IARE

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : Muhd fauzan zahaib

Roll Number									
2	3	9	5	1	A	0	5	5	H

Class : CSE-C Semester : II

Course Code : ACSD06 Course Name : PPS Lab

Name of the Course Faculty : Dr. M. Madhusudhan Reddy Faculty ID : IARE 10881

Exercise Number : 06 Week Number : 06 Date :

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab		Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	5.1	Build a Graph							
2	5.2	Number of sink nodes in a graph							
3	5.3	connected components in a Graph	4	4		4	4	4	20
4	5.4	Transpose graph							
5	5.5	counting triplets							
6									
7									
8									
9									
10									
11									
12									

Signature of the Student

Signature of the Faculty

5.2 Number of sink nodes in a graph :-

```
nv = int(input())
ne = int(input())
adj_list = {v:[] for v in range(nv)}
edges = []
for i in range(ne):
    edge = tuple(map(int, input().split()))
    edges.append(edge)
for v1, v2 in edges:
    adj_list[v1].append(v2)
sink_nodes = []
for k,v in adj_list.items():
    if len(adj_list[k]) == 0:
        sink_nodes.append(k)
print(len(sink_nodes))
```

Input :-

```
4
2
3 2
3 4
```

output :-

```
3
```

ROLL NUMBER :

5.3 connected components in a graph:

dfs. def dfs (vertex, visited):

visited [vertex-1] = True

for i in adj_list [vertex]:

if not visited [i-1]:

dfs (i, visited)

nv, ne = list(map(int, input().split()))

visited = [false for i in range (nv)]

adj_list = {v+1: [] for v in range (nv)}

edges = []

for i in range (ne):

edge = tuple (map (int, input().split()))

edges.append (edge)

for v_1, v_2 in edges:

adj_list [v_1].append (v_2)

adj_list [v_2].append [v_1]

component = 0

for i in range (1, nv+1):

if visited [i-1] == false

component += 1

dfs (i, visited)

print (component).

ROLL NUMBER :

input :-

8 5

1 2

2 3

2 4

3 5

6 7

output :-

3

5.4 Transpose Graph

```
nv = int (input())
adj_list = {v:[] for v in range (nv)}
for i in range (nv):
    edge = tuple (map(int, input().split (' : ')))
    k = int (edge [0])
    v = []
    for j in edge [1]:
        if j.isdigit():
            v.append (int (j))
    adj_list [k] = v
trans_adj_list = {v:[] for v in range (nv)}
for k, v in adj_list.items ():
```

```
    for i in v:
        trans_adj_list[i].append(k)
for k,v in trans_adj_list.items():
    print(k,':',v)
```

Input:-

```
4
0 : 1
1 : 2
2 : 3
3 :
```

output :-

```
0 : []
1 : [0]
2 : [1]
3 : [2]
```

5.1 Representation of a graph:-

```
nv, ne = list(map(int, input().split(' ')))
edges = []
for i in range(ne):
    edge = tuple(map(int, input().split(' ')))
    edges.append(edge)
adj_mat = [[0]*nv for i in range(nv)]
adj_list = {v: [] for v in range(nv)}
```

```
for v1, v2 in edges :
        adj_mat [v1] [v2] = 1
        adj_mat  [v2] [v1] = 1
        adj_list [v1]. append (v2)
        adj_list [v2]. append (v1)
print ('Adjacency matrix')
for i in adj_mat:
        for j in i [:-1]:
            print (j, end = ' ')
        print (i[-1])
print ('Adjacency list')
for i,v in adj_list .items():
        print (i,':', sorted (list (set(v))))
```

Input:-

3, 3

0, 1

1, 2

2, 0

output:

Adjacency matrix

0, 1, 1

1, 0, 1

1, 1, 0

Adjacency List

0: [1, 2]

1: [0, 2]

2: [0, 1]

Angeet

ROLL NUMBER :

5.5

```
N, M = list(map(int, input().split()))

edges = []

for i in range(ne):
    edge = tuple(map(int, input().split(' ')))
    edges.append(edge)

adj_mat =

wedges = []

for i in range(1, N):
    for j in range(i+1, N+1):
        if j > i:
            if (i, j) not in edges:
                wedges.append((i, j))

triedges = []

for i in range(1, N+1):
    for j in range(i+1, N+1):
        for k in range(j+1, N+1):
            if ((i, j) in edges) and ((j, k) in edges) and ((i, k in
                                                    edges):
                triedges.append((i, j, k))
            elif ((i, j) in wedges) and ((j, k) in wedge and
                                        ((i, k in wedge):
                triedges.append((i, j, k))

print(len(triedges))
```