```python
'''
CSE220 FALL-21
Lab Assignment: 08
Md. Bokhtiar Rahman Juboraz
ID: 20301138
Section: 09

'''




#class treeNode for solution no (1-7)
class treeNode:
   def __init__(self, data=None, left_child=None, right_child=None):
       self.data = data
       self.left_child = left_child
       self.right_child = right_child
# Answer no. 1:
def tree_height(root):
   if root is not None:
       highest_value = max(tree_height(root.left_child), tree_height(root.right_child))
       height = 1 + highest_value
       return height
   else:
       return 0
tree_root = treeNode(20)

tree_root.right_child = treeNode(25)
tree_root.left_child = treeNode(23)

tree_root.left_child.right_child = treeNode(25)
tree_root.left_child.left_child = treeNode(10)

tree_root.right_child.left_child = treeNode(35)
tree_root.right_child.right_child = treeNode(30)

print("The height : ", tree_height(tree_root))


# Answer no. 2:
def updateLevels(node, data, level):
   if node is not None:
       if node.data == data:
           return level

       tree_ground = updateLevels(node.left_child,data, level + 1)

       if tree_ground == 0:
           tree_ground = updateLevels(node.right_child,data, level + 1)
       else:
           return tree_ground

   else:
       return 0
```

```python
def treeLevels(node, data):
   level = updateLevels(node, data, 1)
   return level

tree_root = treeNode(5)
tree_root.left_child = treeNode(1)
tree_root.right_child = treeNode(2)
tree_root.left_child.left_child = treeNode(3)
tree_root.left_child.right_child = treeNode(4)


for i in range(1, 8):
   checkLevels = treeLevels(tree_root, i)

   if checkLevels is not None:
       print("Level of", i,"is", treeLevels(tree_root, i),'\n')
   else:
       print(i, "is not present in the tree, so the level is 0", '\n')


#Inserting roots for traversal operations:
def get_treeRoot(root,newValue):
   if root is not None:
       if newValue<root.data:
           root.left_child = get_treeRoot(root.left_child,newValue)
       else:
           root.right_child = get_treeRoot(root.right_child,newValue)
       return root
   else:
       root=treeNode(newValue)
       return root

tree_root =get_treeRoot(None,35)
get_treeRoot(tree_root,30)
get_treeRoot(tree_root,5)
get_treeRoot(tree_root,6)
get_treeRoot(tree_root,7)
get_treeRoot(tree_root,1)
get_treeRoot(tree_root,70)
get_treeRoot(tree_root,50)
 #answer no. 3:
def pre_order(root):
   if root is not None:
       print(root.data)
       pre_order(root.left_child)
       pre_order(root.right_child)
   else:
       return

print("Pre-order traversal of binary tree is:")
pre_order(tree_root)
```

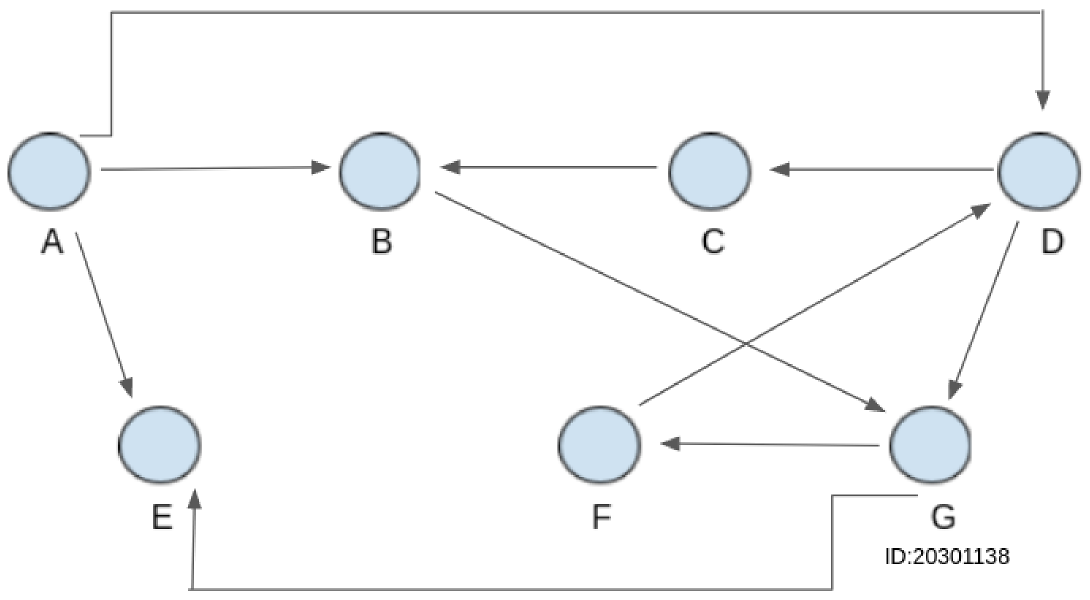
```
       #answer no. 4:
       def in_order(root):
          if root is not None:
              in_order(root.left_child)
              print(root.data)
              in_order(root.right_child)
          else:
              return

       print("In-order traversal of binary tree is:")
       in_order(tree_root)


       #answer no. 5:
       def post_order(root):
          if root is not None:
              post_order(root.left_child)
              post_order(root.right_child)
              print(root.data)
          else:
              return

       print("Post-order traversal of binary tree is:")
       post_order(tree_root)


       #answer no. 6:
       def checkSimilar(ele_1, ele_2):
          if ele_1 == None:
              if ele_2 == None:
                  return True
          elif ele_1 == None or ele_2 == None:
              return False

          if checkSimilar(ele_1, ele_2):
              print("Two trees are exactly same")
          else:
              print("Two trees are not same")

          return ele_1.data == ele_2.data and checkSimilar(ele_1.left_child, ele_2.left_child) and
checkSimilar(ele_1.right_child, ele_2.right_child)

       #new  class for another binary tree
       class newTreeNode2:
          def __init__(self, data):
              self.data = data
              self.left_child = None
              self.right_child = None

       #answer no. 7:
       def makeCopy(self):
          mirror_root = newTreeNode2(self.data)
          if self.right_child is not None:
              mirror_root.left_child = self.right_child.makeCopy()
```

```
    elif self.left_child is not None:
        mirror_root.right_child = self.left_child.makeCopy()
    else:
        mirror_root.right_child = None
        mirror_root.left_child = None
    return mirror_root
```

#Answer of question no. 08:
Given adjacency matrix:

	A	B	C	D	E	F	G
A	0	1	0	1	1	0	0
B	0	0	0	0	0	0	1
C	0	1	0	0	0	0	0
D	0	0	1	0	0	0	1
E	0	0	0	0	0	0	0
F	0	0	0	1	0	0	0
G	0	0	0	0	1	1	0

Order of the matrix: 7x7
Number of Nodes: 7
The Graph Representation of the matrix:


A
B
C
D
E
F
G
ID:20301138