CSE220 LAB assignment 8

Name: Asim Baidya
ID: 20301239
Sec: 11

```
# Tree implementation using nodes
class Node:
    def __init__(self, data, left=None, right=None, parent=None):
        self.data = data
        self.left = left
        self.right = right
        self.parent = parent


# binary tree class
class BinaryTree():
    def __init__(self, array):
        self.root = self.create_tree(array)

    def create_tree(self, array, index=1):
        if index < 0 or index >= len(array) or array[index] is None:
            return None
        node = Node(array[index])
        node.left = self.create_tree(array, index * 2)
        node.right = self.create_tree(array, index * 2 + 1)

        if (node.left is not None):
            node.left.parent = node
        if (node.right is not None):
            node.right.parent = node
        return node


# ---------------------------------------
```

```
# task 1
def tree_max(left, right):
    if left > right:
        return left
    return right


def height(root):
    if root is None:
        return -1
    return 1 + tree_max(height(root.left), height(root.right))


# task 2
def level(node):
    if node.parent is None:
        return 0
    return 1 + level(node.parent)


# task 3
def pre_order(node):
    if node is None:
        return
    print(node.data, end=' ')
    pre_order(node.left)
    pre_order(node.right)


# task 4
def in_order(node):
    if node is None:
        return
    pre_order(node.left)
    print(node.data, end=' ')
    pre_order(node.right)


# task 5
```

```
def post_order(node):
    if node is None:
        return
    pre_order(node.left)
    pre_order(node.right)
    print(node.data, end=' ')


# task 6.
def is_same_tree(tree_one, tree_two):
    if tree_one == None and tree_two == None:
        return True
    if tree_one == None and tree_two != None or tree_one != None and tree_two ==
None:
        return False
    if tree_one.data != tree_two.data:
        return False
    con_1 = is_same_tree(tree_one.left, tree_two.left)
    con_2 = is_same_tree(tree_one.right, tree_two.right)
    return con_1 and con_2


# task 7
def copy_tree(node, parent=None):
    if node == None:
        return None

    new_node = Node(node.data)

    left_child = copy_tree(node.left, new_node)
    new_node.left = left_child

    right_child = copy_tree(node.right, new_node)
    new_node.right = right_child

    new_node.parent = parent

    return new_node
```

```
# task 8 done


def get_seven_node_tree(array):
    a, b, c, d, e, f, g = array
    pati = Node(a)

    # pati.left
    pati.left = Node(b)

    # pati.right
    pati.right = Node(c)

    # pati.left.{left,right}
    pati.left.left = Node(d)
    pati.left.right = Node(e)

    # pati.right.{left,right}
    pati.right.left = Node(f)
    pati.right.right = Node(g)
    return pati


if __name__ == "__main__":

    pati1 = get_seven_node_tree([x for x in range(1, 8)])
    pati2 = get_seven_node_tree([x for x in range(1, 8)])
    leti1 = get_seven_node_tree([x for x in range(7)])

    root1 = BinaryTree([0] + [x for x in range(1, 8)])
    root2 = BinaryTree([0] + [x for x in range(1, 1026)])
    root3 = BinaryTree([0] + [x for x in range(1, 1023)])

    # test task 1
    print(f"height: {height(pati1)}")
    print(f"height: {height(root2.root)}")
    print(f"height: {height(root3.root)}")

    # test task 2
    root = root2.root
```

```
    print(f"level: {level(root)}")
    print(f"level: {level(root.left)}")
    print(f"level: {level(root.left.left)}")
    print(f"level: {level(root.left.left.left)}")
    print(f"level: {level(root.left.left.left.left)}")

    # test task 3
    print("PreOrder", "-" * 30)
    pre_order(pati1)
    print()

    # test task 4
    print("InOrder", "-" * 30)
    in_order(root1.root)
    print()

    # test task 5
    print("PostOrder", "-" * 30)
    post_order(pati1)
    print()
    print("PostOrder", "-" * 30)
    post_order(root1.root)
    print()

    # task 6
    print(f"is same tree: {is_same_tree(pati1,pati2)}")
    print(f"is same tree: {is_same_tree(pati1,leti1)}")
    print(f"is same tree: {is_same_tree(pati1,root1.root)}")
    root1.root.left.data = 3000000000000000000000000000000000000000000000
    print(f"is same tree: {is_same_tree(pati1,root1.root)}")
    print(f"is same tree: {is_same_tree(pati1,root2.root)}")

    # task 7
    new_pati1 = copy_tree(pati1)
    print(f"copied same tree? : {is_same_tree(pati1,new_pati1)}")
    print("PostOrder of copied tree", "-" * 30)
    post_order(new_pati1)
    print()

# task 8
```

Answer to the question no 08

a] Equivalent graph for given adjacency matrix.

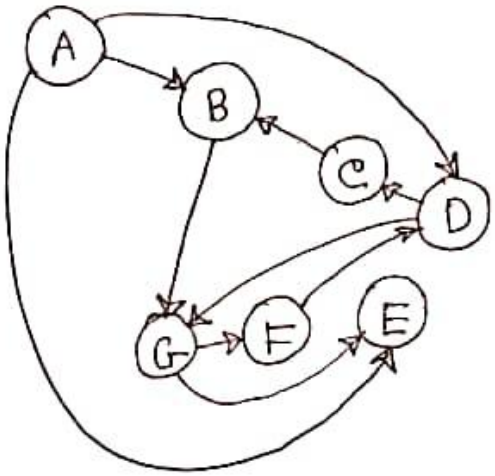

Figure : Directed Graph.