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

class BinarySearchTree:
    def __init__(self):
        self.root = None

    # a) Insert (Handle duplicate entries)
    def insert(self, key):
        def _insert(root, key):
            if root is None:
                return Node(key)
            if key < root.key:
                root.left = _insert(root.left, key)
            elif key > root.key:
                root.right = _insert(root.right, key)
            else:
                print(f"Duplicate entry '{key}' ignored.")
            return root
        self.root = _insert(self.root, key)

    # b) Delete
    def delete(self, key):
        def _delete(root, key):
            if root is None:
```

```
            return root
        if key < root.key:
            root.left = _delete(root.left, key)
        elif key > root.key:
            root.right = _delete(root.right, key)
        else:
            if root.left is None:
                return root.right
            elif root.right is None:
                return root.left
            temp = self._min_value_node(root.right)
            root.key = temp.key
            root.right = _delete(root.right, temp.key)
        return root
    self.root = _delete(self.root, key)

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

# c) Search
def search(self, key):
    def _search(root, key):
        if root is None or root.key == key:
            return root
        if key < root.key:
```

```
            return _search(root.left, key)
        return _search(root.right, key)
    return _search(self.root, key) is not None

# d) Display tree (Traversal)
def inorder(self):
    def _inorder(root):
        if root:
            _inorder(root.left)
            print(root.key, end=" ")
            _inorder(root.right)
    _inorder(self.root)
    print()

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

# f) Display - Mirror image
def mirror(self):
    def _mirror(root):
        if root:
            root.left, root.right = root.right, root.left
            _mirror(root.left)
```

```
            _mirror(root.right)
        _mirror(self.root)

    # g) Create a copy
    def copy(self):
        def _copy(root):
            if root is None:
                return None
            new_node = Node(root.key)
            new_node.left = _copy(root.left)
            new_node.right = _copy(root.right)
            return new_node
        new_tree = BinarySearchTree()
        new_tree.root = _copy(self.root)
        return new_tree

    # h) Display all parent nodes with their child nodes
    def display_parents(self):
        def _display_parents(root):
            if root:
                if root.left or root.right:
                    print(
                        f"Parent: {root.key}, Left Child: {root.left.key if root.left else None}, Right Child:
{root.right.key if root.right else None}")
                _display_parents(root.left)
                _display_parents(root.right)
        _display_parents(self.root)
```

```
    # i) Display leaf nodes
    def display_leaves(self):
        def _display_leaves(root):
            if root:
                if root.left is None and root.right is None:
                    print(root.key, end=" ")
                _display_leaves(root.left)
                _display_leaves(root.right)
        _display_leaves(self.root)
        print()

    # j) Display tree level-wise
    def level_order(self):
        if self.root is None:
            return
        queue = [self.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)
        print()

# --------------------------------
# Example usage
# --------------------------------
```

```
bst = BinarySearchTree()
bst.insert(50)
bst.insert(30)
bst.insert(70)
bst.insert(20)
bst.insert(40)
bst.insert(60)
bst.insert(80)

print("Inorder Traversal:")
bst.inorder()

print("Depth of Tree:", bst.depth())

print("\nParent Nodes:")
bst.display_parents()

print("\nLeaf Nodes:")
bst.display_leaves()

print("\nLevel Order Traversal:")
bst.level_order()

print("\nMirror Image:")
bst.mirror()
bst.inorder()
```

```
print("\nCopy of BST:")
bst_copy = bst.copy()
bst_copy.inorder()

print("\nDeleting 70:")
bst.delete(70)
bst.inorder()
```