# Note: I formatted the provided inputs with only the numbers and copied them to a .txt file and worked with those input files. Ex. input.txt, input2.txt

# LEVEL-1’s Code:

#imports

import collections

#reading the input file & making appropriate data for usage

lines=[]

with open("./input.txt") as file:

lines1=file.readlines()

for line in lines1:

l=line.strip("\n")

l=l.strip(" ")

lines.append(l)

#this one is Lina's Position

goalPos=int(lines[-1])

#getting only the edges from input

lines.pop(0)

lines.pop(0)

lines.pop(-1)

#getting edges properly to form a graph

edges=[]

for line in lines:

line=line.split(" ")

line=[int(x) for x in line]

edges.append(line)

graph = collections.defaultdict(list)

#creating the graph

for edge in edges:

e0, e1 = edge[0], edge[1]

graph[e0].append(e1)

graph[e1].append(e0)

#function for calculating minimum number of moves needed

#using BFS traversing and saving the previously visited nodes

def min\_move(graph, startPos, goalPos):

visited = []

queue = [[startPos]]

#traversing the graph using a queue

while queue:

path = queue.pop(0)

currentNode = path[-1]

#checking if the node is visited

if currentNode not in visited:

#getting the neighboring nodes

neighNodes = graph[currentNode]

for neighbour in neighNodes:

#creating paths by traversing the graph

pathNew = list(path)

pathNew.append(neighbour)

queue.append(pathNew)

#checking if it's the goal node

if neighbour == goalPos:

print("Minimum Moves needed : ", len(pathNew)-1)

return

#adding to visited list

visited.append(currentNode)

#CALLING THE FUNCTION

#Nora's position is the startPos

startPos=0

#we got Lina's position earlier from input, goalPos is Lina's position

print("Nora's Position : ", startPos)

print("Lina's Position : ", goalPos)

min\_move(graph, startPos, goalPos)

# 

# LEVEL-2’s Code:

#imports

import collections

#reading the input file & making appropriate data for usage

lines=[]

with open("./input2.txt") as file:

lines1=file.readlines()

for line in lines1:

l=line.strip("\n")

l=l.strip(" ")

lines.append(l)

#this one is Lina's Position

goalPos=int(lines[-3])

#this is Nora's Position

startPos1=int(lines[-2])

#this is Lara's Position

startPos2=int(lines[-1])

#getting only the edges from input

lines.pop(0)

lines.pop(0)

lines.pop(-1)

lines.pop(-1)

lines.pop(-1)

#getting edges properly to form a graph

edges=[]

for line in lines:

line=line.split(" ")

line=[int(x) for x in line]

edges.append(line)

graph = collections.defaultdict(list)

#creating the graph

for edge in edges:

e0, e1 = edge[0], edge[1]

graph[e0].append(e1)

graph[e1].append(e0)

#function for calculating minimum number of moves needed

#using BFS traversing and saving the previously visited nodes

def min\_move(graph, startPos, goalPos):

visited = []

queue = [[startPos]]

#traversing the graph using a queue

while queue:

path = queue.pop(0)

currentNode = path[-1]

#checking if the node is visited

if currentNode not in visited:

#getting the neighboring nodes

neighNodes = graph[currentNode]

for neighbour in neighNodes:

#creating paths by traversing the graph

pathNew = list(path)

pathNew.append(neighbour)

queue.append(pathNew)

#checking if it's the goal node

if neighbour == goalPos:

return len(pathNew)-1

#adding to visited list

visited.append(currentNode)

#CALLING THE FUNCTION FOR BOTH

nora=min\_move(graph, startPos1, goalPos)

lara=min\_move(graph, startPos2, goalPos)

print("Nora's Position : ", startPos1)

print("Lara's Position : ", startPos2)

print("Lina's Position : ", goalPos)

print("Minimum moves needed for Nora : ", nora)

print("Minimum moves needed for Lara : ", lara)

if(nora<lara): winner="Nora"

elif(lara<nora): winner="Lara"

else: winner="Both"

print("\nWinner : ",winner)

# 

# LEVEL-3’s Code:

#imports

import collections

#reading the input file & making appropriate data for usage

lines=[]

with open("./input3.txt") as file:

lines1=file.readlines()

for line in lines1:

l=line.strip("\n")

l=l.strip(" ")

lines.append(l)

#now we take Lina's Position to start from

#this one is Lina's Position

startPos=int(lines.pop(16))

#and we take the others position as GOAL POSITION

goalPosCount=int(lines.pop(16))

goalPosList=[]

for x in range(goalPosCount):

goalPosList.append(int(lines.pop(16)))

#getting only the edges from input

lines.pop(0)

lines.pop(0)

#getting edges properly to form a graph

edges=[]

for line in lines:

line=line.split(" ")

line=[int(x) for x in line]

edges.append(line)

graph = collections.defaultdict(list)

#creating the graph

for edge in edges:

e0, e1 = edge[0], edge[1]

graph[e0].append(e1)

graph[e1].append(e0)

#function for calculating minimum number of moves needed

#using BFS traversing and saving the previously visited nodes

def min\_move(graph, startPos, goalPosList):

visited = []

queue = [[startPos]]

#traversing the graph using a queue

while queue:

path = queue.pop(0)

currentNode = path[-1]

#checking if the node is visited

if currentNode not in visited:

#getting the neighboring nodes

neighNodes = graph[currentNode]

for neighbour in neighNodes:

#creating paths by traversing the graph

pathNew = list(path)

pathNew.append(neighbour)

queue.append(pathNew)

#checking if it's in the goal nodes list

if neighbour in goalPosList:

return len(pathNew)-1

#adding to visited list

visited.append(currentNode)

#CALLING THE FUNCTION FOR ALL

print("Minimum moves needed to win : ", min\_move(graph, startPos, goalPosList))