EX - 12 BINARY SEARCH TREE

T SADAKOPA RAMAKRISHNAN | 3122225002109 | IT - B

Q1)Provide an implementation of Binary Search Trees with various operations of Insert, Delete, Find, Findmin and Findmax. Use Linked Binary Tree for the implementation.

Code:

```
class Node:
    __slots__ = ['value', 'left_child', 'right_child']

    def __init__(self, value=None):
        """
        Initialize a Node with a given value.

        Args:
            value: The value to be stored in the node.
        """
        self.value = value
        self.left_child = None
        self.right_child = None


class BinarySearchTree:
    def __init__(self):
        """
        Initialize a Binary Search Tree with a root node.
        """
        self.root = None

    def insert(self, value):
        """
        Insert a value into the binary search tree.

        Args:
            value: The value to be inserted.
        """
        if self.root is None:
            self.root = Node(value)
        else:
```

```
            self._insert(value, self.root)

    def _insert(self, value, cur_node):
        """
        Helper method to recursively insert a value into the
binary search tree.

        Args:
            value: The value to be inserted.
            cur_node: The current node being traversed.

        Returns:
            None
        """
        if value < cur_node.value:
            if cur_node.left_child is None:
                cur_node.left_child = Node(value)
            else:
                self._insert(value, cur_node.left_child)
        elif value > cur_node.value:
            if cur_node.right_child is None:
                cur_node.right_child = Node(value)
            else:
                self._insert(value, cur_node.right_child)
        else:
            print("Value already in Tree!")

    def print_tree(self):
        """
        Print the values of the binary search tree in order.

        Returns:
            None
        """
        if self.root is not None:
            self._print_tree(self.root)

    def _print_tree(self, cur_node):
        """
        Helper method to recursively print the values of the
binary search tree in order.
```

```
        Args:
            cur_node: The current node being traversed.

        Returns:
            None
        """
        if cur_node is not None:
            self._print_tree(cur_node.left_child)
            print(str(cur_node.value), end=" ")
            self._print_tree(cur_node.right_child)

    def height(self):
        """
        Calculate the height of the binary search tree.

        Returns:
            The height of the tree.
        """
        if self.root is not None:
            return self._height(self.root, 0)
        else:
            return 0

    def _height(self, cur_node, cur_height):
        """
        Helper method to recursively calculate the height of the
binary search tree.

        Args:
            cur_node: The current node being traversed.
            cur_height: The height of the current node.

        Returns:
            The maximum height between the left and right
subtrees.
        """
        if cur_node is None:
            return cur_height
        left_height = self._height(cur_node.left_child,
cur_height)
```

```
        right_height = self._height(cur_node.right_child,
cur_height)
        return max(left_height, right_height)

    def search(self, value):
        """
        Search for a value in the binary search tree.

        Args:
            value: The value to search for.

        Returns:
            True if the value is found, False otherwise.
        """
        if self.root is not None:
            return self._search(value, self.root)
        else:
            return False

    def _search(self, value, cur_node):
        """
        Helper method to recursively search for a value in the
binary search tree.

        Args:
            value: The value to search for.
            cur_node: The current node being traversed.

        Returns:
            True if the value is found, False otherwise.
        """
        if value == cur_node.value:
            return True
        elif value < cur_node.value and cur_node.left_child is
not None:
            return self._search(value, cur_node.left_child)
        elif value > cur_node.value and cur_node.right_child is
not None:
            return self._search(value, cur_node.right_child)

    def find_max(self):
```

```
        """
        Find the maximum value in the binary search tree.

        Returns:
            The maximum value in the tree or None if the tree is
empty.
        """
        if self.root is None:
            return None
        return self._find_max(self.root)

    def _find_max(self, cur_node):
        """
        Helper method to recursively find the maximum value in
the binary search tree.

        Args:
            cur_node: The current node being traversed.

        Returns:
            The maximum value in the tree.
        """
        if cur_node.right_child is None:
            return cur_node.value
        return self._find_max(cur_node.right_child)

    def find_min(self):
        """
        Find the minimum value in the binary search tree.

        Returns:
            The minimum value in the tree or None if the tree is
empty.
        """
        if self.root is None:
            return None
        return self._find_min(self.root)

    def _find_min(self, cur_node):
        """
```

```
        Helper method to recursively find the minimum value in
the binary search tree.

        Args:
            cur_node: The current node being traversed.

        Returns:
            The minimum value in the tree.
        """
        if cur_node.left_child is None:
            return cur_node.value
        return self._find_min(cur_node.left_child)


tree = BinarySearchTree()

tree.insert(5)
tree.insert(1)
tree.insert(3)
tree.insert(2)
tree.insert(7)
tree.insert(10)
tree.insert(0)
tree.insert(20)
tree.print_tree()
print()
print("tree height:", str(tree.height()))
print("search(10):", tree.search(10))
print("search(30):", tree.search(30))
```

Output: