Implement a B-Tree data structure: Takeaways

Syntax

- B-tree implementation:

```
class BTree:
    def __init__(self, split_threshold):
        self.root = Node()
        self.split_threshold = split_threshold
        self.height = 0
        self.size = 0
    def __len__(self):
        return self.size
    def _find_node(self, current_node, key):
        if current_node.contains_key(key):
            return current_node
        if current_node.is_leaf():
            return None
        child_index = current_node.get_insert_index(key)
        return self._find_node(current_node.children[child_index], key)
    def contains(self, key):
        node = self._find_node(self.root, key)
        if node is None:
            return False
        return True
    def _add(self, current_node, key, value):
        if current_node.is_leaf():
            current_node.insert_entry(key, value)
        else:
            child_index = current_node.get_insert_index(key)
            self._add(current_node.children[child_index], key, value)
        if len(current_node) > self.split_threshold:
            parent = current_node.split()
            if current_node == self.root:
                self.root = parent
                self.height += 1
    def add(self, key, value):
        self._add(self.root, key, value)
        self.size += 1
    def get_value(self, key):
        node = self._find_node(self.root, key)
        if node is None:
```

```
            return None
        return node.get_value(key)
```

Concepts

- A B-tree is an ordered dictionary. It is less efficient than a plain dictionary, but the order can be leveraged to perform other types of queries.
- B-tree operations have logarithmic time complexities *O(log(N))* where *N* is the number of entries.

Resources

- B-Tree