```
class Graph:
    def __init__(self,nodes):
        self.nodes=nodes
        self.graph={}

    def add_edges(self, parent , child):

        if not self.graph.get(parent) :
            node=set()
            node.add(child)
            self.graph[parent]=node

        else:
            self.graph[parent].add(child)

class BFS():
    def __init__(self,algo_input):
        self.graph=algo_input[0]
        self.source=algo_input[1]
        self.target=algo_input[2]
        self.pop_index=0
        self.run()



    def run(self):
        explored= [[self.source]]
        while explored:

            self.path = explored.pop(self.pop_index)
            last_node = self.path[-1]


            if last_node == self.target:
                self.list_path()
                break


            else:
                for node in self.graph[last_node]:
                    if node not in self.path:
                        directoty = self.path + [node]
                        explored.append(directoty)


    def list_path(self):
        return self.path
```

```
file_1 =open('level_1_file.txt','r')
file_1=list(file_1.readlines())


g=Graph(int(file_1[0]))  #This graph class can take both number and string as a node name
file_1.remove(file_1[0])
edges=int(file_1[0])
file_1.remove(file_1[0])
count=0


for i in file_1:
    if count==edges:
        break
    count=count+1
    edge=i.split()
    g.add_edges(edge[0],edge[-1])
    g.add_edges(edge[1],edge[0])

target_position = file_1[-1].replace("\n",'')

algo_input=(g.graph, '0', target_position)
moves=len(BFS(algo_input).list_path())-1
print(moves)
```

```
file_2 =open('level_2_file.txt','r')
file_2=list(file_2.readlines())

g=Graph(int(file_2[0]))
file_2.remove(file_2[0])
edges=int(file_2[0])
file_2.remove(file_2[0])
count=0

for i in file_2:
    if count==edges:
        break
    count=count+1
    edge=i.split()
    g.add_edges(edge[0],edge[-1])
    g.add_edges(edge[1],edge[0])

position_of_lina = file_2[-3].replace("\n",'')
```

```
position_of_nora = file_2[-2].replace("\n",'')

position_of_lara = file_2[-1].replace("\n",'')


nora_to_lina = (g.graph, "1", position_of_lina)

lara_to_lina = (g.graph, position_of_lara, position_of_lina)


moves_of_nora=len(BFS(nora_to_lina).list_path())-1

moves_of_lara=len(BFS(lara_to_lina).list_path())-1


print("Nora" if moves_of_nora < moves_of_lara else "Lara")
```

```
import networkx as nx

def Modified_BFS(G,compitators,target):

    min_moves=1000

    for i in competitors:

        new_min=len(nx.shortest_path(G,i,target))

        min_moves=min(min_moves,new_min)


    return min_moves-1




file_3 =open('level_3_file.txt','r')

file_3=list(file_3.readlines())

number_of_nodes=int(file_3[0])

file_3.remove(file_3[0])

number_edge=int(file_3[0])

file_3.remove(file_3[0])

count=0
```

```
edges=[]
for i in file_3:
    if count==number_edge:
        break
    count=count+1
    edge=i.split()
    edges.append(tuple(edge))


G=nx.Graph()
G.add_edges_from(edges)
file_3=file_3[count:]
position_of_lina=file_3[0].replace("\n","")

competitors=[]
for i in file_3[1:]:
    competitors.append(i.replace("\n",""))
print(Modified_BFS(G,competitors,position_of_lina))
```