CSE222 / BİL505
Data Structures and Algorithms
Homework #8 – Report

Person Class:

The Person class is designed to represent an individual in a social network. Each Person object contains attributes such as name, age, list of hobbies, and a timestamp indicating when the person was created.

Attributes

The Person class has the following attributes:

- name: A String representing the name of the person.
- age: An int representing the age of the person.
- hobbies: A List<String> representing the hobbies of the person.
- timestamp: A Date object indicating when the Person object was created.

SocialNetworkGraph Class:

The SocialNetworkGraph class represents a social network where each individual, referred to as a Person, can form friendships with other individuals. The functionalities provided by this class to manage the network, including adding/removing people, managing friendships, finding shortest paths, counting clusters, and suggesting friends.

Person Management

Adding a Person

The addPerson method adds a new person to the social network. Each person is represented by a Person object containing their name, age, and a list of hobbies.

```
public void addPerson(String name, int age, List<String> hobbies) {
    Person person = new Person(name, age, hobbies);
    people.put(name, person);
    friendships.put(person, new ArrayList<>());
    System.out.println("Person added: " + person);
}
```

Removing a Person

The removePerson method removes a person and all associated friendships from the network.

```
public void removePerson(String name) {  1 usage
    Person p = people.get(name);
    if (p == null) {
        System.out.println("Person not found: " + name);
        return;
    }
    for (Person person : friendships.keySet()) {
        friendships.get(person).remove(p);
    }
    friendships.remove(p);
    people.remove(name);
    System.out.println("Person removed: " + p.name);
}
```

Friendship Management

Adding a Friendship

The addFriendship method creates a bidirectional friendship between two people.

```
public void addFriendship(String name1, String name2) {  1 usage
    Person person1 = people.get(name1);
    Person person2 = people.get(name2);
    if (person1 != null && person2 != null) {
        friendships.get(person1).add(person2);
        friendships.get(person2).add(person1);
        System.out.println("Friendship added between " + person1.name + " and " + person2.name);
    } else {
        System.out.println("One or both persons not found in the network.");
    }
}
```

Removing a Friendship

The removeFriendship method removes an existing friendship between two people.

```
public void removeFriendship(String name1, String name2) {  1 usage
    Person person1 = people.get(name1);
    Person person2 = people.get(name2);
    if (person1 != null && person2 != null) {
        if (friendships.containsKey(person1) && friendships.get(person1).contains(person2)) {
            friendships.get(person1).remove(person2);
        }
        if (friendships.containsKey(person2) && friendships.get(person2).contains(person1)) {
            friendships.get(person2).remove(person1);
        }
        System.out.println("Friendship removed between " + person1.name + " and " + person2.name);
    } else {
        System.out.println("One or both persons not found in the network.");
    }
}
```

Network Analysis

There are two bfs function one of used to stop at given node(person) and other one visit all nodes until there are no visited one.

```
private void bfs(Person start, Set<Person> visited, List<Person> cluster) {  1 usage
    Queue<Person> queue = new LinkedList<>();
    queue.add(start);
    visited.add(start);

    while (!queue.isEmpty()) {
        Person current = queue.poll();
        cluster.add(current);

        for (Person neighbor : friendships.get(current)) {
            if (!visited.contains(neighbor)) {
                queue.add(neighbor);
                visited.add(neighbor);
            }
        }
    }
}
```

```java
private void bfs(Person start, Person end) {  1 usage
    Map<Person, Person> previous = new HashMap<>();
    Queue<Person> queue = new LinkedList<>();
    Set<Person> visited = new HashSet<>();
    queue.add(start);
    visited.add(start);
    boolean found = false;
    while (!queue.isEmpty() && !found) {
        Person current = queue.poll();

        for (Person neighbor : friendships.get(current)) {
            if (!visited.contains(neighbor)) {
                visited.add(neighbor);
                previous.put(neighbor, current);
                queue.add(neighbor);

                if (neighbor.equals(end)) {
                    found = true;
                    break;
                }
            }
        }
    }
    if (found) {
        printPath(start, end, previous);
    } else {
        System.out.println("No path found between " + start.name + " and " + end.name);
    }
}
```

Finding the Shortest Path

The findShortestPath method uses Breadth-First Search (BFS) to find and print the shortest path between two people.

```java
public void findShortestPath(String startName, String endName) {  1 usage
    Person start = people.get(startName);
    Person end = people.get(endName);
    if (start != null && end != null) {
        bfs(start, end);
    } else {
        System.out.println("People not found.");
    }
}
```

Counting Clusters

The countClusters method identifies and counts clusters of people in the network. Each cluster represents a connected component.

```
public List<List<Person>> countClusters() {  1 usage
    Set<Person> visited = new HashSet<>();
    List<List<Person>> clusters = new ArrayList<>();

    for (Person person : friendships.keySet()) {
        if (!visited.contains(person)) {
            List<Person> cluster = new ArrayList<>();
            bfs(person, visited, cluster);
            clusters.add(cluster);
        }
    }
    printCluster(clusters);
    return clusters;
}
```

Suggesting Friends

The suggestFriends method recommends friends for a given person based on mutual friends and common hobbies.

Calculate the score as follows:

– Each mutual friend contributes 1 point.

– Each common hobby contributes 0.5 points.

– Total score = (number of mutual friends) * 1 + (number of common hobbies) * 0.5.

It sort the candidates by their scores in descending order.

It display the top N suggested friends with the score breakdown.

```java
public void suggestFriends(String personName, int maxSuggestions) {  1 usage
    Person person = people.get(personName);
    if (person == null) {
        System.out.println("Person not found.");
        return;
    }

    Map<Person, Double> scores = new HashMap<>();

    for (Person candidate : friendships.keySet()) {
        if (!candidate.equals(person) && !friendships.get(person).contains(candidate)) {
            int mutualFriends = 0;
            for (Person friend : friendships.get(person)) {
                if (friendships.get(candidate).contains(friend)) {
                    mutualFriends++;
                }
            }

            Set<String> commonHobbies = new HashSet<>(person.hobbies);
            commonHobbies.retainAll(candidate.hobbies);
            double score = mutualFriends * 1 + commonHobbies.size() * 0.5;

            if (score > 0) {
                scores.put(candidate, score);
            }
        }
    }

    List<Map.Entry<Person, Double>> sortedCandidates = new ArrayList<>(scores.entrySet());
    sortedCandidates.sort((a, b) -> Double.compare(b.getValue(), a.getValue()));

    System.out.println("Friend suggestions for " + personName + ":");
    for (int i = 0; i < Math.min(maxSuggestions, sortedCandidates.size()); i++) {
        Map.Entry<Person, Double> entry = sortedCandidates.get(i);
        System.out.println(entry.getKey().name + " (Score: " + entry.getValue() + ")");
    }
}
```

The SocialNetworkGraph class offers a comprehensive set of methods for managing a social network. The functionality includes adding and removing individuals, managing friendships, finding shortest paths, counting clusters, and suggesting potential friends. These features are implemented using fundamental graph algorithms such as Breadth-First Search (BFS) to ensure efficient operations on the social network.

Main Class:

The main method is the core of the Main class. It initializes a SocialNetworkGraph object and a Scanner object to read user input. The method then enters an infinite loop, displaying a menu of options to the user and handling their input accordingly.

Screenshoot of some testing :

1)Add Person

```
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 1
Enter name: Memo
Enter age: 26
Enter hobbies (comma separated): hiking
Person added: Memo (Age: 26, Hobbies: [hiking])
```

2)Remove person

```
Number of cluster found: 2
Cluster 1:
Mert
Memo
Esra
Cluster 2:
Cano
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 2
Enter name of the person to remove: Cano
Person removed: Cano
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 7
Number of cluster found: 1
Cluster 1:
Mert
Memo
Esra
```

3)Add friendship

```
Number of cluster found: 2
Cluster 1:
NewPerson
Cluster 2:
Mert
Memo
Esra
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 3
Enter the name of the first person: NewPerson
Enter the name of the second person: Mert
Friendship added between NewPerson and Mert
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 7
Number of cluster found: 1
Cluster 1:
NewPerson
Mert
Memo
Esra
```

4)Remove frirendship

```
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 4
Enter the name of the first person: NewPerson
Enter the name of the second person: Mert
Friendship removed between NewPerson and Mert
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 7
Number of cluster found: 2
Cluster 1:
NewPerson
Cluster 2:
Mert
Memo
Esra
```

5)Find shortest path

```
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 5
Enter the name of the start person: Mert
Enter the name of the end person: Esra
Shortest path: Mert -> Memo -> Esra
```

6)Suggest friend – Esra's 1 points comes from mutual friend and NewPerson 0.5 points come from common hobbies.

```
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 6
Enter the name of the person: Mert
Enter the maximum number of suggestions: 2
Friend suggestions for Mert:
Esra (Score: 1.0)
NewPerson (Score: 0.5)
```

7)Count cluster – NewPerson added new and there is no friend so there are 2 cluster.

```
===== Social Network Analysis Menu =====
1. Add person
2. Remove person
3. Add friendship
4. Remove friendship
5. Find shortest path
6. Suggest friends
7. Count clusters
8. Exit
Please select an option: 7
Number of cluster found: 2
Cluster 1:
NewPerson
Cluster 2:
Mert
Memo
Esra
```