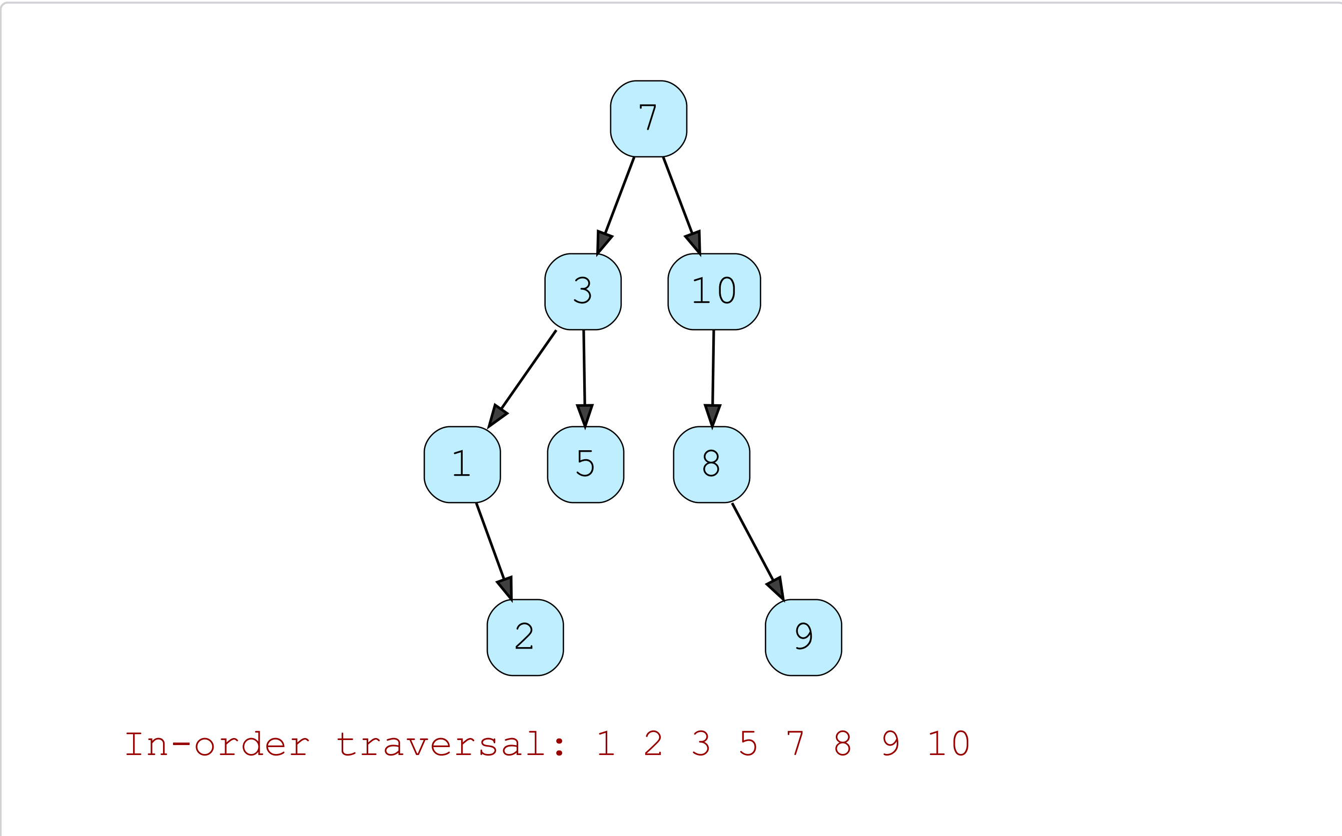
Solution Review: Checking the BST property

This lesson contains the solution review for Checking the BST property challenge.

We'll cover the following

- Implementation
- Explanation

Recall the in-order traversal that we learned in the Binary Trees chapter. The in-order traversal of a Binary Search Tree gives us the list of nodes in sorted order.



In the code widget below, we have implemented the in-order traversal for `BST` and you can confirm the traversal in the illustration above.

```
class Node(object):
```

```python
  def __init__(self, data):
    self.data = data
    self.left = None
    self.right = None


class BST(object):
    def __init__(self, root):
        self.root = Node(root)

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
        else:
            self._insert(data, self.root)

    def _insert(self, data, cur_node):
        if data < cur_node.data:
            if cur_node.left is None:
                cur_node.left = Node(data)
                cur_node.left.parent = cur_node
            else:
                self._insert(data, cur_node.left)

        elif data > cur_node.data:
            if cur_node.right is None:
                cur_node.right = Node(data)
                cur_node.right.parent = cur_node
            else:
                self._insert(data, cur_node.right)
```

We have discussed in-order traversal above because we'll be using a similar idea to check whether a tree satisfies the BST property or not. If we traverse a binary tree in-order and it results in a sorted list, then the tree satisfies the BST property.

Implementation

Now let's discuss the implementation of the solution we provided for the challenge in the previous lesson.

```python
def is_bst_satisfied(self):
    def helper(node, lower=float('-inf'), upper=float('inf')):
        if not node:
            return True

        val = node.data
        if val <= lower or val >= upper:
            return False

        if not helper(node.right, val, upper):
```

```
            return False
        if not helper(node.left, lower, val):
            return False

        return True

    return helper(self.root)
```

Explanation

In the `is_bst_satisfied` method, we define an inner method on **line 2**, `helper`, which takes `node`, `lower` and `upper` as input parameters. On **line 3**, we have the base case which caters to an empty tree or a `None` node. If `node` is `None`, `True` is returned from the method on **line 4**. Otherwise, the execution proceeds to **line 6** where `val` is made equal to `node.data`.

Next, we check if `val` is less or equal to `lower` or if `val` is greater or equal to `upper` on **line** 7. If any of the two conditions is `True`, `False` is returned from the method on **line 8**. This is because the value of the current node should be greater than all the values of the children in the left subtree, and it should be less than all the values of the children in the right subtree.

Now that we have checked the BST property for the current node, it's time to check it for the subtrees. On **line 10**, we make a recursive call to the right subtree of the current node. `node.right` is passed as `node`, `val` is passed as `lower` while `upper` stays the same. `lower` is now the lower bound for the right subtree as all the children in the right subtree have to be greater than the value of the current node. If the recursive call returns `False`, the condition on **line 10** will evaluate to `True` and `False` will be returned from the method.

Similarly, the left subtree is evaluated through a recursive call on **line 12**. Now `val` is passed as `upper` for the recursive call as all the children in the left subtree have to be less than the value of the current node.

If none of the conditions before **line 14** evaluate to `True`, `True` is returned on **line 14** declaring that the BST property is satisfied.

You can run the following code where we have the entire implementation of the `BST` class that we discussed in this chapter.

```
class Node(object):
  def __init__(self, data):
    self.data = data
    self.left = None
```

```
        self.left = None
        self.right = None


class BST(object):
    def __init__(self, root):
        self.root = Node(root)

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
        else:
            self._insert(data, self.root)

    def _insert(self, data, cur_node):
        if data < cur_node.data:
            if cur_node.left is None:
                cur_node.left = Node(data)
                cur_node.left.parent = cur_node
            else:
                self._insert(data, cur_node.left)

        elif data > cur_node.data:
            if cur_node.right is None:
                cur_node.right = Node(data)
                cur_node.right.parent = cur_node
            else:
                self._insert(data, cur_node.right)
        else:
            print("Value already in tree!")

    def inorder_print_tree(self):
        if self.root:
            self._inorder_print_tree(self.root)

    def _inorder_print_tree(self, cur_node):
        if cur_node:
            self._inorder_print_tree(cur_node.left)
            print(str(cur_node.data))
            self._inorder_print_tree(cur_node.right)

    def find(self, data):
        if self.root:
            is_found = self._find(data, self.root)
            if is_found:
                return True
            return False
        else:
            return None

    def _find(self, data, cur_node):
        if data > cur_node.data and cur_node.right:
            return self._find(data, cur_node.right)
        elif data < cur_node.data and cur_node.left:
            return self._find(data, cur_node.left)
        if data == cur_node.data:
            return True

    def is_bst_satisfied(self):
        def helper(node, lower=float('-inf'), upper=float('inf')):
            if not node:
                return True
```

```
            val = node.data
            if val <= lower or val >= upper:
                return False

            if not helper(node.right, val, upper):
                return False
            if not helper(node.left, lower, val):
                return False
            return True

        return helper(self.root)

bst = BST(4)
bst.insert(2)
bst.insert(8)
bst.insert(5)
bst.insert(10)

tree = BST(1)
tree.root.left = Node(2)
tree.root.right = Node(3)
tree.root.left.left = Node(4)
tree.root.left.right = Node(5)
tree.root.right.left = Node(6)
tree.root.right.right = Node(7)
tree.root.right.right.right = Node(8)

print(bst.is_bst_satisfied())
print(tree.is_bst_satisfied())
```

Congratulations! We have completed the Data Structures part of the course. Now, we'll focus on algorithms in the remaining course. I hope you were able to enjoy learning about data structures. Happy learning!