```
from typing import List, Optional, Any


class BSTNode:
    """
    Binary Search Tree Node.
    Attributes:
        key: The key value of the node.
        values: A list of values associated with the key.
        left: The left child node.
        right: The right child node.
    Methods:
        add_value(value: Any) -> None:
            Adds a value to the list of values associated with the
key.
        get_values_count() -> int:
            Returns the number of values associated with the key.
    """
    def __init__(self, key: Any):
        self.key: Any = key
        self.values: List[Any] = []
        self.left: Optional['BSTNode'] = None
        self.right: Optional['BSTNode'] = None

    def add_value(self, value: Any) -> None:
        """
        Adds a value to the node.

        Parameters:
            value (Any): The value to be added.

        Returns:
            None
        """
        self.values.append(value)


    def get_values_count(self):
        """
        Returns the number of values stored in the node.

        Returns:
            int: The number of values stored in the node.
        """
        return len(self.values)
```

```
—--------------------------------


from typing import Optional, Any, List, Generator

from indexer.abstract_index import AbstractIndex
from indexer.trees.bst_node import BSTNode

class BinarySearchTreeIndex(AbstractIndex):
    """
    A binary search tree implementation of an index.
    This class represents a binary search tree index, which allows
for efficient insertion, search, and traversal operations.
    It inherits from the AbstractIndex class.

    Methods:
        insert(key: Any, value: Any) -> None:
            Inserts a new node with the given key and value into the
binary search tree.
        search(key: Any) -> List[Any]:
            Searches for nodes with the given key in the binary
search tree and returns their values.
        count_nodes() -> int:
            Counts the number of nodes in the binary search tree.
        tree_height() -> int:
            Calculates the height of the binary search tree.
        get_keys_in_order() -> List[Any]:
            Returns a list of keys in the binary search tree in
ascending order.
        get_leaf_keys() -> List[Any]:
            Returns a list of keys of leaf nodes in the binary search
tree.
        get_avg_value_list_len() -> float:
            Calculates the average length of value lists in the
binary search tree.
    """

    def __init__(self):
        super().__init__()
        self.root: Optional[BSTNode] = None

    def _insert_recursive(self, current_node: Optional[BSTNode], key:
Any, value: Any) -> BSTNode:
        """
```

```
        Recursively inserts a new node with the given key and value
into the binary search tree.
        If key is found, value is added to key's associated list.
        If key is not found, it is added and value is appended as its
first doc.
        Args:
            current_node (Optional[BSTNode]): The current node being
evaluated.
            key (Any): The key of the new node.
            value (Any): The value of the new node.
        Returns:
            BSTNode: The root node of the modified binary search
tree.
        """
        if not current_node:
            node = BSTNode(key)
            node.add_value(value)
            return node
        elif key < current_node.key:
            current_node.left =
self._insert_recursive(current_node.left, key, value)
        elif key > current_node.key:
            current_node.right =
self._insert_recursive(current_node.right, key, value)
        elif key == current_node.key:
            current_node.add_value(value)
        return current_node

    def _search_recursive(self, node: Optional[BSTNode], key: Any) ->
List[Any]:
        """
        Recursively searches for a key in the binary search tree.
Returns the list of docs
        in which the key is found.
        Args:
            node (Optional[BSTNode]): The current node being checked.
            key (Any): The key to search for.
        Returns:
            List[Any]: A list of values associated with the key.
        Raises:
            KeyError: If the key is not found in the tree.
        """

        if node is None:
            return []
```

```
            #raise KeyError(f"Key {key} not found in the tree.")
        if key < node.key:
            return self._search_recursive(node.left, key)
        elif key > node.key:
            return self._search_recursive(node.right, key)
        else:
            return node.values

    def _inorder_traversal_generator(self, node: Optional[BSTNode])
-> Generator[BSTNode, None, None]:
        """
        Generates an inorder traversal of the binary search tree
starting from the given node. Used
        with the __iter__ dunder function.
        Args:
            node (Optional[BSTNode]): The starting node for the
traversal.
        Yields:
            Generator[BSTNode, None, None]: The nodes in the binary
search tree in inorder traversal order.
        """

        if node:
            yield from self._inorder_traversal_generator(node.left)
            yield node
            yield from self._inorder_traversal_generator(node.right)

    def _count_nodes(self, node: Optional[BSTNode]) -> int:
        """
        Recursively counts the number of nodes in the binary search
tree.
        Parameters:
        - node (Optional[BSTNode]): The root node of the binary
search tree.
        Returns:
        - int: The number of nodes in the binary search tree.
        """

        if node is None:
            return 0
        return 1 + self._count_nodes(node.left) +
self._count_nodes(node.right)

    def _tree_height(self, node: Optional[BSTNode]) -> int:
        """
```

```
        Calculate the height of the binary search tree rooted at the
given node.
        Args:
            node (Optional[BSTNode]): The root node of the binary
search tree.
        Returns:
            int: The height of the binary search tree.
        """

        if node is None:
            return 0
        left_height = self._tree_height(node.left)
        right_height = self._tree_height(node.right)
        return 1 + max(left_height, right_height)

    def _get_leaf_keys(self, node: Optional[BSTNode], leaves:
List[Any]) -> None:
        """
        Recursively traverses the binary search tree and appends the
keys of the leaf nodes to the 'leaves' list.
        Args:
            node (Optional[BSTNode]): The current node being
traversed.
            leaves (List[Any]): The list to which the keys of the
leaf nodes will be appended.
        Returns:
            None
        """

        if node is None:
            return
        if node.left is None and node.right is None:
            leaves.append(node.key)
        else:
            self._get_leaf_keys(node.left, leaves)
            self._get_leaf_keys(node.right, leaves)

    def __iter__(self) -> Generator[BSTNode, None, None]:
        """
        Returns an iterator that performs an inorder traversal of the
binary search tree.

        Yields:
            BSTNode: The next node in the inorder traversal.
```

```
        """
        yield from self._inorder_traversal_generator(self.root)

    def insert(self, key: Any, value: Any) -> None:
        """
        Inserts a key-value pair into the binary search tree.

        Parameters:
            key (Any): The key to be inserted.
            value (Any): The value associated with the key.

        Returns:
            None
        """
        self.root = self._insert_recursive(self.root, key, value)

    def search(self, key: Any) -> List[Any]:
        """
        Search for a key in the binary search tree.

        Parameters:
            key (Any): The key to search for.

        Returns:
            List[Any]: A list of values associated with the key. If
the key is not found, an empty list is returned.
        """
        # convert key to lowercase for searching
        key = key.lower()

        return self._search_recursive(self.root, key)

    def count_nodes(self) -> int:
        """
        Counts the number of nodes in the binary search tree.

        Returns:
            int: The number of nodes in the binary search tree.
        """
        return self._count_nodes(self.root)

    def tree_height(self) -> int:
        """
        Calculate the height of the binary search tree.
```

```
        Returns:
            int: The height of the binary search tree.
        """
        return self._tree_height(self.root)

    def get_keys_in_order(self) -> List[Any]:
        """
        Returns a list of keys in the binary search tree in ascending
order.

        Returns:
            List[Any]: A list of keys in ascending order.
        """
        keys: List[Any] = []
        for node in self:
            keys.append(node.key)
        return keys

    def get_leaf_keys(self) -> List[Any]:
        """
        Returns a list of keys of all the leaf nodes in the binary
search tree.

        Returns:
            List[Any]: A list of keys of all the leaf nodes in the
binary search tree.
        """
        leaves: List[Any] = []
        self._get_leaf_keys(self.root, leaves)
        return leaves

    def get_avg_value_list_len(self):
        """
        Calculate the average length of the value lists in the binary
search tree.
        Returns:
            float: The average length of the value lists.
        """
        list_len_sum = 0
        num_keys = 0
        for node in self:
            list_len_sum = list_len_sum + node.get_values_count()
            num_keys = num_keys + 1

        return (list_len_sum / num_keys)
```