Lab 9

Task 1: AVL

You have to implement the AVL ADT using link list data structure.

we have the following Node and AVLclass.

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

```
class AVL:
    def __init__(self):
        self.root = None


    def height(self, node):
    if not node:
        return 0
    return node.height


    def insert_value(self, value):
       self.root = self.insert(self.root, value)


    def insert(self, root, value):
      #Impelment the following method.

    def balance_factor(self, node):
      #Impelment the following method.

    def rotation(self, node):
      #Impelment the following method.

    def rotation_ll(self, node):
      #Impelment the following method.

    def rotation_lr(self, node):
      #Impelment the following method.

    def rotation_rr(self, node):
      #Impelment the following method.

    def rotation_rl(self, node):
      #Impelment the following method.


    def inorder(self, p):
        if p.left:
            self.inorder(p.left)
        print(p.info, end=" ")
        if p.right:
            self.inorder(p.right)
    def inorder_traversal(self):
        if self.root:
            self.inorder(self.root)

    def getHeight(self):
        return self.calculateHeight(self.root)

    def calculateHeight(self, root):
        if root is None:
            return 0
        lh = self.calculateHeight(root.left)
        rh = self.calculateHeight(root.right)
        return max(lh, rh) + 1

#Test your code using the following driver code

# Creating AVL and inserting random values
tree = AVL()
for _ in range(500):
    val = random.randint(10, 2000)
    tree.insert_value(val)

print("Height:", tree.getHeight())
print("\nIn Order:\t", end="")

tree.inorder_traversal()
```

Task 2:

Add the following method in BST ADT which you implemented in the Last Lab.

```
def construct_from_traversals(self, in_order, pre_order):
```

The method **construct_from_traversals(self, in_order, pre_order)** in the BST class is used to construct a binary search tree from its in-order and pre-order traversals.

Let us consider the below traversals:

- Inorder sequence: D B E A F C
- Preorder sequence: A B D E C F

In a Preorder sequence, the leftmost element is the root of the tree. So we know 'A' is the root for given sequences. By searching 'A' in the Inorder sequence, we can find out all elements on the left side of 'A' is in the left subtree and elements on right in the right subtree. So we know the below structure now.

```
         A
       /   \
     /       \
   D B E      F C
```

We recursively follow the above steps and get the following tree.

```
         A
       /   \
     /       \
    B         C
   / \        /
 /     \   /
D       E  F
```

Driver

```python
# Example 1 usage:
bst = BinarySearchTree()
in_order = [ 'D' ,'B' ,'E' ,'A' ,'F' ,'C' ]
pre_order = ['A' ,'B' ,'D' ,'E' ,'C' ,'F']

bst.root = bst.construct_from_traversals(in_order, pre_order)

# Verify constructed tree
print("In-order traversal of constructed BST:")
bst.display_in_order()

# Verify constructed tree
print("Post-order traversal of constructed BST:")
bst.display_post_order()


# Example 2 usage:
bst2 = BinarySearchTree()
in_order = [ 5, 10, 15, 25, 27, 30, 35, 40, 45, 50, 52, 55, 60, 65, 70,
75, 80, 85, 90, 100]
pre_order = [50, 25, 10, 5, 15, 40, 30, 27, 35, 45, 75, 60, 55, 52, 65,
70, 90, 80, 85, 100]

bst2.root = bst2.construct_from_traversals(in_order, pre_order)

# Verify constructed tree
print("In-order traversal of constructed BST:")
bst2.display_in_order()

# Verify constructed tree
print("Post-order traversal of constructed BST:")
bst2.display_post_order()
```

The output of the following driver should be:

```
In-order traversal of constructed BST:
D B E A F C
Post-order traversal of constructed BST:
D E B F C A
In-order traversal of constructed BST:
5 10 15 25 27 30 35 40 45 50 52 55 60 65 70 75 80 85 90 100
Post-order traversal of constructed BST:
5 15 10 27 35 30 45 40 25 52 55 70 65 60 85 80 100 90 75 50
```