*#.................................level01.....start.............................................................*def BFS(graph, nodes, source, des):  
 visited = []  
 distance = []  
 queue = []  
  
 for i in range(0, nodes, 1):  
 visited.append(False)  
 distance.append(-100)  
  
  
  
 visited[source] = True  
 distance[source] = 0;  
 queue.append(source)  
  
 while len(queue) != 0:  
 dq = queue.pop()  
 for i in range(0, len(graph[dq]), 1):  
 if visited[graph[dq][i]] == False:  
 visited[graph[dq][i]] = True  
 distance[graph[dq][i]] = distance[dq] + 1  
 queue.append(graph[dq][i])  
  
  
  
 return distance[des]  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 with open(**'level1.txt'**) as f:  
 nodes = int(f.readline())  
 edges = int(f.readline())  
 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)  
  
 x = int(f.readline())  
  
 source = 0  
 minMoves = BFS(graph, nodes, source, x)  
 print(minMoves)  
  
  
*#................................END.....................................................*

*#....................................level02.........start.......................................................................*def BFS(graph, nodes, source, des):  
 visited = []  
 distance = []  
 queue = []  
  
 for i in range(0, nodes, 1):  
 visited.append(False)  
 distance.append(-100)  
  
  
  
 visited[source] = True  
 distance[source] = 0;  
 queue.append(source)  
  
 while len(queue) != 0:  
 dq = queue.pop()  
 for i in range(0, len(graph[dq]), 1):  
 if visited[graph[dq][i]] == False:  
 visited[graph[dq][i]] = True  
 distance[graph[dq][i]] = distance[dq] + 1  
 queue.append(graph[dq][i])  
  
  
  
 return distance[des]  
  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 with open(**'level2.txt'**) as f:  
 nodes = int(f.readline())  
 edges = int(f.readline())  
 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)  
  
  
 x = int(f.readline())  
  
 p = int(f.readline())  
 q = int(f.readline())  
  
  
 Nora = BFS(graph, nodes,p, x)  
 Lara = BFS(graph, nodes,q, x)  
  
  
 if Nora < Lara:  
 print(**"Nora"**)  
 else:  
 print(**"Lara"**)  
  
*#............................................END..........................................................*

*#......................................level03................start...................................*def BFS(graph, nodes, source):  
 visited = []  
 distance = []  
 queue = []  
  
 for i in range(0, nodes, 1):  
 visited.append(False)  
 distance.append(-100)  
  
 visited[source] = True  
 distance[source] = 0;  
 queue.append(source)  
  
 while len(queue) != 0:  
 dq = queue.pop()  
 for i in range(0, len(graph[dq]), 1):  
 if visited[graph[dq][i]] == False:  
 visited[graph[dq][i]] = True  
 distance[graph[dq][i]] = distance[dq] + 1  
 queue.append(graph[dq][i])  
  
 return distance  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 with open(**'level3.txt'**) as f:  
 nodes = int(f.readline())  
 edges = int(f.readline())  
 graph = [[] for i in range(nodes)]  
  
 for i in range(edges):  
 u, v = map(int, f.readline().split())  
 graph[v].append(u)  
  
 x = int(f.readline())  
 k = int(f.readline())  
  
 participant = []  
  
 for i in range(0, k, 1):  
 participant.append(int(f.readline()))  
  
 distance = BFS(graph, nodes, x)  
 minMove = 1000000;  
 for i in range(0, len(participant), 1):  
 if distance[participant[i]] <= minMove:  
 minMove = distance[participant[i]]  
  
 print(minMove)  
  
*#.......................................................END............................................................*