```
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("Duplicate entry ignored!")
        return root

    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)

    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:
            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):
        while node.left:
            node = node.left
        return node

        # 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=" ")

    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)
    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)

    def copy_tree(self, root):
        if root is None:
            return None
        new_node = Node(root.key)
        new_node.left = self.copy_tree(root.left)
        new_node.right = self.copy_tree(root.right)
        return new_node

    def parent_child(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} -> Left Child {root.right.key} ")
                self.parent_child(root.left)
                self.parent_child(root.right)

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



bst = BST()

while True:
```

```
print("""
\n --- Binary Search Tree Menu ---
1.Insert
2.Delete
3.Search
4.Inorder Traversal
5.Preorder Traversal
6.Postorder Traversal
7.Level Order Traversal
8.Depth of Tree
9.Mirror Image
10.Copy Tree
11.Display Parent & Child
12.Display Leaf Nodes
0.Exit
""")
choice = input("Enter your Choice: ")

if choice == '1':
    val = int(input("Enter value to insert: "))
    bst.root = bst.insert(bst.root, val)
elif choice == '2':
    val = int(input("Enter value to delete: "))
    bst.root = bst.delete(bst.root, val)
elif choice == '3':
    val = int(input("Enter value to search: "))
    found = bst.search(bst.root, val)
    print("Found!" if found else "Not Found!")
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("Level Order Traversal: ", end=" ")
        bst.level_order(bst.root)
        print()
    elif choice == '8':
        print("Depth of tree: ", bst.depth(bst.root))
    elif choice == '9':
        bst.mirror(bst.root)
        print("Mirror Image Created!")
    elif choice == '10':
        copied_root = bst.copy_tree(bst.root)
        print("Inorder of Copied Tree: ", end=" ")
        bst.inorder(copied_root)
        print()
    elif choice == '11':
        bst.parent_child(bst.root)
    elif choice == '12':
        print("Leaf Nodes: ", end=" 1")
        bst.leaf_nodes(bst.root)
        print()
    elif choice == '0':
        print("Exiting....:)")
        break
    else:
        print("Invalid choice! Try Again.")
```