Weekly Lab

AVL Tree

In this lab session, we will implement a self-balancing binary search tree: AVL Tree. Source code needs to be distributed in a single file named **StudentID.py**.

Each node in the AVL Tree should be defined with the following structure:

```python
class Node:
    def __init__(self, key: int):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1
```

Exercise 1: Basic AVL Tree Operations

Implement the following basic functions and operations on the AVL Tree as follows:

1. Create a new Node from a given value.

   `def new_node(data: int) -> Node`

2. Insert a new value into an AVL Tree (do not add existing keys).

   `def insert_node(root: Node, data: int) -> Node`

3. Search a Node with a given value from an AVL Tree. Return `None` if not found.

   `def search_node(root: Node, data: int) -> Node`

4. Delete a Node with a given value from an AVL Tree.

   `def delete_node(root: Node, data: int) -> Node`

5. Traversal in **Pre-order**, **In-order**, **Post-order** and **Level-order**.

   `def nlr(root: Node) -> None`

   `def lnr(root: Node) -> None`

   `def lrn(root: Node) -> None`

   `def level_order(root: Node) -> None`

Exercise 2: Additional AVL Tree Problems

With the AVL Tree implemented in Exercise 1, solve the following problems:

1. Check if the given AVL Tree is a full tree.

   ```
   def is_full(root: Node) -> bool
   ```

2. Check if the given AVL Tree is a complete tree.

   ```
   def is_complete(root: Node) -> bool
   ```

3. Check if the given AVL Tree is a perfect tree.

   ```
   def is_perfect(root: Node) -> bool
   ```

4. Find all nodes in the tree with 2 child nodes, and the left child is a divisor of the right child. Print them to the console in ascending order.

   ```
   def print_special_nodes(root: Node) -> None
   ```

5. Find the Least Common Ancestor (LCA) for any two given nodes in AVL Tree.

   ```
   def find_common_ancestor(root: Node, x: int, y: int) -> int
   ```

Note:
You are free to define any helper functions if necessary.
You may test your code in a `main` function with hardcoded examples. Suggested test sequence:

- Initialize an empty AVL Tree.
- Insert the values: 8, 6, 5, 7, 10, 9.
- Display the tree using Pre-order, In-order, and Post-order traversals.
- Delete the node with key 8. Then display the tree using Level-order traversal.
- Solve the problems in Exercise 2.

The end.