Working with Binary Search Trees: Takeaways

Syntax

- Implementing an AVL tree:

```
class AVLTree(BST):
    def __init__(self):
        super().__init__()
    def _add_recursive(self, current_node, value):
        if current_node is None:
            return AVLNode(value)
        if value <= current_node.value:
            current_node.left_child = self._add_recursive(current_node.left_child, value)
        else:
            current_node.right_child = self._add_recursive(current_node.right_child, value)
        current_node.calculate_height_and_imbalance()
        if abs(current_node.imbalance) == 2:
            return self._balance(current_node)
        return current_node
    def get_height(self):
        return self.root.height
    def _rotate_left(self, node):
        pivot = node.right_child
        node.right_child = pivot.left_child
        pivot.left_child = node
        node.calculate_height_and_imbalance()
        pivot.calculate_height_and_imbalance()
        return pivot
    def _rotate_right(self, node):
        pivot = node.left_child
        node.left_child = pivot.right_child
        pivot.right_child = node
        node.calculate_height_and_imbalance()
        pivot.calculate_height_and_imbalance()
        return pivot
    def _balance(self, node):
        if node.imbalance == 2:
            pivot = node.left_child
            if pivot.imbalance == 1:
                return self._rotate_right(node)
            else:
                node.left_child = self._rotate_left(pivot)
                return self._rotate_right(node)
```

```
        else:
            pivot = node.right_child
            if pivot.imbalance == -1:
                return self._rotate_left(node)
            else:
                node.right_child = self._rotate_right(pivot)
                return self._rotate_left(node)
```

Concepts

- AVL trees are an implementation of binary search trees that are automatically balanced to ensure the efficiency of the tree operations.
- The height of an AVL three is *O(log(N))*. Therefore, all AVL tree methods have this complexity. This makes them much more suited to query data than a list.

Resources

- AVL Tree Deletions
- AVL Tree
- Tree Rotations
- Using assertions
- Unit testing