Institute of Pervasive Computing

Algorithms and Data Structures 1
Summer term 2024
Jäger, Rihani, Vázquez Pufleau, Anzengruber-Tanase

Assignment 4

Deadline: **Thu. 09.05.2024, 12:00**
Submission via: **Moodle**

Elaboration time

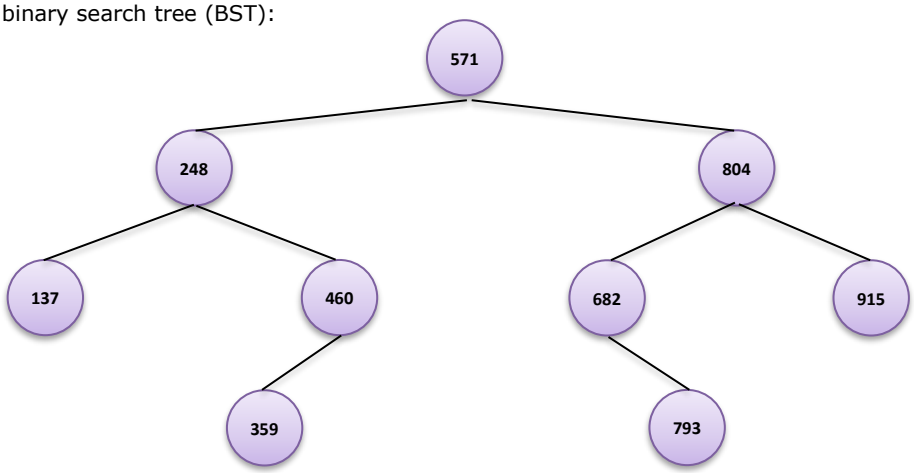
Remember the time you need for the elaboration of this assignment and document it in Moodle.

Trees

For this assignment, please submit the pdf of the pen-and-paper work and the source code of your **`bst.py`** and **`tree_node.py`** implementations. As usual, don't change the given interface, but you can add auxiliary methods and reuse code where possible.
NOTE: Add all necessary code-files in the top-level of the submitted zip.

1. Binary Search Tree – Insert and remove — 2+2+2 points

Given is this binary search tree (BST):



Apply the following operations in this order:

1. **Insert** a new node with the number formed by the last three digits of your student ID.
   Example: For the student ID k24680**123**, the inserted node should use the key **123** (leading zeros are ignored). Describe the required comparisons and draw the BST after the insertion.
2. **Remove** the root node (with the key **571**) from the BST after you inserted the new node from (1).
   Explain what changes have to be applied to the BST, e.g., which node becomes the new root node, how to rearrange the old parent-child-connections to maintain the order, etc. Draw the BST after the removal.
3. **Insert** a new node with the number formed by the first three digits of your student ID.
   Example: For the student ID k**246**80123, the inserted node should use the key **246** (leading zeros are ignored). Describe the required comparisons and draw the BST after the insertion.

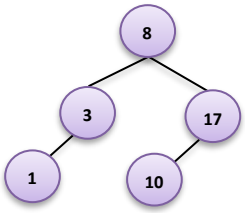
2. Binary Search Tree – Implementation — 18 points

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

Notes on `find_comparison()`:

This method shall create a BST from a given linear list (**without duplicates**) and analyze the runtime for the search of a key. To do this, determine the **number of comparisons** needed to find the key in both, the BST (key comparison + comparison for branch decision) as well as the linear list (key comparison). Example of the **number of comparisons** needed to find a key in a BST:

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



Assignment 4

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.

For your runtime tests also consider analyzing **large lists** and search for the **last inserted element** (= worst case for linear list) to see the significantly better performance of the BST. To evaluate the runtime and number of comparisons, you can run the unit tests `test_runtime_comparison_check_bst_with_pregen_list` and `test_runtime_comparison_check_list_with_pregen_list`.

```
class BinarySearchTree:
    """Binary-Search-Tree implemented for didactic reasons."""

    def insert(self, key: int, value: Any) -> None:
        """Insert a new node into BST."""

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

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

    def remove(self, key: int) -> None:
        """Remove 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."""

    @property
    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 return_max_key(self) -> TreeNode:
        """Return the node with the largest key. """
 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."""
```

```
class TreeNode:

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

    @property
    def is_external(self) -> bool:
        """Return if node is an external node."""

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