CODE:

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

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

    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

    def search(self, root, key):
        if root is None:
            return None
        if root.key == key:
            return root
        if key < root.key:
            return self.search(root.left, key)
        return self.search(root.right, key)

    def delete(self, root, key):
        if root is None:
            print(f"Value {key} not found in tree!")
            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:
```

```
            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 = self.delete(root.right, temp.key)
        return root

    def min_value_node(self, node):
        current = node
        while current.left:
            current = current.left
        return current

    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=" ")

    def depth(self, root):
        if root is None:
            return 0
        return 1 + max(self.depth(root.left), self.depth(root.right))

    def mirror(self, root):
        if root:
            root.left, root.right = root.right, root.left
```

```
        self.mirror(root.left)
        self.mirror(root.right)
    return root

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

def show_parents(self, root):
    if root:
        if root.left:
            print(f"Parent {root.key} → Left Child {root.left.key}")
        if root.right:
            print(f"Parent {root.key} → Right Child {root.right.key}")
        self.show_parents(root.left)
        self.show_parents(root.right)

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

def display_levelwise(self, root):
    if not root:
        print("(empty tree)")
        return
    q = [(root, 0)]
    levels = {}
    while q:
        node, l = q.pop(0)
        levels.setdefault(l, []).append(str(node.key) if node else " ")
        if node:
            q.append((node.left, l + 1))
            q.append((node.right, l + 1))
    max_level = max(levels)
```

```
        for l in range(max_level + 1):
            spacing = " " * (2 ** (max_level - l))
            print(spacing.join(levels[l]).center(80))


def menu():
    bst = BST()
    while True:
        print("\n------- MENU -------")
        print("1. Insert")
        print("2. Search")
        print("3. Delete")
        print("4. Inorder Traversal")
        print("5. Preorder Traversal")
        print("6. Postorder Traversal")
        print("7. Depth of Tree")
        print("8. Mirror Tree")
        print("9. Copy Tree")
        print("10. Show Parent → Child nodes")
        print("11. Show Leaf Nodes")
        print("12. Display Level-wise")
        print("13. Exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            key = int(input("Enter value to insert: "))
            bst.root = bst.insert(bst.root, key)
        elif choice == "2":
            key = int(input("Enter value to search: "))
            found = bst.search(bst.root, key)
            print("Found!" if found else "Not found.")
        elif choice == "3":
            key = int(input("Enter value to delete: "))
            bst.root = bst.delete(bst.root, key)
        elif choice == "4":
            print("Inorder Traversal: ", end="")
            bst.inorder(bst.root)
            print()
        elif choice == "5":
            print("Preorder Traversal: ", end="")
            bst.preorder(bst.root)
```

```
            print()
        elif choice == "6":
            print("Postorder Traversal: ", end="")
            bst.postorder(bst.root)
            print()
        elif choice == "7":
            print("Depth of Tree:", bst.depth(bst.root))
        elif choice == "8":
            bst.root = bst.mirror(bst.root)
            print("Tree mirrored.")
        elif choice == "9":
            copy_root = bst.copy(bst.root)
            print("Inorder Traversal of Tree Copy: ", end="")
            bst.inorder(copy_root)
            print()
        elif choice == "10":
            print("Parent → Child nodes:")
            bst.show_parents(bst.root)
        elif choice == "11":
            print("Leaf Nodes: ", end="")
            bst.show_leaves(bst.root)
            print()
        elif choice == "12":
            print("Tree Level-wise:")
            bst.display_levelwise(bst.root)
        elif choice == "13":
            print("Exiting program.")
            break
        else:
            print("Invalid choice.")

if __name__ == "__main__":
    menu()
```

```
------- MENU -------
1. Insert
2. Search
3. Delete
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Depth of Tree
8. Mirror Tree
9. Copy Tree
10. Show Parent → Child nodes
11. Show Leaf Nodes
12. Display Level-wise
13. Exit
Enter your choice: 1
Enter value to insert: 85
```

```
------- MENU -------
1. Insert
2. Search
3. Delete
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Depth of Tree
8. Mirror Tree
9. Copy Tree
10. Show Parent → Child nodes
11. Show Leaf Nodes
12. Display Level-wise
13. Exit
Enter your choice: 12
Tree Level-wise:
                                        85

                                    12

                                        45
```