Lab-08

January 1, 2022

[1]:
```
#!/usr/bin/env python3

########################################
##### Name: Abir Ahammed Bhuiyan ########
##### Section: 06                ########
##### ID: 20101197               ########
########################################

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

class BinTree:
    def __init__(self):
        pass

    def height(self, root):
        if root is None:
            return 0
        return 1 + self.maxDepth(self.height(root.left), self.height(root.left))

    def maxDepth(self, left, right):
        if left > right:
            return left
        return right

    def preOrderTrav(self, root):
        if root:
            print(root.datum, end=" ")
            self.preOrderTrav(root.left)
            self.preOrderTrav(root.right)

    def inOrderTrav(self, root):
        if root:
```

```
            self.inOrderTrav(root.left)
            print(root.datum, end=" ")
            self.inOrderTrav(root.right)

    def postOrderTrav(self, root):
        if root:
            self.postOrderTrav(root.left)
            self.postOrderTrav(root.right)
            print(root.datum, end=" ")


    def levelFinder(self, root, key, level=0):
        if root == None:
            return False
        if root.datum == key:
            return level
        flag = self.levelFinder(root.left, key, level+1)

        if flag != False:
            return flag

        return self.levelFinder(root.right, key, level+1)

    def sameChecker(self, root1, root2):
        if root1 == None and root2 == None:
            return True

        if root1 != None and root2 != None:
            return root1.datum == root2.datum and self.sameChecker(root1.left,␣
↪root2.left) and self.sameChecker(root1.right, root2.right)

        return False

    def replicate(self, root):
        if root == None:
            return root

        newRoot = Node(root.datum)

        newRoot.left = self.replicate(root.left)
        newRoot.right = self.replicate(root.right)

        return newRoot
```

```
if __name__ == "__main__":

# imagine something like this
#
#               A-root
#
#       B-n2              C-n3
#
# D-n4      E-n5      F-n6      G-n7

# building the tree
    root = Node('A')
    n2 = Node('B')
    n3 = Node('C')
    n4 = Node('D')
    n5 = Node('E')
    n6 = Node('F')
    n7 = Node('G')

    root.left = n2
    n2.parent = root

    root.right = n3
    n3.parent = root

    n2.left = n4
    n4.parent = n2

    n2.right = n5
    n5.parent = n2

    n3.right = n7
    n7.parent = n3

    n3.left = n6
    n6.parent = n3

    btr = BinTree() # instantiation

    print("############## Task 1 ################")
    print(btr.height(root)) # should print 3

    print("############## Task 2 ################")
    print(btr.levelFinder(root, 'B')) # Should print 1

    print("############## Task 3 ################")
    print("Pre-order Traversal:")
```

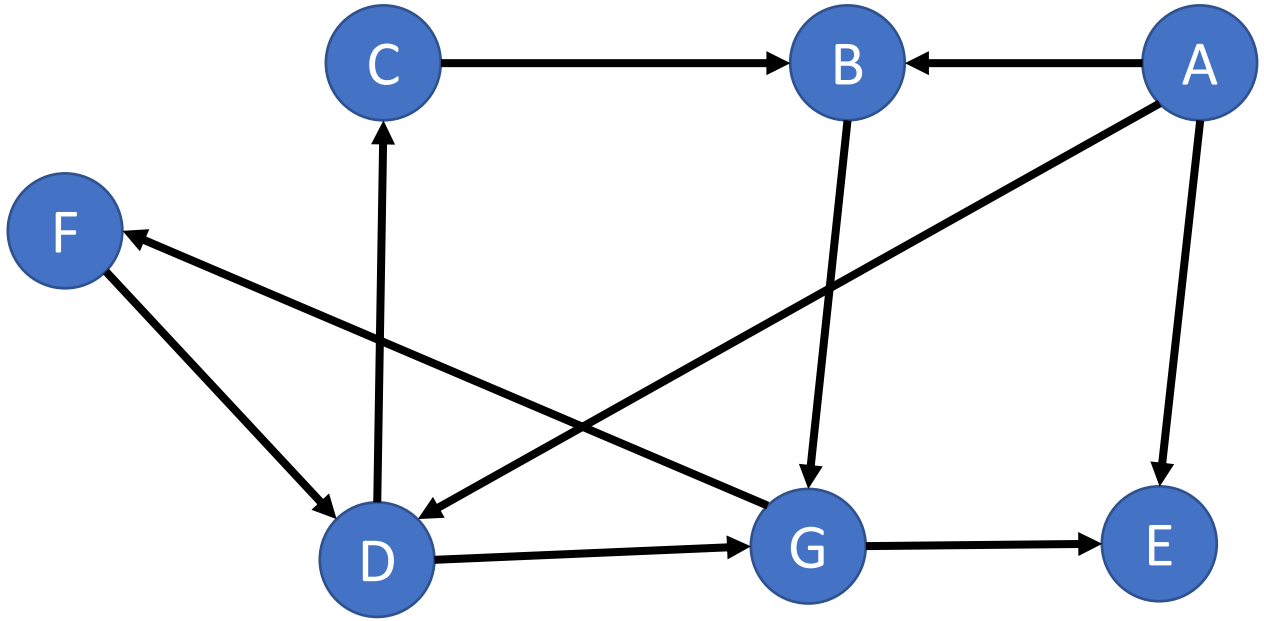
```
    btr.preOrderTrav(root)
    print()

    print("############# Task 4 ###############")
    print("In-order Traversal:")
    btr.inOrderTrav(root)
    print()

    print("############# Task 5 ###############")
    print("Post-order Traversal:")
    btr.postOrderTrav(root)
    print()


    print("############# Task 6 ###############")
    root1 = Node('A')
    root1.left = Node('B')
    root1.right = Node('C')

    root2 = Node('A')
    root2.left = Node('B')
    root2.right = Node('C')

    print(btr.sameChecker(root1, root2)) # should return True

    root1 = Node('A')
    root1.left = Node('B')
    root1.right = Node('C')

    root2 = Node('A')
    root2.left = Node('B')

    print(btr.sameChecker(root1, root2)) # should return False

    print("############# Task 7 ###############")
    newRoot = btr.replicate(root)

    print(newRoot.datum)           # should print A
    print(newRoot.left.left.datum) # should print D
    print(newRoot.right.datum)     # should print C
```

```
############# Task 1 ###############
3
############# Task 2 ###############
1
############# Task 3 ###############
Pre-order Traversal:
```

```
A B D E C F G
############# Task 4 ###############
In-order Traversal:
D B E A F C G
############# Task 5 ###############
Post-order Traversal:
D E B F G C A
############# Task 6 ###############
True
False
############# Task 7 ###############
A
D
C
```

[ ]:

Task 8 (a)



Equivalent graph for that particular adjacency matrix