Assignment no.: 7

Name : Kapare Swapnali Namdev

Class : SE

Div : A

Roll no : A-65

Problem statement :

Implement various operations on a Binary Search Tree, such as insertion, deletion, display, and search.

Input :

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        self.root = None
```

```
# Insert a node
def insert(self, key):
    self.root = self._insert(self.root, key)

def _insert(self, node, key):
    if node is None:
        return Node(key)
    if key < node.key:
        node.left = self._insert(node.left, key)
    else:
        node.right = self._insert(node.right, key)
    return node

# In-order Traversal (display)
def inorder(self):
    print("In-order Traversal:", end=" ")
    self._inorder(self.root)
    print()

def _inorder(self, node):
    if node:
        self._inorder(node.left)
        print(node.key, end=" ")
        self._inorder(node.right)

# Search for a node
def search(self, key):
    return self._search(self.root, key)

def _search(self, node, key):
    if node is None or node.key == key:
        return node
    if key < node.key:
        return self._search(node.left, key)
    else:
        return self._search(node.right, key)

# Delete a node
def delete(self, key):
    self.root = self._delete(self.root, key)

def _delete(self, node, key):
    if node is None:
        return node
    if key < node.key:
        node.left = self._delete(node.left, key)
    elif key > node.key:
        node.right = self._delete(node.right, key)
    else:
        # Node with only one child or no child
        if node.left is None:
            return node.right
        elif node.right is None:
            return node.left
        # Node with two children: get inorder successor
```

```
            temp = self._min_value_node(node.right)
            node.key = temp.key
            node.right = self._delete(node.right, temp.key)
        return node

    def _min_value_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current

# --- Testing the BST ---
if __name__ == "__main__":
    bst = BST()

    # Insert elements
    elements = [50, 30, 70, 20, 40, 60, 80]
    print("Inserting elements:", elements)
    for el in elements:
        bst.insert(el)

    # Display tree (in-order)
    bst.inorder()

    # Search for a key
    key_to_search = 60
    print(f"Searching for {key_to_search}...")
    result = bst.search(key_to_search)
    print("Found!" if result else "Not Found.")

    # Delete a node
    key_to_delete = 50
    print(f"Deleting node {key_to_delete}...")
    bst.delete(key_to_delete)

    # Display tree after deletion
    bst.inorder()
```

Output :

```
Inserting elements: [50, 30, 70, 20, 40, 60, 80]
In-order Traversal: 20 30 40 50 60 70 80
Searching for 60...
Found!
Deleting node 50...
In-order Traversal: 20 30 40 60 70 80
```