**Assignment01\_Task01**

f=open('input.txt',mode='r')

f=f.readlines()

edge=int(f[1]) # total connections

w=len(f)

lina=int(f[w-1]) # position of lina

graph={}

for i in range(2,2+edge):

p=f[i].strip().split()

p1,p2=p

p3=int(p1)

p4=int(p2)

if p3 not in graph.keys():

graph[p3]=list()

graph[p3].append(p4)

def bfs\_modified(graph, start, end):

visited = []

queue = [[start]] #sub-graph

if start == end:

print("Minimum moves=",start)

return

while queue:

p = queue.pop(0) # slicing sub-graph/queue

con = p[-1] # connected nodes of start node ; con=connected

if con not in visited:

n = graph[con] # getting connected edged-nodes of con family

for i in n:

new\_p = list(p)

new\_p.append(i)

queue.append(new\_p)

if i == end:

print("Minimum moves=",len(new\_p)-1)

return

visited.append(con)

bfs\_modified(graph,0,lina)

**Assignment01\_Task02**

f=open('input2.txt',mode='r')

f=f.readlines()

edge=int(f[1]) # total connections

w=len(f)

lara=int(f[w-1]) # position of lina

nora=int(f[w-2]) # position of nora

lina=int(f[w-3]) # position of lina

graph={}

for i in range(2,2+edge):

p=f[i].strip().split()

p1,p2=p

p3=int(p1)

p4=int(p2)

if p3 not in graph.keys():

graph[p3]=list()

graph[p3].append(p4)

def bfs\_modified(graph, start, end):

visited = []

queue = [[start]] #sub-graph

if start == end:

return start

while queue:

p = queue.pop(0) # slicing sub-graph/queue

con = p[-1] # connected nodes of start node ; con=connected

if con not in visited:

n = graph[con] # getting connected edged-nodes of con family

for i in n:

new\_p = list(p)

new\_p.append(i)

queue.append(new\_p)

if i == end:

return len(new\_p)-1

visited.append(con)

nora\_move=bfs\_modified(graph,nora,lina) # total moves of Nora

lara\_move=bfs\_modified(graph,lara,lina) # total moves of Lara

if nora\_move<lara\_move: print("Winner:","Nora")

elif nora\_move>lara\_move: print("Winner:","Lara")

else: print("Both reached at the same time")

**Assignment01\_Task03**

f=open('input3.txt',mode='r')

f=f.readlines()

c=int(f[0])

edge=int(f[1]) # total connections

w=len(f)

lina=int(f[2+edge-1])

participants=int(f[2+edge])

k=list()

for i in range(2+edge+1,w):

k.append(int(f[i]))

graph={}

for i in range(2,2+edge-1):

p=f[i].strip().split()

p1,p2=p

p3=int(p1)

p4=int(p2)

if p4 not in graph.keys():

graph[p4]=list()

graph[p4].append(p3)

m=list()

def bfs\_modified(graph, start,k):

for i in k:

visited = []

queue = [[start]] #sub-graph

while queue:

p = queue.pop(0) # slicing sub-graph/queue

con = p[-1] # connected nodes of start node ; con=connected

if con not in visited :

n = graph[con] # getting connected edged-nodes of con family

for j in n:

new\_p = list(p)

new\_p.append(j)

queue.append(new\_p)

if i == j:

m.append(len(new\_p)-1)

visited.clear()

queue.clear()

new\_p.clear()

bfs\_modified(graph, lina,k) # bfs\_modified method has been called only once

print(min(m))

**Input01(Task\_01)**

9

13

0 1

0 2

0 3

1 3

1 4

2 3

3 5

3 6

4 8

4 7

5 6

6 7

7 8

6

**Input02(Task\_02)**

9

12

0 1

0 2

0 3

1 3

1 4

2 3

3 5

4 8

4 7

5 6

6 7

7 8

7

5

3

**Input03(Task\_03)**

10

14

0 1

0 3

1 3

1 4

2 3

3 5

4 7

4 8

5 6

6 7

6 9

7 8

8 9

9

5

0

1

3

5

7