Level 1:

```
if __name__ == '__main__':
    with open("input1") as f:
        nodes = int(f.readline())
        edges = int(f.readline())
        #print(nodes, edges)
        graph = [[] for i in range(nodes)]

        for i in range(edges):
            u, v = map(int, f.readline().split())
            graph[u].append(v)
            graph[v].append(u)
        #for i in range(nodes):
            #print(i, "=>", graph[i])

        goal = int(f.readline())
        #print(goal)

        queue = [0]
        visitedlist = [False]*nodes

        distancelist = [0]*nodes

        #print(visitedlist)

        while queue:
            node = queue.pop(0)
            #print(node)
            if(node == goal):
                print(distancelist[node])

            else:

                for x in range(len(graph[node])):
                    if(visitedlist[graph[node][x]]==False):
                        visitedlist[graph[node][x]] = True
                        distancelist[graph[node][x]] = distancelist[node]+1
                        queue.append(graph[node][x])
```

Level 2:

```
if __name__ == '__main__':
    with open("input1") as f:
        nodes = int(f.readline())
        edges = int(f.readline())
        #print(nodes, edges)
        graph = [[] for i in range(nodes)]

        for i in range(edges):
            u, v = map(int, f.readline().split())
            graph[u].append(v)
            graph[v].append(u)
        #for i in range(nodes):
            #print(i, "=>", graph[i])

        goal = int(f.readline())
        #print(goal)

        norapos = int(f.readline())
        larapos = int(f.readline())

        def bfs(pos,goal,graph):
            queue = [pos]
            visitedlist = [False]*nodes

            distancelist = [0]*nodes

            #print(visitedlist)

            moves=0
            while queue:
                node = queue.pop(0)
                #print(node)
                if(node == goal):
                    moves= distancelist[node]
                    return moves

                else:

                    for x in range(len(graph[node])):
                        if(visitedlist[graph[node][x]]==False):
                            visitedlist[graph[node][x]] = True
                            distancelist[graph[node][x]] = distancelist[node]+1
```

```
                    queue.append(graph[node][x])

    # FOR FINDING NORA'S MOVES TO REACH THE GOAL
    noramoves = bfs(norapos, goal, graph)
    # FOR FINDING LARA'S MOVES TO REACH THE GOAL
    laramoves = bfs(larapos, goal, graph)
    #print(noramoves)
    #print(laramoves)

    if(norapos>larapos):
        print("Nora")
    elif(norapos==larapos):
        print("Both")
    else:
        print("lara")
```

Level 3:

```
if __name__ == '__main__':
    with open("input1") as f:
        nodes = int(f.readline())
        edges = int(f.readline())
        #print(nodes, edges)
        graph = [[] for i in range(nodes)]

        for i in range(edges):
            u, v = map(int, f.readline().split())
            #graph[u].append(v)
            graph[v].append(u)
        #for i in range(nodes):
            #print(i, "=>", graph[i])

        goal = int(f.readline())  #Lina's position
        #print(goal)

        numberofparticipants = int(f.readline())
        participants=[0]*numberofparticipants

        for x in range(numberofparticipants):
            participants[x]= int(f.readline())
        #print(participants)

        def bfs(pos,participants,graph):
```

```
    queue = [pos]
    visitedlist = [False]*nodes

    distancelist = [0]*nodes

    #print(visitedlist)

    moves=0
    while queue:
        node = queue.pop(0)
        #print(node)
        if(node in participants):
            moves= distancelist[node]
            #print('k',node,'will kill first')
            return moves

        else:

            for x in range(len(graph[node])):
                if(visitedlist[graph[node][x]]==False):
                    visitedlist[graph[node][x]] = True
                    distancelist[graph[node][x]] = distancelist[node]+1
                    queue.append(graph[node][x])

movestokill = bfs(goal,participants,graph)
print(movestokill)
```