Assignment 02

Deadline: **Wed. 20.11.2024, 23:59**
Submission via: **Moodle**

Time log: Please remember the time you needed to solve this assignment and log it in the feedback form in Moodle! This information is fully anonymous.

Balanced Trees

1. AVL tree — 20 points

Implement the **insertion and removal** of nodes in an AVL tree according to the procedure presented in lecture and exercise. Use the provided skeletons of the classes `AVLTree` and `AVLNode` (for Java as well the interface `IAVLTree`).

The following methods need to be implemented in the class `AVLTree` (we are aware that these method names do not conform to the Python style guide (PEP 8) but kindly ask you to use them as-is for assignment conformance across CS and AI courses):

`getTreeRoot`	...	This method returns the root of the AVL tree.
`getTreeHeight`	...	returns the current height of the AVL tree in *O(1)*.
`getSize`	...	returns the number of nodes in the tree.
`insertNode`	...	inserts a new `AVLNode` into the tree, given a key and a value. Duplicate keys are not allowed. Values must be >= 0. Return `true` on success, `false` otherwise. This function is partially implemented. The BST insert is complete, your task is to update the heights, check for AVL integrity and restructure the tree if needed.
`removeNode`	...	removes an `AVLNode` based on a given key and returns `true` on success, `false` otherwise. This function is partially implemented. The BST remove is complete, your task is to update the heights, check for AVL integrity and restructure the tree if needed.

When searching for the nodes *x*, *y*, and *z* (see corresponding slides of exercise 02), go upwards from the node you just inserted. For this purpose, **each** `AVLNode` stores a reference to its parent node. Furthermore, each `AVLNode` contains the field `height` to store the height of the node within the AVL tree.

Hints:

- For inserting and removing nodes an auxiliary function `restructure()` might be useful, which performs a **restructuring** (if necessary) starting from a given node ***n*** upwards. This method is called after the insertion or removal of a node.

  Consider that restructuring can also violate the balancing on higher levels of the AVL tree!

- Think about further auxiliary methods that improve the readability of your code. For example, if you implement the function `restructure` mentioned above, an additional function to check if a given (sub)tree is balanced might be useful.

- To achieve a query of the height in *O(1)*, an update of the heights of all affected nodes must be made when a node has been inserted or removed.

- You are free to declare and implement further data structures and methods as you need them, such as e.g., a class that manages *x, y,* and *z* nodes for restructuring, …

Submission:
For this assignment, please submit your `AVLTree` source file. In case you have implemented further code outside of this class/source file, submit that as well. The `AVLNode` class as well as the `IAVLTree` interface (for Java) **must not** be changed and therefore don't need to be submitted.