Working with Binary Search Trees: Takeaways

Syntax

- Implementing a binary search tree (BST):

```
class BST:
    def __init__(self):
        self.node = None
    def insert(self, value=None):
        node = Node(value=value)
        if not self.node:
            self.node = node
            self.node.right = BST()
            self.node.left = BST()
            return
        if value > self.node.value:
            if self.node.right:
                self.node.right.insert(value=value)
            else:
                self.node.right.node = node
            return
        if self.node.left:
            self.node.left.insert(value=value)
        else:
            self.node.left.node = node
    def inorder(self, tree):
        if not tree or not tree.node:
            return []
        return (
            self.inorder(tree.node.left) +
            [tree.node.value] +
            self.inorder(tree.node.right)
    )
```

- Searching a BST:

```
class BST(BaseBST):

    def search(self, value):
        if not self.node:
            return False
        if value == self.node.value:
            return True
        result = False
        if self.node.left:
            result = self.node.left.search(value)
        if self.node.right:
            result = self.node.right.search(value)
    return result
```

- Performing rotation operations:

```
class BST(BaseBST):

    def left_rotate(self):

        old_node = self.node

        new_node = self.node.right.node

        if not new_node:

            return

        new_right_sub = new_node.left.node

        self.node = new_node

        old_node.right.node = new_right_sub

        new_node.left.node = old_node

    def right_rotate(self):

        old_node = self.node

        new_node = self.node.left.node

        if not new_node:

            return

        new_left_sub = new_node.right.node

        self.node = new_node

        old_node.left.node = new_left_sub

        new_node.right.node = old_node
```

Concepts

- A BST provides the ability to run efficient range queries on data sets. A BST requires the following conditions to hold:
    - Every value in a nodes' left sub-tree has a value that is less than or equal to the parent node.
    - Every value in a nodes' right sub-tree has a value that is greater than or equal to the parent node.
- Every new node in a BST is inserted in sorted order.
- Searching for an item in a balanced binary tree has time complexity of . On the other hand, searching for an item in an unbalanced binary tree has time complexity .

- A BST that stays balanced for every insert is called a self-balancing BST.
- A tree rotation operation involves changing the structure of the tree while maintaining the order of the elements.

Resources

- Binary Search Tree
- Tree Rotations