Name : Mirza Meherab Hosen Rudra
ID : 21101048

Problem -01

```
class Node:
    def __init__(self, data, parent_node, left, right):
        self.data = data
        self.parent_node = parent_node
        self.left = left
        self.right = right


class binarytree:
    def __init__(self):
        self.root = Node(1, None, None, None)
        self.second_node = Node(2, self.root, None, None)
        self.thrd_node = Node(3, self.root, None, None)
        self.fourth_node = Node(4, self.thrd_node, None, None)
        self.fifth_node = Node(5, self.thrd_node, None, None)
        self.sixt_node = Node(6, self.fifth_node, None, None)
        self.seven_node = Node(7, self.fifth_node, None, None)
        self.root.left = self.second_node
        self.root.right = self.thrd_node
        self.thrd_node.left = self.fourth_node
        self.thrd_node.right = self.fifth_node
        self.fifth_node.left = self.sixt_node
        self.fifth_node.right = self.seven_node

    def maximum(self, first, second):
        if second <= first:
            return first
        else:
            return second

    def height(self, root):
        if root is None:
            return 0

        return 1 + self.maximum(self.height(root.left), self.height(root.right))


tree = binarytree()
print("No. 01 >>>")
print("Height : ", tree.height(tree.root))
print("\n")
```

Problem -03

```
class Node:
    def __init__(self, data):
```

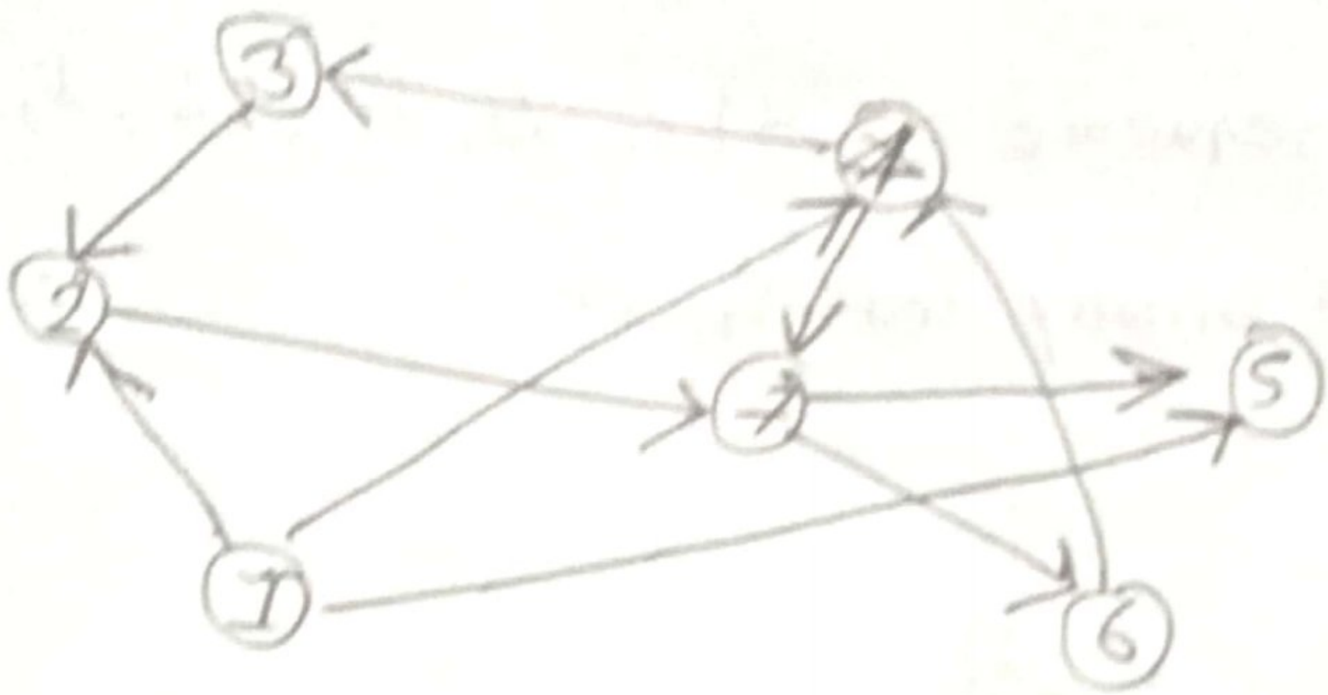
```
        self.data = data
        self.parent = None
        self.left = None
        self.right = None


def get_level(node, data, l):
    if node is None:
        return 0
    if data == node.data:
        return l

    leveld = get_level(node.left, data, l + 1)
    if leveld != 0:
        return leveld

    leveld = get_level(node.right, data, l + 1)
    return leveld


def getlevel(node, data):
    return get_level(node, data, 1)


root = Node(1)
second_node = Node(2)
thrd_node = Node(3)
fourth_node = Node(4)
fifth_node = Node(5)
sixt_node = Node(6)
seven_node = Node(7)
eight_node = Node(8)
root.left = second_node
root.right = thrd_node
thrd_node.left = fourth_node
thrd_node.right = fifth_node
fifth_node.left = sixt_node
fifth_node.right = seven_node
seven_node.left = eight_node

print("No. 02 >>>")
print(f"LEVEL: {getlevel(root,8)}")
print("\n")

# Problem -03


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

```
def preorder(node):
    if node is None:
        return
    else:
        print(node.data, end="  ")
        preorder(node.left)
        preorder(node.right)


root = Node(1)
second_node = Node(2)
thrd_node = Node(3)
fourth_node = Node(4)
fifth_node = Node(5)
sixt_node = Node(6)
seven_node = Node(7)
eight_node = Node(8)
root.left = second_node
root.right = thrd_node
thrd_node.left = fourth_node
thrd_node.right = fifth_node
fifth_node.left = sixt_node
fifth_node.right = seven_node
seven_node.left = eight_node


print("No. 03 >>>")
preorder(root)
print("\n")

# Problem -04


class Node:
    def __init__(self, data):
        self.data = data
        self.parent = None
        self.left = None
        self.right = None


def inorder(node):
    if node is None:
        return
    else:
        inorder(node.left)
        print(node.data, end="  ")
        inorder(node.right)


root = Node(1)
second_node = Node(2)
thrd_node = Node(3)
fourth_node = Node(4)
fifth_node = Node(5)
```

```
sixt_node = Node(6)
seven_node = Node(7)
eight_node = Node(8)
root.left = second_node
root.right = thrd_node
thrd_node.left = fourth_node
thrd_node.right = fifth_node
fifth_node.left = sixt_node
fifth_node.right = seven_node
seven_node.left = eight_node


print("No. 04 >>>")
inorder(root)
print("\n")

# Problem -05
```

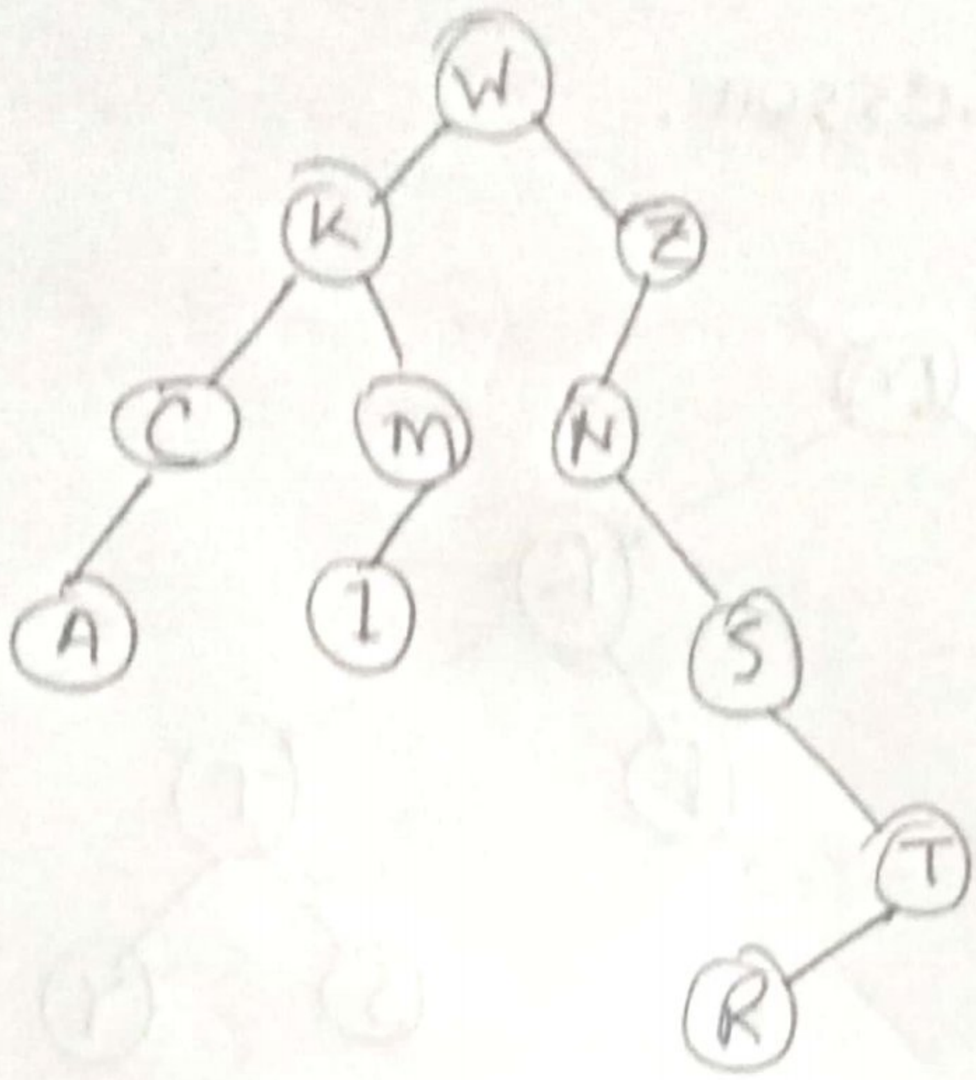
```
class Node:
    def __init__(self, data):
        self.data = data
        self.parent = None
        self.left = None
        self.right = None


def postorder(node):
    if node is None:
        return
    else:
        postorder(node.left)
        postorder(node.right)
        print(node.data, end="  ")


root = Node(1)
second_node = Node(2)
thrd_node = Node(3)
fourth_node = Node(4)
fifth_node = Node(5)
sixt_node = Node(6)
seven_node = Node(7)
eight_node = Node(8)
root.left = second_node
root.right = thrd_node
thrd_node.left = fourth_node
thrd_node.right = fifth_node
fifth_node.left = sixt_node
fifth_node.right = seven_node
seven_node.left = eight_node

print("No. 05 >>>")
postorder(root)
print("\n")
```

Ans 6



The equivalent graph for the matrix in the question has been drawn above.

7



sequence for Pre-order, in-order & postorder are as ~~foo~~ follows;

pre-order → WKCAMIZNSTR

In-order → ACKIMWNSRTZ

post-order → ACIMKRTSNZW

Scanned with CamScanner

7(a)

Given sequence (3, 51, 6, 65, 17, 12, 1, 22, -3, 15).

Resultant binary search is :

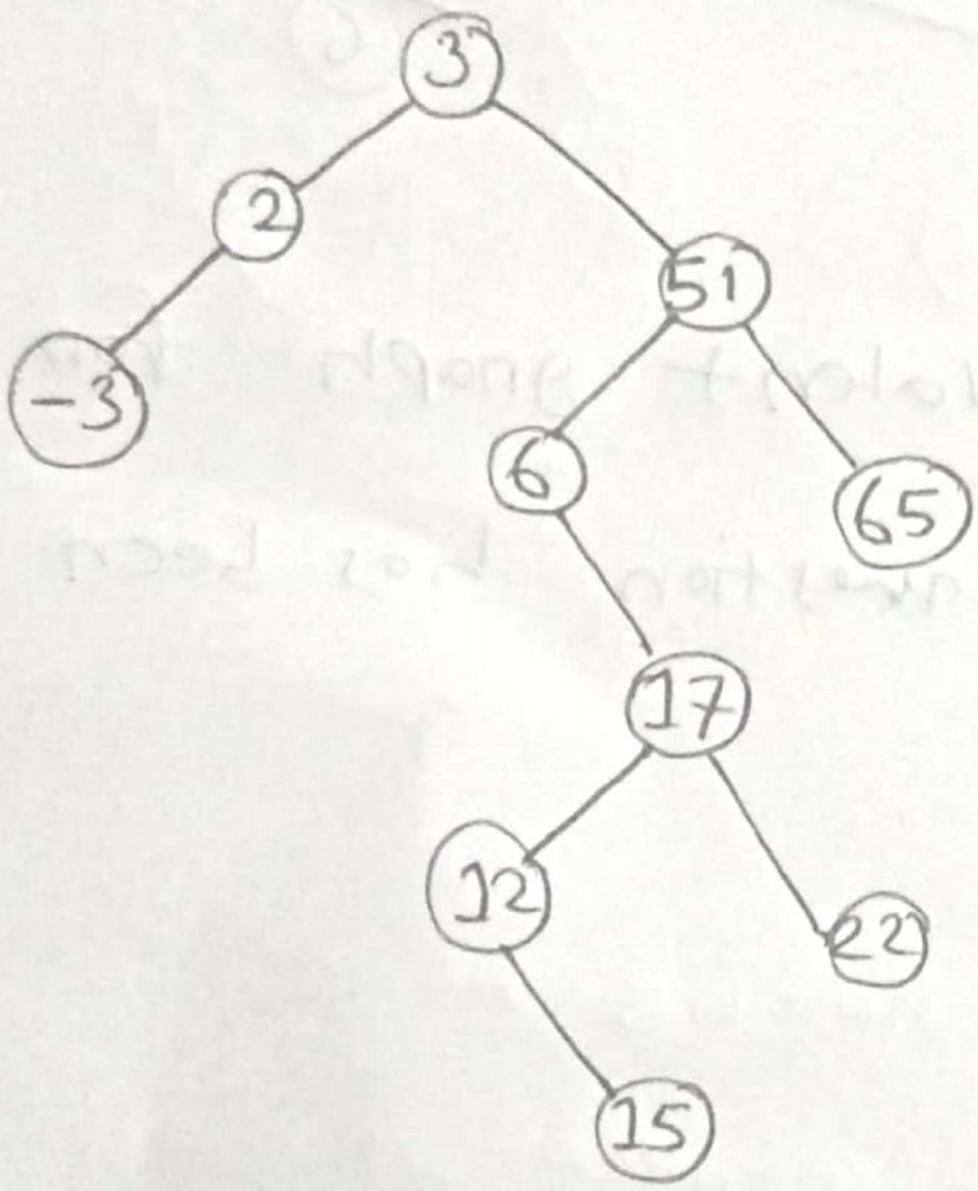
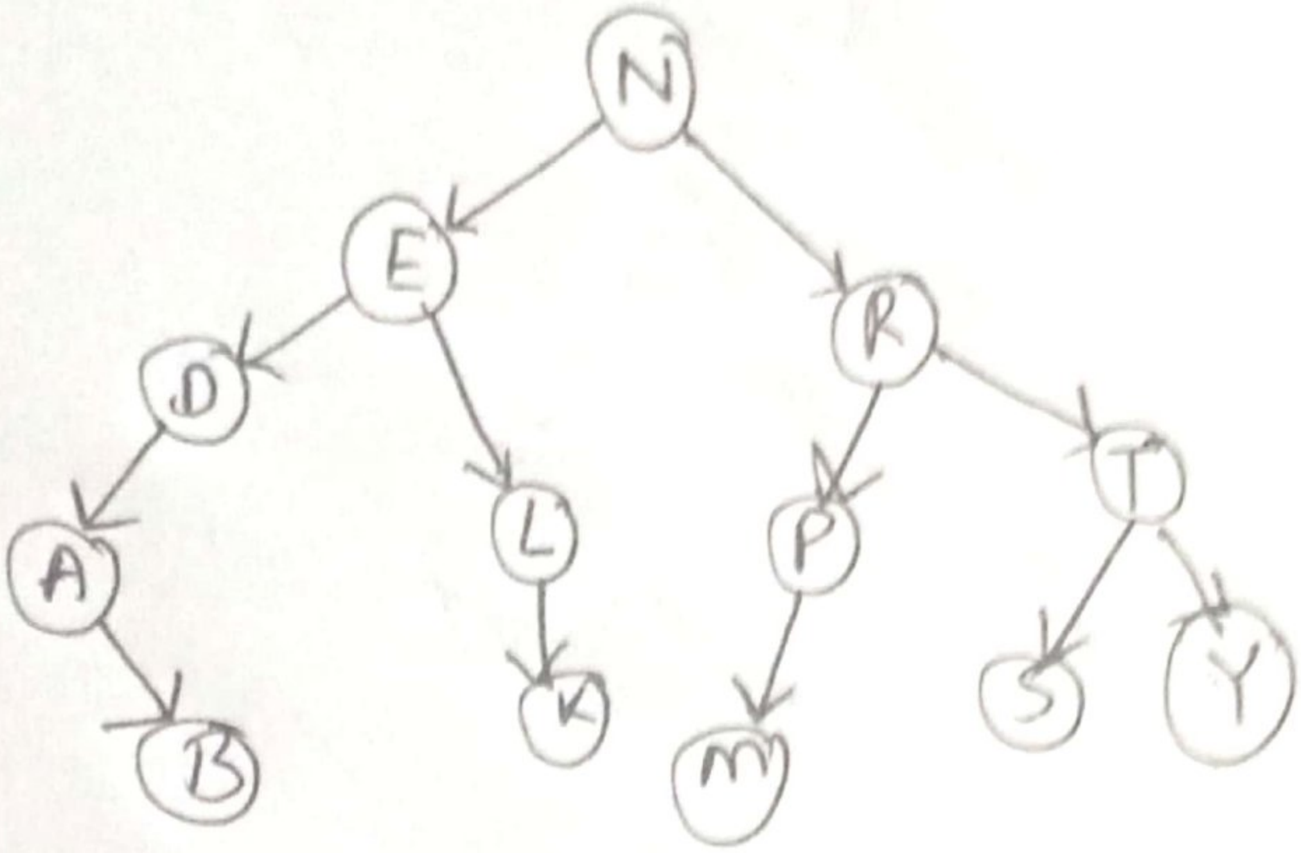


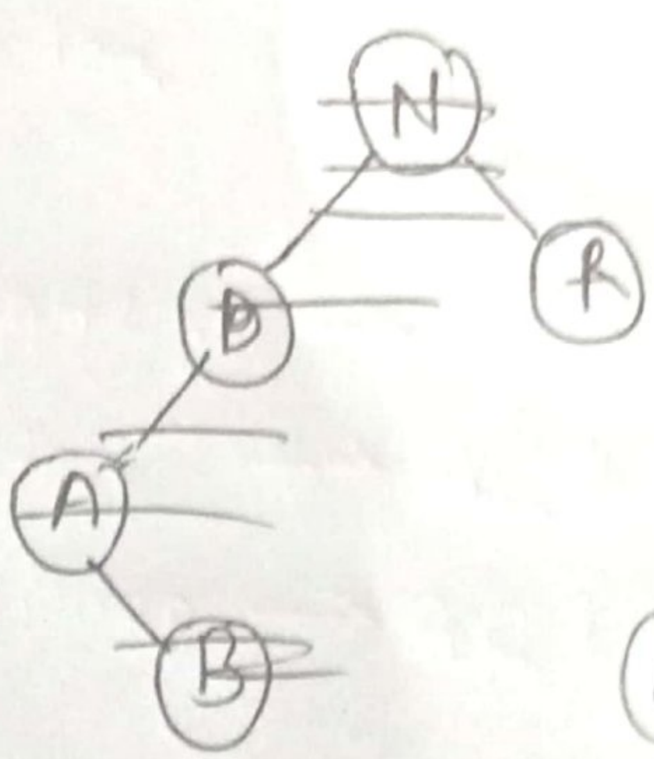
Fig: Bst

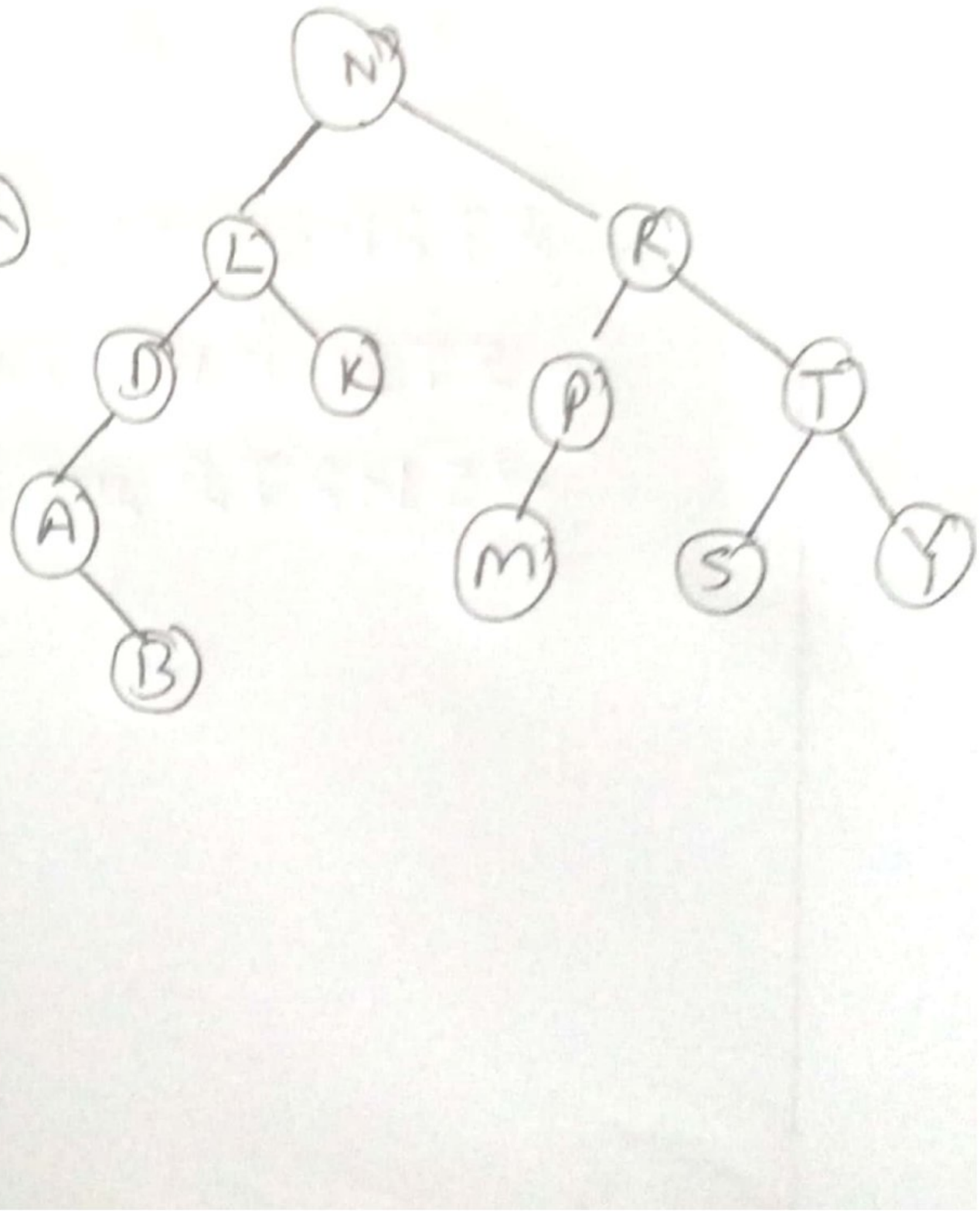
(Ans)

7(b)



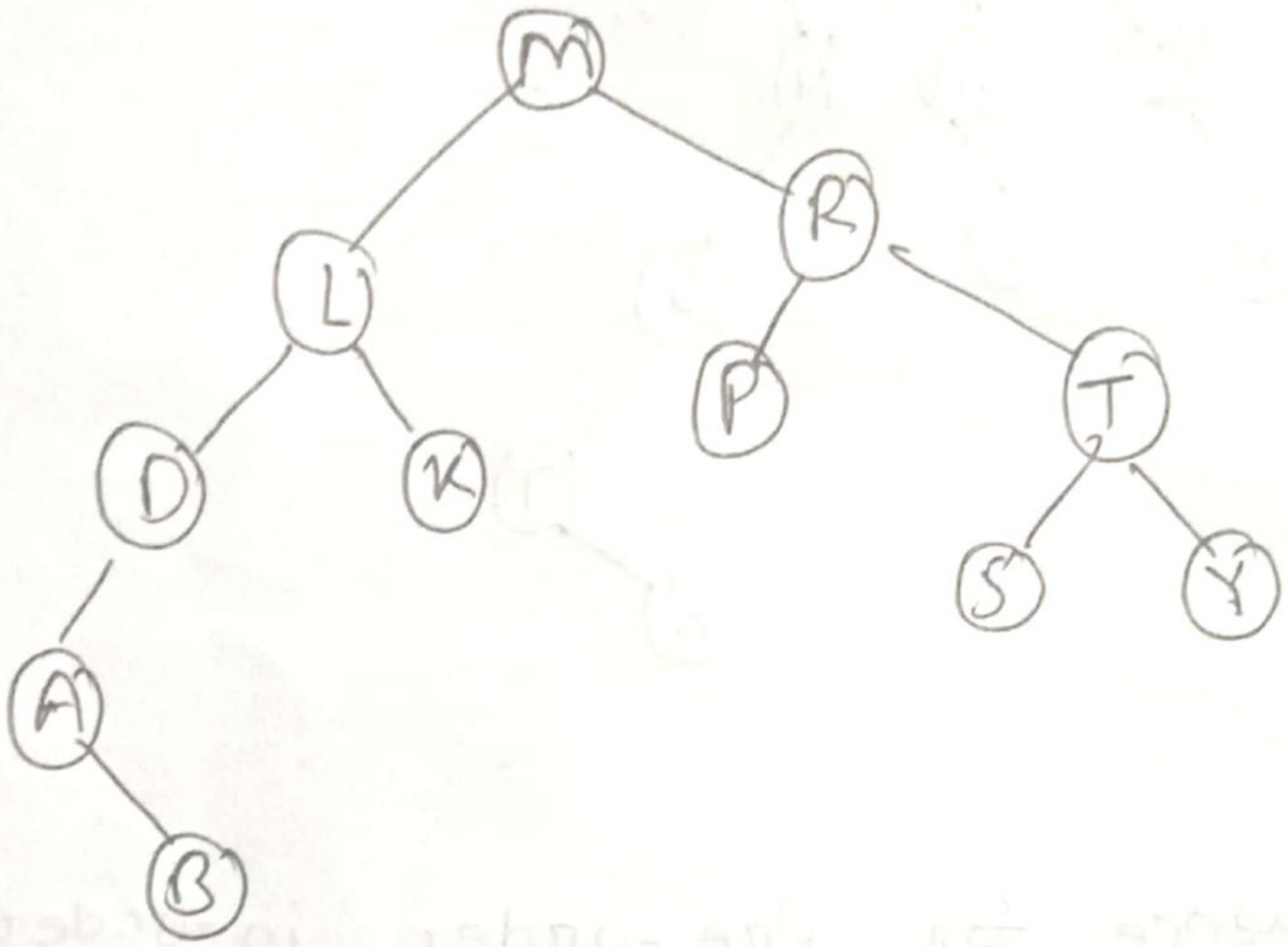
(a) Step -1 : Remove E with help of its Succesor.



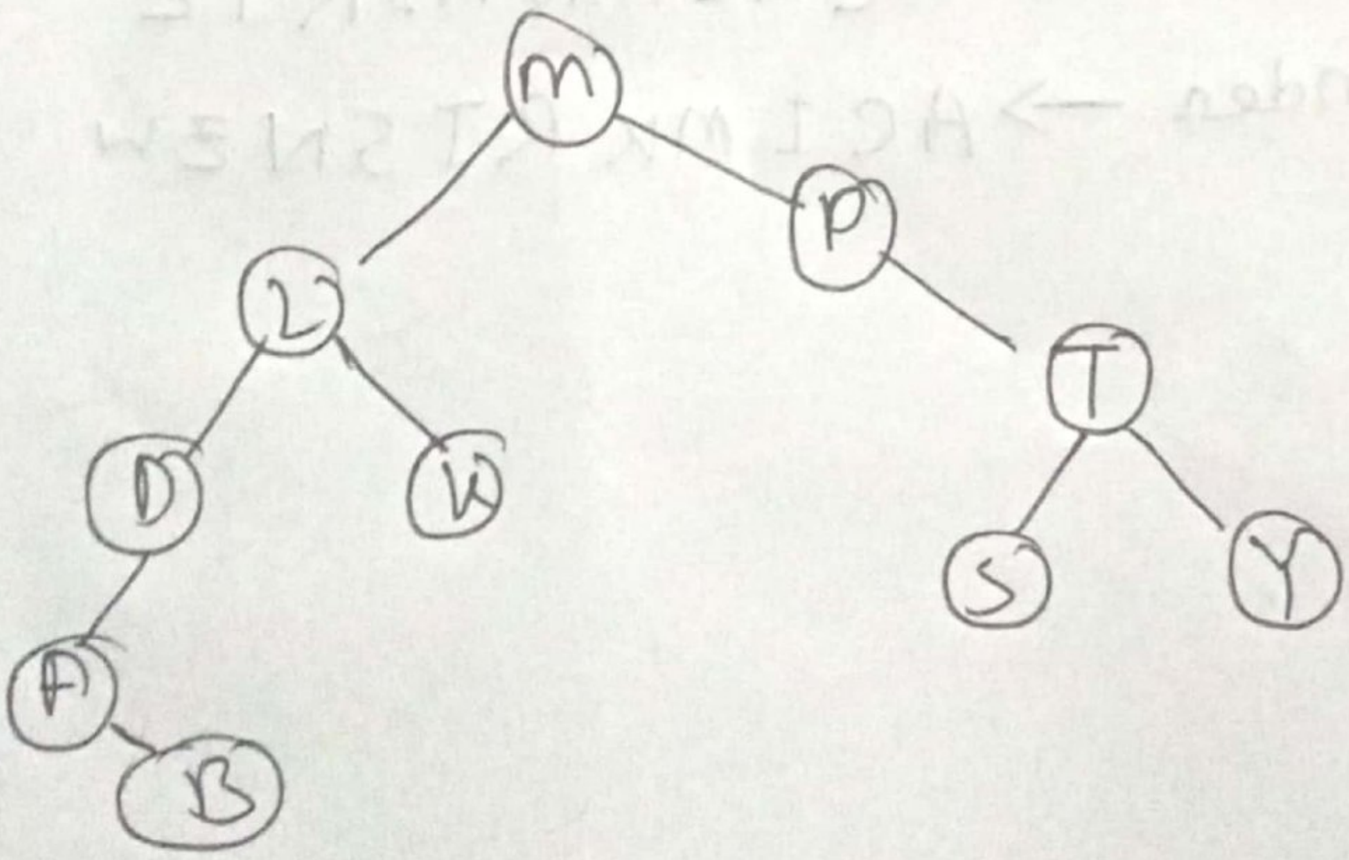


Scanned with CamScanner

(b) STEP-2 : Remove Node n with the help of its succesSon.



(c) STEP-3 : Remove node R with the help of its predecesSon



Scanned with CamScanner