JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 2
Winter term 2021

Assignment 02

Deadline: **Wed. 24.11.2021, 23:59**
Submission via: **Moodle**

Time log

Remember the time you needed to implement your solution of this assignment and log it in the exercise-specific survey in Moodle! This information is fully anonymous.

Balanced Trees

1. AVL tree 24 points

Implement the **insertion and removal** of nodes in an AVL tree according to the *cut&link* procedure presented in lecture and exercise. Use the provided skeletons of the classes `AVLTree` and `AVLNode`.

The following methods need to be implemented in the class `AVLTree`:

`get_tree_root` … This method returns the root node of the AVL tree.
`get_tree_height` … returns the current height of the AVL tree in *O(1)*.
`get_tree_size` … returns the number of nodes in the tree.
`to_array` … return the tree's values in form of a `float`-array (pre-order)
`find_by_key` … searches for a given key and returns the corresponding node's value if found, otherwise `None`.
`insert` … inserts a new `AVLNode` into the tree, given a key and a value. Duplicate keys are not allowed. Return `true` on success, `False` otherwise.
`remove_by_key` … removes an `AVLNode` based on a given key and returns `True` on success, `False` otherwise.

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 variable `height` to store the height of the node within the AVL tree.

Please note that the `AVLNode` class as well as the given class skeleton for the `AVLTree` **must not** be changed.

Hints:

- For inserting and removing nodes an auxiliary function `restructure()` might be useful, which performs a one-time **Cut&Link 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,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.