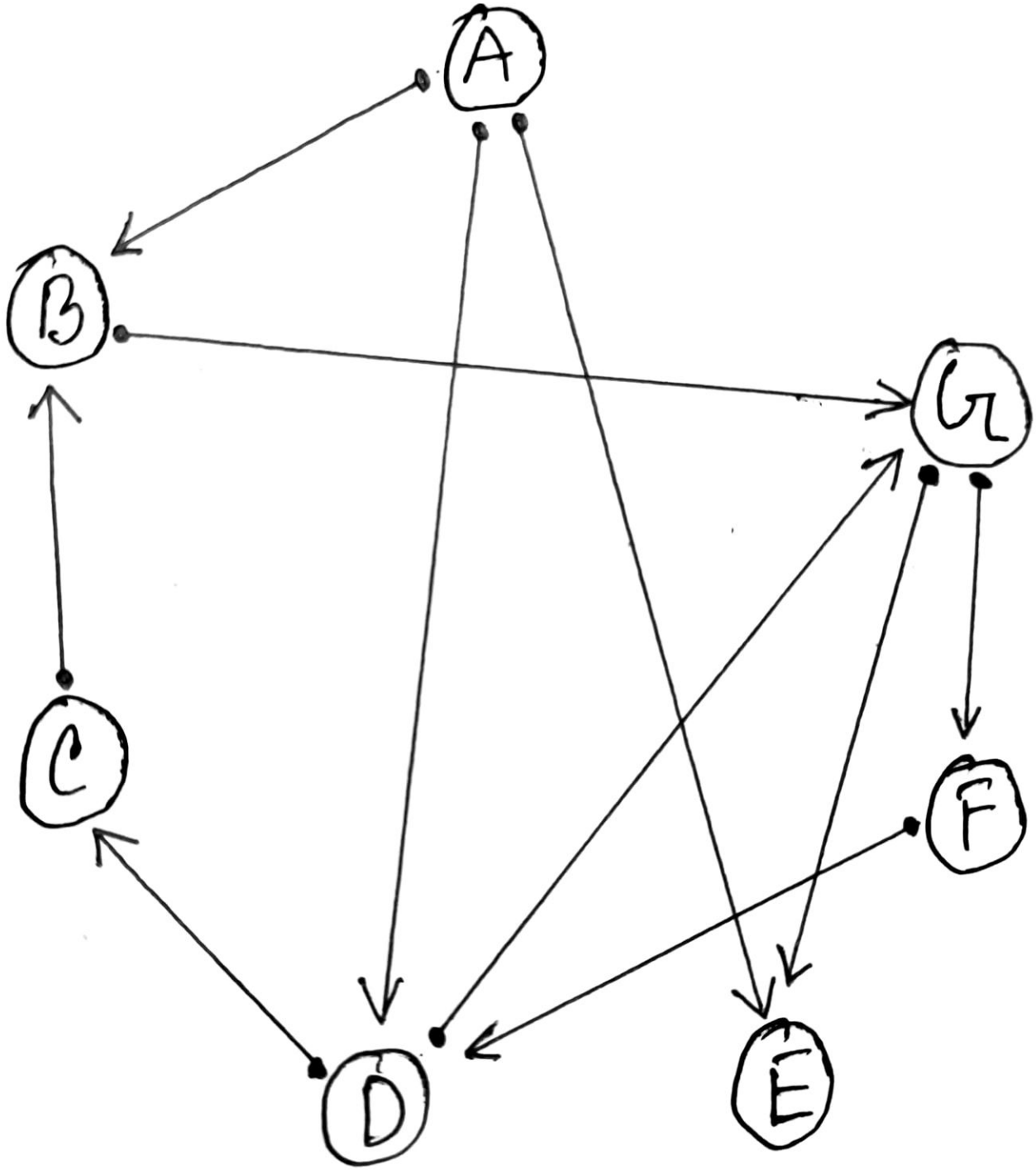
```
"""
ID: 20301038
Section: 11
MD ASIFUL ALAM

"""

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


class binaryTreeclass:
    def __init__(self, x):
        self.tree = self.create_tree(x)
        self.arr = x

    def create_tree(self, x, i=1):
        if i < 0 or i >= len(x) or i is None:
            return
        else:
            root = Node(x[i])
            root.left = self.create_tree(x, 2 * i)
            root.right = self.create_tree(x, 2 * i + 1)
            if root.left is not None:
                root.left.parent = root
            return root

def maxhight(self,root):
    if root is None:
        return 0
    else:
        left = 1 + self.maxhight(root.left)
        right = 1 + self.maxhight(root.right)
        if left > right:
            return left
        else:
             return right

def level(self,root):
    if root.parent is None:
        return 0
    else:
        return 1 + self.level(root.parent)

    def preOrderTraversal(self,root):
        if root is not None:
            print(root.elem, end=" ")
            self.preOrderTraversal(root.left)
            self.preOrderTraversal(root.right)

    def InOrderTraversal(self,root):
        if root != None:
            self.InOrderTraversal(root.left)
            print(root.elem, end=" ")
            self.InOrderTraversal(root.right)

    def Post_order_Traversal(self,root):
        if root != None:
            self.Post_order_Traversal(root.left)
            self.Post_order_Traversal(root.right)
            print(root.elem, end=" ")
#
    def check_Tree_Same(self,root, root2):
```

```
        if root == None or root2 == None:
            return root == root2
        if root.elem != root2.elem:
            return False
        if not self.check_Tree_Same(root.left, root2.left):
            return False
        if not self.check_Tree_Same(root.right, root2.right):
            return False
        return True

    def copy_tree_create(self,root, newRoot=None):
        if root is None:
            return
        if newRoot is None:
            newRoot = Node(root.elem)
        if root.left is not None:
            newRoot.left = Node(root.left.elem)
        if root.right is not None:
            newRoot.right = Node(root.right.elem)
        self.copy_tree_create(root.left, newRoot.left)
        self.copy_tree_create(root.right, newRoot.right)
        print(newRoot.elem, end=" ")

x = [None, 10, 5, 7, 17, 9, None, 25]
y = binaryTreeclass(x)

print("Maximum height of a tree is:", y.maxhight(y.tree))

print("Level :", y.level(y.tree.left.left))
y.preOrderTraversal(y.tree)
print()

y.InOrderTraversal(y.tree)
print()

y.Post_order_Traversal(y.tree)
#
z = [None, 10, 5, 7, 17, 9, None, 25]
l2 = binaryTreeclass(z)
# root2 = l2.tree
print()

print(y.check_Tree_Same(y.tree, l2.tree))

y.copy_tree_create(y.tree)
```


A
B
G
C
F
D
E