```
from collections import deque, defaultdict

# Create graph
def build_social_graph():
    graph = defaultdict(list)
    n = int(input("Enter number of people: "))
    names = []
    for _ in range(n):
        name = input("Enter name: ").strip()
        names.append(name)

    m = int(input("Enter number of friendship connections: "))
    print("Enter friendships as pairs (e.g., Alice Bob):")
    for _ in range(m):
        u, v = input().split()
        graph[u].append(v)
        graph[v].append(u)

    return graph, names

# BFS for friend recommendations (level 2 only)
def bfs_recommendations(graph, person):
    visited = set()
    queue = deque([(person, 0)])
    visited.add(person)
    recommendations = set()

    while queue:
        current, level = queue.popleft()
        if level == 2:
            recommendations.add(current)
        if level > 2:
            break
        for neighbor in graph[current]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, level + 1))

    if person in recommendations:
        recommendations.remove(person)
    direct_friends = set(graph[person])
    return recommendations - direct_friends

# DFS for full social circle
```

```
def dfs(graph, person, visited=None):
    if visited is None:
        visited = set()
    visited.add(person)
    for neighbor in graph[person]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
    return visited

# Main
graph, people = build_social_graph()
print("\nPeople in Network:", ', '.join(people))

target = input("Enter person to analyze: ")

print("\n🔍 DFS - Social Circle of", target)
social_circle = dfs(graph, target)
print("Social Circle:", ', '.join(social_circle))

print("\n🤝 BFS - Friend Recommendations for", target)
suggestions = bfs_recommendations(graph, target)
if suggestions:
    print("Recommended Friends:", ', '.join(suggestions))
else:
    print("No recommendations (all friends already connected or too far).")
```