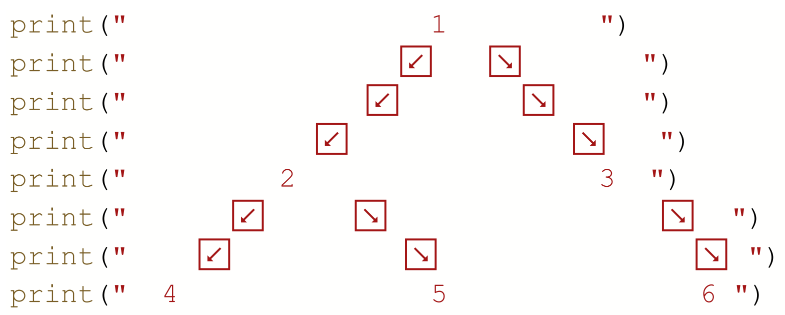
Lab Assignment 08

Name: Moinul Hossain Bhuiyan

Id: 20301002

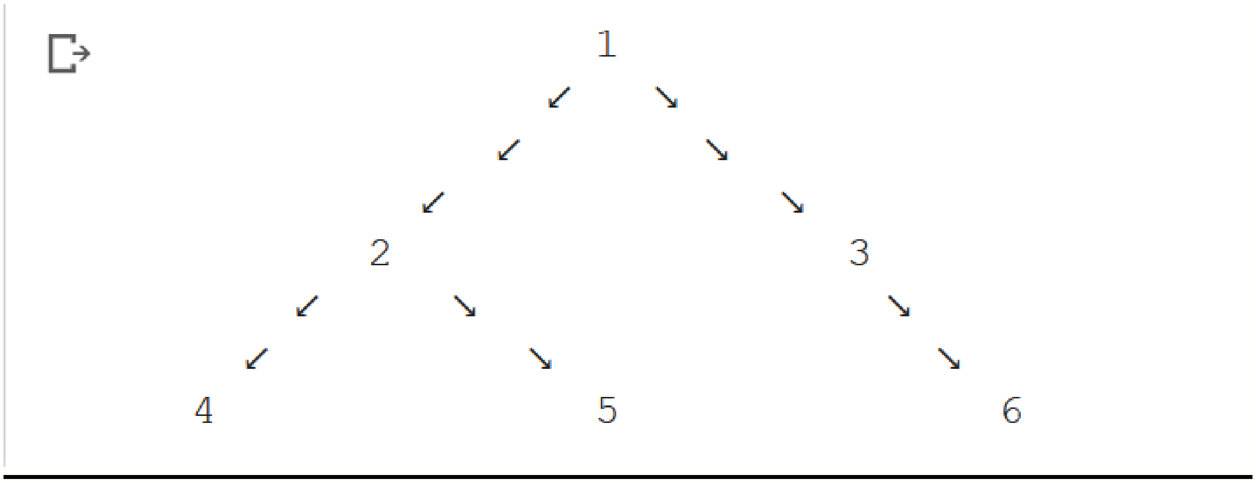
Section06

Visualization Of the tree



```
print("                  1         ")
print("                ↙   ↘          ")
print("              ↙       ↘        ")
print("           ↙             ↘      ")
print("         2                  3  ")
print("      ↙     ↘                ↘      ")
print("    ↙          ↘               ↘     ")
print("  4               5               6 ")
```

The Output Would be:

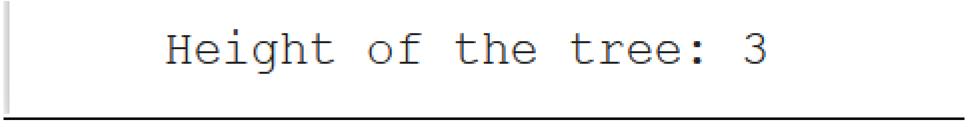


```
                  1
                ↙   ↘
              ↙       ↘
           ↙             ↘
         2                  3
      ↙     ↘                ↘
    ↙          ↘               ↘
  4               5               6
```

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))
```

Output Would Be:

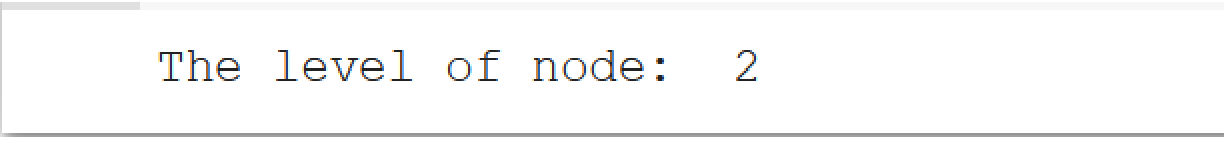


```
Height of the tree: 3
```

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))
```

Output Would Be:

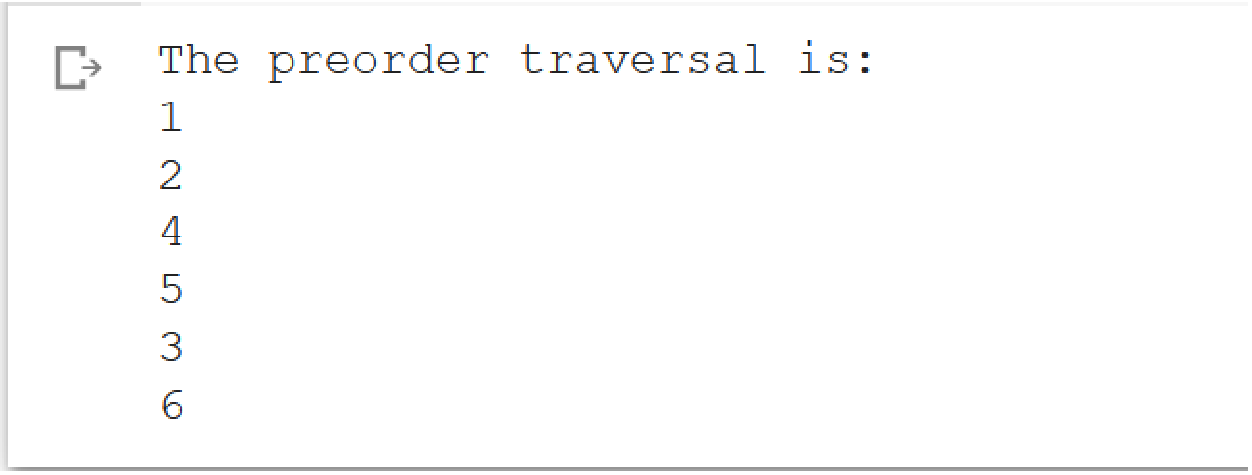


```
The level of node:  2
```

Task03

```python
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)
```

Output Would Be:

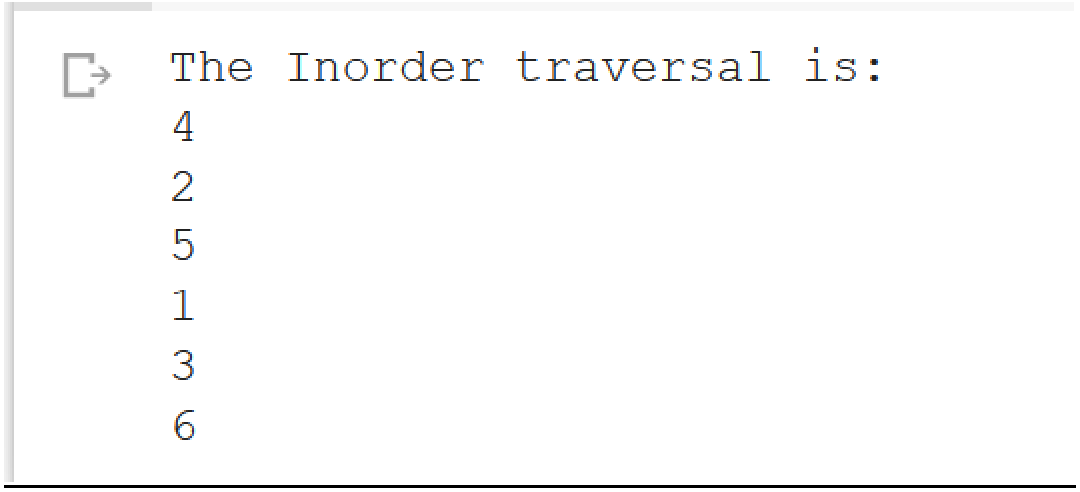


```
The preorder traversal is:
1
2
4
5
3
6
```

Task04

```python
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)
```

Output Would Be:

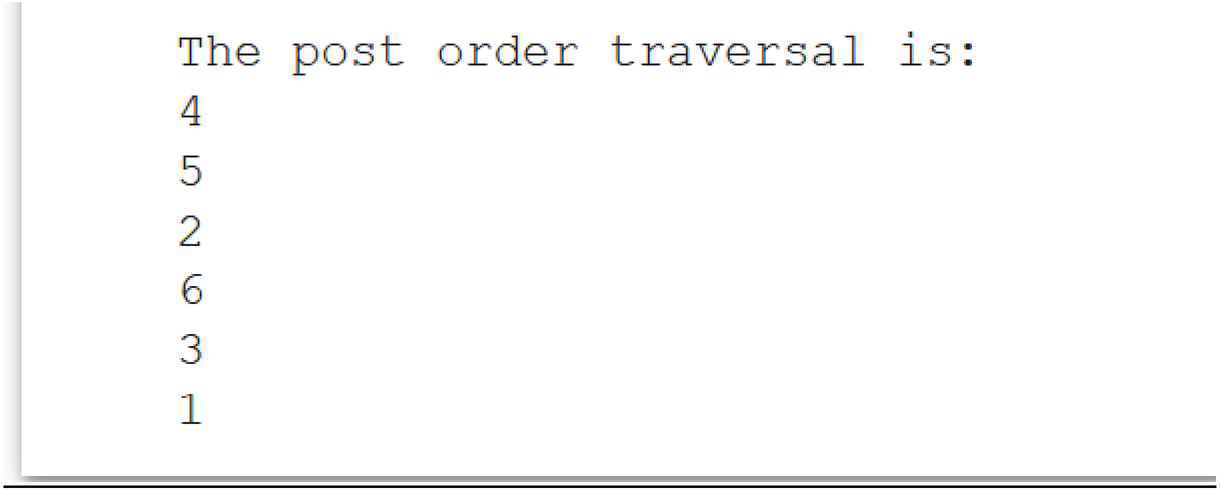


```
The Inorder traversal is:
4
2
5
1
3
6
```

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)
```

Output Would Be:

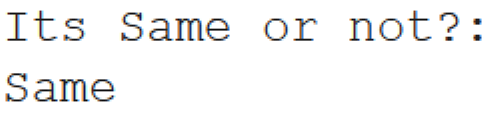


```
The post order traversal is:
4
5
2
6
3
1
```

Task06

```
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 smornt(a, b):
    result =""
    i = 0
    while i<len(a):
        j = 0
        while j<len(b):
            if i==j:
                if a[i] is b[j]:
                    result="Same"
                else:
                    result="Not same"
            j += 1
        i += 1
    print(result)
arr_2 = [None, 1, 2, 3, 4, 5, None, 6]
arr_3 = [None, 1, 2, 3, 4, 5, None, 6]
print("Its Same or not?:")
smornt(arr_3,arr_2)
```

Output Would be:

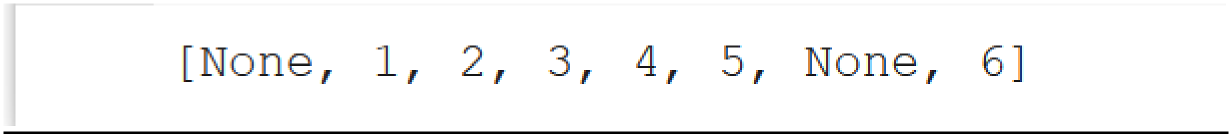


```
Its Same or not?:
Same
```

Task07

```
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 copy(a):
    b = [None for _ in range(len(a))]
    i = 0
    while i<len(a):
        b[i]=a[i]
        i += 1
    return b
arr = [None, 1, 2, 3, 4, 5, None, 6]
print(copy(arr))
```

Output Would Be:



```
[None, 1, 2, 3, 4, 5, None, 6]
```

Task08

Drawing the equivalent graph of the given adjacent matrix: