOrderTraversal(){
        if (rootNode != null)
        rootNode.inOrderTraversal();
}
```

In the nested **BSTNode** class, we have provided an empty method:

```
public void inOrderTraversal(){

}
```

Your task is to complete this method by implementing a recursive **in-order traversal** that prints the value of the node. A main method has been provided and when running it, the following should be printed.

```
******* Tree 1 : 3 nodes ***********
 1
 2
 3
 4
******* Tree 2 : 1 node ***********
 1

******* Tree 3 : empty ***********
```

## 2. A BETTER TRAVERSAL (2 POINTS)

The method from Part 1 isn't particularly useful beyond debugging, and it would be much more useful to be able to return the traversal for use in the program. In the task, a doubly linked list (**DLinkedList.java**) implementation for integers has been provided. In **BinarySearchTree.java** there is an empty method:

```
public DLinkedList returnOrderTraversal(){
        return null; // dummy your code goes here.
}
```

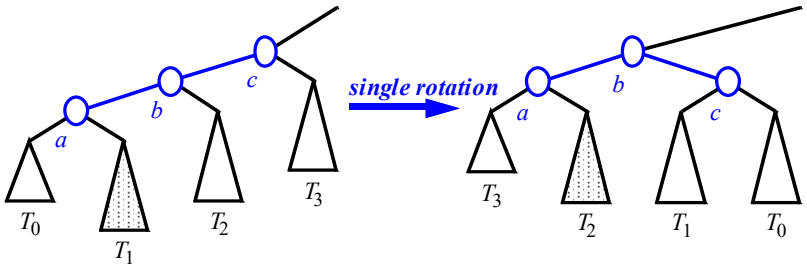
Your task is to complete this method such that a doubly linked list is returned with elements added in a in-order traversal. Your method should be recursive and you may need to implement a suitable method in the **BSTNode** class (comparable to how this was done in part 1). Test methods have been provided as Junit test. These should succeed.

## 3. BALANCING AVL TREES [CHALLENGING] (1 POINT)

Your task is to implement the two single rotations of AVL trees which we discussed in lecture 14. This should be completed in class **AVLTree.java**. The first task is to complete the right rotation method:

```
private AVLNode rotateRight(AVLNode c){
}
```

This should complete the following rotation



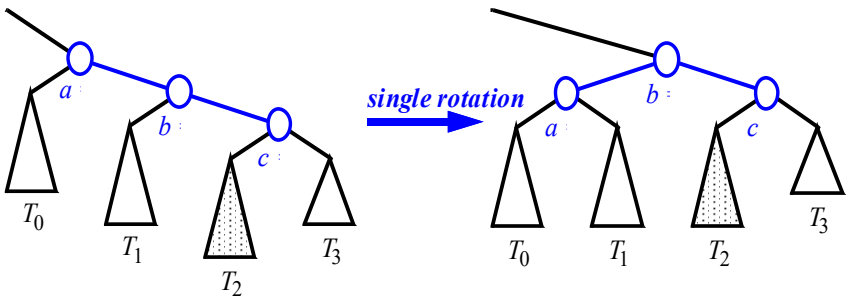

and return the new node that is the root of this subtree (i.e. the node labelled by B). Note that node **c** is given as argument. Remember to re-compute the height of **b** and **c** after this rotation, before you return the **b** node.

Next the following method:

```
private AVLNode rotateLeft(AVLNode a){

}
```

should do the following rotation



Here, node **a** is given as argument and you should return node **b.** Here, you need to re-compute the height of **a** and **b**. A main method which prints a tree out has been provided in the class so that you can test your code. Remember that the balance for each node should be -1,0, or 1.

## 4. DELETION OF AVL TREES [CHALLENGING & OPTIONAL]

Implement AVLTree deletion as discussed in lecture 14. An empty method has been provided in **AVLTree.java**.