BRAC UNIVERSITY

CSE220

DATA STRUCTURES

Lab 08

Student Information:

NAME: MD. DANIAL ISLAM ID: 20101534
SECTION: 07

Date: 1 January 2022

```
# Node class
class Node:
    def __init__(self,d,p=None,l=None,r=None):
        self.data = d
        self.parent = p
        self.l = l
        self.r = r
# making tree ds
head = Node("A")
head.l = Node("B",head)
head.r = Node("C",head)
head.l.l = Node("D",head.l)
head.l.r = Node("E",head.l)
head.r.l = Node("F",head.r)
head.r.r = Node("G",head.r)
head.l.l.l = Node("H",head.l.l)
head.l.l.r = Node("I",head.l.l)
head.l.r.l = Node("J",head.l.r)
head.r.l.l = Node("K",head.r.l)
head.r.r.l = Node("L",head.r.r)
head.r.r.r = Node("M",head.r.r)
class Tree_Task:
    # Task 1 : RECURSIVELY calculate the height of a tree.
    @classmethod
    def height_check(cls,tr):
        if tr is None:
            return 0
        return 1 + max(cls.height_check(tr.l),cls.height_check(tr.r))

    # Task 2 : RECURSIVELY calculate the level of a Node in a tree.
    @classmethod
    def level_check(cls,head):
        if head is None:
            return None
        if head.parent is None:
            return 0
        return 1+cls.level_check(head.parent)

    @classmethod
    # Task 3 : Print elements of all the Nodes of a tree using Pre-order
Traversal.
    def pre_order(cls,head):
        if head is not None:
            print("{} ".format(head.data),end="")
            cls.pre_order(head.l)
            cls.pre_order(head.r)
    @classmethod
    # Task 4: Print elements of all the Nodes of a tree using In-order Traversal
    def in_order(cls,head):
    #     if(head.l != None or head.r != None):
    #         in_order(head.l)
    #     print(head.data,end=" ")
    #     if(head.l != None or head.r != None):
    #         in_order(head.r)
```

```
        if head is None:
                return
        cls.in_order(head.l)
        print("{} ".format(head.data),end="")
        cls.in_order(head.r)
    @classmethod
    # Task 5 : Print elements of all the Nodes of a tree using Post-order
Traversal.
    def post_order(cls,head):
        if head is None:
            return
        cls.post_order(head.l)
        cls.post_order(head.r)
        print("{} ".format(head.data),end="")

    @classmethod
    # Task 6 : Write a method which will evaluate whether two trees are exactly
same or not.
    def isExact(cls,head,head2):
        if not(head!=None) and not(head2!=None):
            return True
        if not(head == None and head2 == None) and not(head.data!=head2.data):
            return cls.isExact(head.l,head2.l) and cls.isExact(head.r,head2.r)
        return False

print("\n//=======Task 1 -- RECURSIVELY calculate the height of a tree=======//")
arr_1 = [None,"A","B","C","D","E","F","G","H","I","J",None,"K",None,"L","M"]
print("Array tree:",arr_1)
print('Height is: {}'.format(Tree_Task.height_check(head)))


print("\n//=======Task 2 -- RECURSIVELY calculate the level of a Node in a
tree.=======//")
node = head.r.r.l
print("External Node data : ",node.data)
print("Level of {} is {}".format(node.data,Tree_Task.level_check(node)))
node2 = head.r.r
print("External Node data : ",node2.data)
print("Level of {} is {}".format(node2.data,Tree_Task.level_check(node2)))


print("\n//=======Task 3 -- Print elements of all the Nodes of a tree using
Pre-order Traversal.=======//")
print("Tree: ",arr_1)
print("Pre order Print: ")
Tree_Task.pre_order(head)



print("\n\n//=======Task 4 -- Print elements of all the Nodes of a tree using
In-order Traversal=======//")
print("Tree: ",arr_1)
print("In order Print: ")
Tree_Task.in_order(head)
```

```
print("\n\n//=======Task 5 -- Print elements of all the Nodes of a tree using
Post-order Traversal.=======//")

print("Tree: ",arr_1)
print("Post order Print: ")
Tree_Task.post_order(head)

arr_2 = [None,"A","B","C","D","E","F","G","I","H","J",None,"K",None,"L","M"]

head2 = Node("A")
head2.l = Node("B",head2)
head2.r = Node("C",head2)
head2.l.l = Node("D",head2.l)
head2.l.r = Node("E",head2.l)
head2.r.l = Node("F",head2.r)
head2.r.r = Node("G",head2.r)
head2.l.l.l = Node("I",head2.l.l)
head2.l.l.r = Node("H",head2.l.l)
head2.l.r.l = Node("J",head2.l.r)
head2.r.l.l = Node("K",head2.r.l)
head2.r.r.l = Node("L",head2.r.r)
head2.r.r.r = Node("M",head2.r.r)



print("\n\n//=======Task 6 -- Write a method which will evaluate whether two
trees are exactly same or not.=======//")

print("Tree 1:",arr_1)
print("Tree 2:",arr_2)
print("Tree 1 and Tree 2 Same?:",Tree_Task.isExact(head,head2))

# Task 7: Write a method which will return a copy (new tree) of a given tree.
def copyTree(head):
    while (head.l!= None or head.r!=None):
       pass
```

Task 8