Name: Shouvik Banerjee Argha

ID: 20301118

Sec:10

#Task01

class Node(object):

  def \_\_init\_\_(self, c, lft, rht, pnt):

    self.e = None

    self.left = None

    self.right = None

    self.parent = None

    self.e=c

    self.left=lft

    self.right=rht

    self.parent=pnt

def tree(a, i):

  if i<0 or i>=len(a) or a[i] is None:

    return None

  else:

    root = Node(a[i],None,None,None)

    root.left = tree(a,2\*i)

    root.right = tree(a,2\*i+1)

    if root.left is not None:

      root.left.parent = root

    if root.right is not None:

      root.right.parent = root

    return root

  def max(r\_l, r\_r):

    if r\_l> r\_r:

      return r\_l

    return r\_r

def height(root):

  if root is None:

    return 0

  return 1+max(height(root.left),height(root.right))

arr = [None, 1, 2, 3, 4, 5, None, 6]

tree = tree(arr,1)

print("Height of the tree:",height(tree))

#Task02

class Node(object):

  def \_\_init\_\_(self, c, lft, rht, pnt):

    self.e = None

    self.left = None

    self.right = None

    self.parent = None

    self.e=c

    self.left=lft

    self.right=rht

    self.parent=pnt

def tree(a, i):

  if i<0 or i>=len(a) or a[i] is None:

    return None

  else:

    root = Node(a[i],None,None,None)

    root.left = tree(a,2\*i)

    root.right = tree(a,2\*i+1)

    if root.left is not None:

      root.left.parent = root

    if root.right is not None:

      root.right.parent = root

    return root

  def max(r\_l, r\_r):

    if r\_l> r\_r:

      return r\_l

    return r\_r

def level(n):

  if n.parent is None:

    return 0

  return 1+level(n.parent)

arr = [None, 1, 2, 3, 4, 5, None, 6]

x = tree(arr,1)

print("The level of node: ",level(x.left.right))

#Task03

class Node(object):

  def \_\_init\_\_(self, c, lft, rht, pnt):

    self.e = None

    self.left = None

    self.right = None

    self.parent = None

    self.e=c

    self.left=lft

    self.right=rht

    self.parent=pnt

def tree(a, i):

  if i<0 or i>=len(a) or a[i] is None:

    return None

  else:

    root = Node(a[i],None,None,None)

    root.left = tree(a,2\*i)

    root.right = tree(a,2\*i+1)

    if root.left is not None:

      root.left.parent = root

    if root.right is not None:

      root.right.parent = root

    return root

  def max(r\_l, r\_r):

    if r\_l> r\_r:

      return r\_l

    return r\_r

def preordertraversal(r):

  if r is not None:

    print(r.e)

    preordertraversal(r.left)

    preordertraversal(r.right)

arr = [None, 1, 2, 3, 4, 5, None, 6]

pre = tree(arr,1)

print("The preorder traversal is: ")

preordertraversal(pre)

#Task04

class Node(object):

 def \_\_init\_\_(self, c, lft, rht, pnt):

   self.e = None

   self.left = None

   self.right = None

   self.parent = None

   self.e=c

   self.left=lft

   self.right=rht

   self.parent=pnt

def tree(a, i):

  if i<0 or i>=len(a) or a[i] is None:

    return None

  else:

    root = Node(a[i],None,None,None)

    root.left = tree(a,2\*i)

    root.right = tree(a,2\*i+1)

    if root.left is not None:

      root.left.parent = root

    if root.right is not None:

      root.right.parent = root

    return root

  def max(r\_l, r\_r):

    if r\_l> r\_r:

      return r\_l

    return r\_r

def inordertraversal(r):

  if r is not None:

    inordertraversal(r.left)

    print(r.e)

    inordertraversal(r.right)

arr = [None, 1, 2, 3, 4, 5, None, 6]

in\_ord = tree(arr,1)

print("The Inorder traversal is: ")

inordertraversal(in\_ord)

#Task05

class Node(object):

  def \_\_init\_\_(self, c, lft, rht, pnt):

    self.e = None

    self.left = None

    self.right = None

    self.parent = None

    self.e=c

    self.left=lft

    self.right=rht

    self.parent=pnt

def tree(a, i):

  if i<0 or i>=len(a) or a[i] is None:

    return None

  else:

    root = Node(a[i],None,None,None)

    root.left = tree(a,2\*i)

    root.right = tree(a,2\*i+1)

    if root.left is not None:

      root.left.parent = root

    if root.right is not None:

      root.right.parent = root

    return root

  def max(r\_l, r\_r):

    if r\_l> r\_r:

      return r\_l

    return r\_r

def postordertraversal(r):

  if r is not None:

    postordertraversal(r.left)

    postordertraversal(r.right)

    print(r.e)

arr = [None, 1, 2, 3, 4, 5, None, 6]

post = tree(arr,1)

print("The post order traversal is: ")

postordertraversal(post)

#Task06

def Berlin(x,y):

  i=0

  if len(x)==len(y):

    j=0

    while j<len(x):

      if x[j]==y[j]:

        i=i+1

      else:

        i=i+0

      j=j+1

  if i==len(x) and i==len(y):

    return "Same"

  else:

    return "Not same"

x = [None, 1, 2, 3, 4, 5, None, 6]

y = [None, 1, 2, 3, 4, 5, None, 6]

print("Its Same or not?:")

Berlin(x,y)

#Task07

def copy(a):

    n = [0]\*len(a)

    i = 0

    while i<len(a):

      n[i] = a[i]

      i=i+1

    return n

a = [None, 1, 2, 3, 4, 5, None, 6]

print(copy(a))

**#Task08**

**Equivalent Graph:**

