//Assignment 4

```
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 (handles duplicate)
    def insert(self, root, key):
        if root is None:
            return Node(key)
        if key < root.key:
            root.left = self.insert(root.left, key)
        elif key > root.key:
            root.right = self.insert(root.right, key)
        else:
            print(f"Duplicate entry {key} ignored.")
        return root

    # Search a node
    def search(self, root, key):
        if root is None or root.key == key:
            return root
        if key < root.key:
            return self.search(root.left, key)
        return self.search(root.right, key)

    # Find minimum value node (helper for delete)
    def minValueNode(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current
```

```
# Delete a node
def delete(self, root, key):
    if root is None:
        return root
    if key < root.key:
        root.left = self.delete(root.left, key)
    elif key > root.key:
        root.right = self.delete(root.right, key)
    else:
        # Node with only one child or no child
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        # Node with two children
        temp = self.minValueNode(root.right)
        root.key = temp.key
        root.right = self.delete(root.right, temp.key)
    return root

# Traversals
def inorder(self, root):
    if root:
        self.inorder(root.left)
        print(root.key, end=" ")
        self.inorder(root.right)

def preorder(self, root):
    if root:
        print(root.key, end=" ")
        self.preorder(root.left)
        self.preorder(root.right)

def postorder(self, root):
    if root:
        self.postorder(root.left)
        self.postorder(root.right)
        print(root.key, end=" ")
```

```
# Depth of tree
def depth(self, root):
    if root is None:
        return 0
    return 1 + max(self.depth(root.left), self.depth(root.right))

# Mirror image of tree
def mirror(self, root):
    if root:
        root.left, root.right = root.right, root.left
        self.mirror(root.left)
        self.mirror(root.right)

# Copy tree
def copy(self, root):
    if root is None:
        return None
    new_node = Node(root.key)
    new_node.left = self.copy(root.left)
    new_node.right = self.copy(root.right)
    return new_node

# Display parents with children
def display_parents(self, root):
    if root:
        if root.left or root.right:
            print(f"Parent {root.key}: ", end="")
            if root.left:
                print(f"Left Child = {root.left.key}", end=" ")
            if root.right:
                print(f"Right Child = {root.right.key}", end=" ")
            print()
        self.display_parents(root.left)
        self.display_parents(root.right)

# Display leaf nodes
def display_leaves(self, root):
```

```
        if root:
            if root.left is None and root.right is None:
                print(root.key, end=" ")
            self.display_leaves(root.left)
            self.display_leaves(root.right)

    # Display level-wise
    def level_order(self, root):
        if root is None:
            return
        queue = [root]
        while queue:
            current = queue.pop(0)
            print(current.key, end=" ")
            if current.left:
                queue.append(current.left)
            if current.right:
                queue.append(current.right)


# ----------------- Main Program -----------------
if __name__ == "__main__":
    tree = BST()
    root = None

    while True:
        print("\n--- Binary Search Tree Menu ---")
        print("1. Insert")
        print("2. Delete")
        print("3. Search")
        print("4. Display (Inorder, Preorder, Postorder)")
        print("5. Display Depth of Tree")
        print("6. Display Mirror Image")
        print("7. Create Copy of Tree")
        print("8. Display Parents with Children")
        print("9. Display Leaf Nodes")
        print("10. Display Level-wise")
        print("11. Exit")
```

```
choice = int(input("Enter choice: "))

if choice == 1:
    val = int(input("Enter value: "))
    root = tree.insert(root, val)

elif choice == 2:
    val = int(input("Enter value to delete: "))
    root = tree.delete(root, val)

elif choice == 3:
    val = int(input("Enter value to search: "))
    res = tree.search(root, val)
    if res:
        print(f"Found {val} in tree.")
    else:
        print(f"{val} not found.")

elif choice == 4:
    print("Inorder: ", end=""); tree.inorder(root); print()
    print("Preorder: ", end=""); tree.preorder(root); print()
    print("Postorder: ", end=""); tree.postorder(root); print()

elif choice == 5:
    print("Depth of tree:", tree.depth(root))

elif choice == 6:
    print("Mirror Image created.")
    tree.mirror(root)
    tree.inorder(root); print()

elif choice == 7:
    copy_root = tree.copy(root)
    print("Copied Tree (Inorder): ", end="")
    tree.inorder(copy_root); print()

elif choice == 8:
```

```
    print("Parents with their children:")
    tree.display_parents(root)

elif choice == 9:
    print("Leaf nodes: ", end="")
    tree.display_leaves(root); print()

elif choice == 10:
    print("Level order: ", end="")
    tree.level_order(root); print()

elif choice == 11:
    print("Exiting program...")
    break

else:
    print("Invalid choice! Try again.")
```

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 22

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 44

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree

8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 66

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 25

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 78

--- Binary Search Tree Menu ---

1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 1
Enter value: 95

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 3
Enter value to search: 95
Found 95 in tree.

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree

8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 4
Inorder: 22 25 44 66 78 95
Preorder: 22 44 25 66 78 95
Postorder: 25 95 78 66 44 22

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 5
Depth of tree: 5

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 6
Mirror Image created.

95 78 66 44 25 22

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 7
Copied Tree (Inorder): 95 78 66 44 25 22

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 8
Parents with their children:
Parent 22: Left Child = 44
Parent 44: Left Child = 66 Right Child = 25
Parent 66: Left Child = 78
Parent 78: Left Child = 95

--- Binary Search Tree Menu ---
1. Insert

2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 9
Leaf nodes: 95 25

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 10
Level order: 22 44 66 25 78 95

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes

10. Display Level-wise
11. Exit
Enter choice: 2
Enter value to delete: 99

--- Binary Search Tree Menu ---
1. Insert
2. Delete
3. Search
4. Display (Inorder, Preorder, Postorder)
5. Display Depth of Tree
6. Display Mirror Image
7. Create Copy of Tree
8. Display Parents with Children
9. Display Leaf Nodes
10. Display Level-wise
11. Exit
Enter choice: 11