```python
# ======================================
# CLASS NODE
# ======================================
class Node:
    def __init__(self, data=None):
        self.data = data
        self.right = self.left = None


# ======================================
# IN ORDER Traversal
# ======================================
def inorder(root):
    if root:
        inorder(root.left)
        print(root.data, end=" ")
        inorder(root.right)


# ======================================
# BFS Traversal
# ======================================
def bfs_traversal(root):
    if not root:
        return
    queue = [root]
    while queue:
        node = queue.pop(0)
        print(node.data, end=" ")
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)


# ======================================
# Initialize the tree
# ======================================
root = Node(5)
child1 = Node(3)
child2 = Node(7)
root.left = child1
root.right = child2

child1.left = Node(-2)
```

```python
child1.right = Node(4)
child2.left = Node(6)
child2.right = Node(10)

# ======================================
# Terminal Separator printer
# ======================================
separator = lambda: print("\n", "=" * 45)

separator()

print("In order traversal: ")
inorder(root)


# ======================================
# Recursive INSERT Function
# ======================================
def insert(root, key):
    if not root:
        return Node(key)

    if key < root.data:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)

    return root


# ---------------------------------------------------

insert(root, 11)

separator()
print("In order traversal: ")
inorder(root)

separator()
print("BFS traversal: ")
bfs_traversal(root)

insert(root, 8)

separator()
```

```python
print("BFS traversal: ")
bfs_traversal(root)

insert(root, 9)

separator()
print("BFS traversal: ")
bfs_traversal(root)


# ========================================
# BST Constructor Function
# ========================================
def constructBST(keys):
    root = None
    for key in keys:
        root = insert(root, key)

    return root


# -----------------------------------------------------

separator()
# Construct new Tree
tree_2 = [15, 10, 20, 8, 12, 16, 25]

root_2 = constructBST(tree_2)
print("\nIn order Root 2")
inorder(root_2)
separator()
print("BFS Traversal Root 2")
bfs_traversal(root_2)


# ==============================================
# Recursive function to search in a given BST
# ==============================================
def search(root, key, parent):
    # if the key is not present in the key
    if root is None:
        print("Key not found")
        return

    # if the key is found
```

```
    if root.data == key:
        if parent is None:
            print(f"The node with key {key} is root 
node")
        elif key < parent.data:
            print(
                f"The given key [{key}] is the left node 
of the node with key",
                parent.data,
            )
        else:
            print(
                f"The given key [{key}] is the right node 
of the node with key",
                parent.data,
            )

        return

    # if the given key is less than the root node, recur 
for the left subtree;
    # otherwise, recur for the right subtree

    if key < root.data:
        search(root.left, key, root)
    else:
        search(root.right, key, root)


separator()
print("SEARCH")
search(root_2, 16, None)

separator()


# ======================================
# CREATE ADJACENCY LIST Function
# ======================================
def create_adj_list(root, adjacency_list):
    """
    Creates Adjacency List to represent each Node and its 
children
    """
    if root:
```

```
        adjacency_list[root.data] = []
        if root.left:
            
adjacency_list[root.data].append(root.left.data)
            create_adj_list(root.left, adjacency_list)
        if root.right:
            
adjacency_list[root.data].append(root.right.data)
            create_adj_list(root.right, adjacency_list)


adjacency_list = {}
res = create_adj_list(root, adjacency_list)
print("This is the adjacency_list de root:")
for node, neighbor in adjacency_list.items():
    print(node, neighbor)


separator()
adjacency_list = {}
res = create_adj_list(root_2, adjacency_list)
print("This is the adjacency_list of root_2:")
for node, neighbor in adjacency_list.items():
    print(node, neighbor)
```