JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 1
Summer term 2021
Dari Trendafilov
Stefan Grünberger

Assignment 04

Deadline: **20.05.2021, 23:59**
Submission via: **Moodle**

Elaboration time

*Remember the time you need for the elaboration of this assignment and document it in the file **time.txt** according to the structure illustrated in the right box. Please do not pack this file into an archive but upload it as a **separate file**.*

```
#Student ID
K12345678
#Assignment number
04
#Time in minutes
190
```

Trees

As usual, don't change the given interface, but you can add auxiliary methods and reuse code where possible. Submit your **bst.py** and **tree_node.py** implementations.
NOTE: Add all necessary code-files in the top-level of the submitted zip.

1. Binary Search Tree — 24 points

Implement the `BinarySearchTree` class in **bst.py** and the `TreeNode` class in **tree_node.py**, using the provided skeletons.

```python
class BinarySearchTree:

    def insert(self, key: int, value: Any) -> None:
        """Insert a new node into BST.
            key (int): Key which is used for placing the value into the tree.
            value (Any): Value to insert."""

    def find(self, key: int) -> TreeNode:
        """Return node with given key."""

    def size(self) -> int:
        """Return number of nodes contained in the tree."""

    def remove(self, key: int) -> None:
        """Remove a node with given key, maintaining BST-properties."""

    def inorder(self, node: TreeNode = None) -> Generator[TreeNode, None, None]:
        """Yield nodes in inorder."""

    def preorder(self, node: TreeNode = None) -> Generator[TreeNode, None, None]:
        """Yield nodes in preorder."""

    def postorder(self, node: TreeNode = None) -> Generator[TreeNode, None, None]:
        """Yield nodes in postorder."""

    def is_valid(self) -> bool:
       """Return if the tree fulfills BST-criteria."""

    def return_min_key(self) -> TreeNode:
        """Return the node with the smallest key."""

    def find_comparison(self, key: int) -> Tuple[int, int]:
       """Create an inbuilt python list of BST values in preorder and compute the number of comparisons
       needed for finding the key both in the list and in the BST.
       Return the numbers of comparisons for both, the list and the BST."""

    def height(self) -> int:
        """Return height of the tree."""

    def is_complete(self) -> bool:
        """Return if the tree is complete."""


class TreeNode:

    def depth(self) -> int:
        """Return depth of the node, i.e. the number of parents/grandparents etc.

    def is_internal(self) -> bool:
        """Return if node is an internal node."""
```

JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 1
Summer term 2021
Dari Trendafilov
Stefan Grünberger

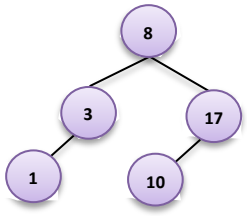
Assignment 04

Deadline: **20.05.2021, 23:59**
Submission via: **Moodle**

Notes on `find_comparison()`:

Example of the **number of comparisons** needed to find a key in a BST.

List = **[**8, 17, 10, 3, 1**]** → BST =



Searching for key '3' requires 4 comparisons for the list, but only 3 comparisons (1. 3==8; 2. 3<8 go left or right, 3. 3==3) for the BST.