CSE 222 HOMEWORK #8

Report on Social Network CLI Assignment

Deniz Altınışık

210104004064

Overview

This assignment involved creating a social network application using Java, which allows users to add, remove, and manage people and their friendships. The application also provides functionalities for finding the shortest path between two people, suggesting friends, and counting clusters within the network. Additionally, the project includes a Makefile for building the Java project and managing Javadoc documentation.

Code Structure

The project is divided into three main Java classes and one makefile:

1. **Person**: Represents an individual in the social network.
2. **SocialNetwork**: Manages the people and friendships within the network.
3. **SocialNetworkCLI**: Provides a command-line interface for interacting with the social network.
4. **Makefile**: Compiles and runs the whole program.

Person Class

The **Person** class contains fields for storing an individual's name, age, hobbies, and the timestamp when they joined the network. It includes standard getter methods, **toString**, **equals**, and **hashCode** methods.

SocialNetwork Class

The **SocialNetwork** class maintains a map of people and their friendships. It provides methods for adding and removing people, managing friendships, finding the shortest path between two people, suggesting friends, and counting clusters in the network.

SocialNetworkCLI Class

The **SocialNetworkCLI** class provides a command-line interface for users to interact with the **SocialNetwork** class. It includes methods for displaying the menu, adding/removing people, managing friendships, finding the shortest path, suggesting friends, and counting clusters.

Makefile

The Makefile automates the compilation and execution of the Java program, as well as the generation and cleanup of Javadoc documentation.

Finding the Shortest Path

Method: findShortestPath

The **findShortestPath** method in the **SocialNetwork** class is designed to find the shortest path between two people in the social network. This method uses a breadth-first search (BFS) algorithm, which is well-suited for finding the shortest path in an unweighted graph. Here's a detailed explanation of how it works:

1. **Input Parameters**:

    - **name1** and **timestamp1**: The name and timestamp of the starting person.
    - **name2** and **timestamp2**: The name and timestamp of the ending person.

2. **Key Generation**:

    - The method combines the person's name and timestamp to create unique keys (**key1** for the start person and **key2** for the end person). This ensures that each person is uniquely identified even if there are multiple people with the same name.

3. **Existence Check**:

    - The method checks if both **key1** and **key2** exist in the **people** map. If either key is not found, it returns an empty list, indicating that the path cannot be found.

4. **BFS Initialization**:

    - A queue is initialized with the starting person.
    - A **previous** map is used to keep track of the path by storing the previous person for each visited person.
    - A **visited** set is used to track visited nodes to prevent revisiting and loops.

5. **BFS Execution**:

    - The method processes nodes in the queue iteratively. For each person, it retrieves their neighbors from the adjacency list (**graph**).
    - If the end person is reached, the loop breaks.

6. **Path Reconstruction**:

    - After the BFS loop, the method reconstructs the path from the **end** person to the **start** person using the **previous** map.
    - The path is constructed in reverse (from end to start), so the method reverses it before returning.

7. **Result**:

    - If the start person is at the beginning of the path, the method returns the constructed path.
    - Otherwise, it returns an empty list, indicating no path was found.

Suggesting Friends

Method: suggestFriends

The **suggestFriends** method provides friend suggestions for a given person based on mutual friends and shared hobbies. Here's a detailed breakdown of how it works:

1. **Input Parameters**:
    - **name**: The name of the person for whom friends are being suggested.
    - **timestamp**: The timestamp when the person joined the network.
    - **maxSuggestions**: The maximum number of friends to suggest.
2. **Key Generation**:
    - Similar to the **findShortestPath** method, a unique key is generated by combining the person's name and timestamp.
3. **Existence Check**:
    - The method checks if the key exists in the **people** map. If the person is not found, it returns an empty list.
4. **Initialization**:
    - A **scoreMap** is initialized to store potential friends and their scores.
    - The current person's hobbies are retrieved.
5. **Score Calculation**:
    - The method iterates over all people in the network. For each potential friend (excluding the current person and their direct friends), it calculates a score based on mutual friends and shared hobbies.
    - Each mutual friend increases the score by 1.
    - Each shared hobby increases the score by 0.5.
6. **Sorting and Selection**:
    - The potential friends are sorted in descending order based on their scores.
    - The method selects up to **maxSuggestions** highest-scoring friends.
7. **Result**:
    - The method returns a list of suggested friends along with their scores.

Counting Clusters

Method: countClusters

The **countClusters** method identifies and counts the number of clusters (connected components) in the social network. Here's a detailed explanation of how it works:

1. **Initialization**:
    - A **visited** set is used to track visited people.
    - A **clusterCount** variable is initialized to zero.
    - A **clusters** list is initialized to store the names of people in each cluster.
2. **Cluster Detection**:
    - The method iterates over all people in the network. For each unvisited person, it initiates a breadth-first search (BFS) to explore the entire cluster.
    - Each new BFS initiation corresponds to discovering a new cluster, so the **clusterCount** is incremented.
    - Empty clusters(a person with no friends) are included in the count.
3. **BFS Execution**:
    - The BFS explores all reachable people from the starting person and marks them as visited.
    - The names of people in the current cluster are collected and added to the **clusters** list.
4. **Result**:
    - The method prints the total number of clusters and the members of each cluster.

Graph Methods

Managing Friendships

The **SocialNetwork** class provides methods for managing friendships, including adding and removing friendships.

Method: addFriendship

1. **Input Parameters**:
    - **name1** and **timestamp1**: The name and timestamp of the first person.
    - **name2** and **timestamp2**: The name and timestamp of the second person.
2. **Key Generation**:
    - Unique keys are generated for both people by combining their names and timestamps.
3. **Existence Check**:
    - The method checks if both keys exist in the **people** map. If either person is not found, it prints an error message.
4. **Adding Friendship**:
    - If both people are found, the method updates the adjacency list (**graph**) to add each person as a friend to the other.

Method: removeFriendship

1. **Input Parameters**:
    - **name1** and **timestamp1**: The name and timestamp of the first person.
    - **name2** and **timestamp2**: The name and timestamp of the second person.
2. **Key Generation**:
    - Unique keys are generated for both people by combining their names and timestamps.
3. **Existence Check**:
    - The method checks if both keys exist in the **people** map. If either person is not found, it prints an error message.

4. **Removing Friendship**:
    - If both people are found, the method updates the adjacency list (**graph**) to remove each person from the other's friend list.

Adding and Removing People

Method: addPerson

1. **Input Parameters**:
    - **name**: The person's name.
    - **age**: The person's age.
    - **hobbies**: A list of the person's hobbies.
    - **timestamp**: The date and time when the person joined the network.
2. **Creating and Storing Person**:
    - A new **Person** object is created and stored in the **people** map using a unique key.
    - An empty list is added to the **graph** map to represent the person's friends.
3. **Confirmation Message**:
    - The method prints a confirmation message indicating that the person was added.

Method: removePerson

1. **Input Parameters**:
    - **name**: The name of the person to be removed.
    - **timestamp**: The timestamp when the person joined the network.
2. **Key Generation**:
    - A unique key is generated by combining the person's name and timestamp.
3. **Existence Check**:
    - The method checks if the key exists in the **people** map. If the person is not found, it prints an error message.

4. **Removing Person**:
    - If the person is found, they are removed from the **people** map and the **graph** map.
    - The method also removes the person from the friend lists of all other people in the network.

Makefile Usage

- **make:** This command will compile all .**java** files and generate corresponding **.class** files.
- **make clean:** This command will remove all generated **.class** files.
- **make javadoc:** This command will generate Javadoc documentation for the Java source file.
- **make cleandoc:** This command will remove the generated Javadoc documentation.

Screenshots

```
Please select an option: 1
Enter name: deniz
Enter age: 23
Enter hobbies (comma-separated): fishing,reading
Person added: deniz (Timestamp: 2024-05-29 22:10:00)
```

```
Please select an option: 1
Enter name: Mert
Enter age: 20
Enter hobbies (comma-separated): writing,football,tennis
Person added: Mert (Timestamp: 2024-05-29 22:11:46)
```

```
Please select an option: 1
Enter name: Cagri
Enter age: 22
Enter hobbies (comma-separated): football,gym,music
Person added: Cagri (Timestamp: 2024-05-29 22:12:48)
```

```
Please select an option: 1
Enter name: Seyma Nur
Enter age: 24
Enter hobbies (comma-separated): gym,swimming,reading,football
Person added: Seyma Nur (Timestamp: 2024-05-29 22:14:12)
```

Added 4 person.

```
Please select an option: 3
Enter first person's name: deniz
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:10:00
Enter second person's name: Mert
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:11:46
Add friendship keys: deniz_2024-05-29 22:10:00, Mert_2024-05-29 22:11:46
Friendship added between deniz and Mert
```

Friendship between deniz and mert

```
Please select an option: 3
Enter first person's name: Cagri
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:12:48
Enter second person's name: Seyma Nur
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:14:12
Add friendship keys: Cagri_2024-05-29 22:12:48, Seyma Nur_2024-05-29 22:14:12
Friendship added between Cagri and Seyma Nur
```

Friendship between Çağrı and Şeyma Nur

```
Please select an option: 6
Enter person's name: deniz
Enter person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:10:00
Enter maximum number of friends to suggest: 3
Suggest friends key: deniz_2024-05-29 22:10:00
Suggested friends: Seyma Nur (Score: 0.5)
```

Suggesting friend to deniz: result is Şeyma Nur due to the common hobby: fishing.(in score information, i updated the code about displaying sharing the score between common hobbies and friends)

```
Please select an option: 3
Enter first person's name: deniz
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:10:00
Enter second person's name: Cagri
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:12:48
Add friendship keys: deniz_2024-05-29 22:10:00, Cagri_2024-05-29 22:12:48
Friendship added between deniz and Cagri
```

Friendship between Çağrı and Deniz

```
Please select an option: 7
Number of clusters found: 1
Cluster 1:
Cagri
Seyma Nur
deniz
Mert
```

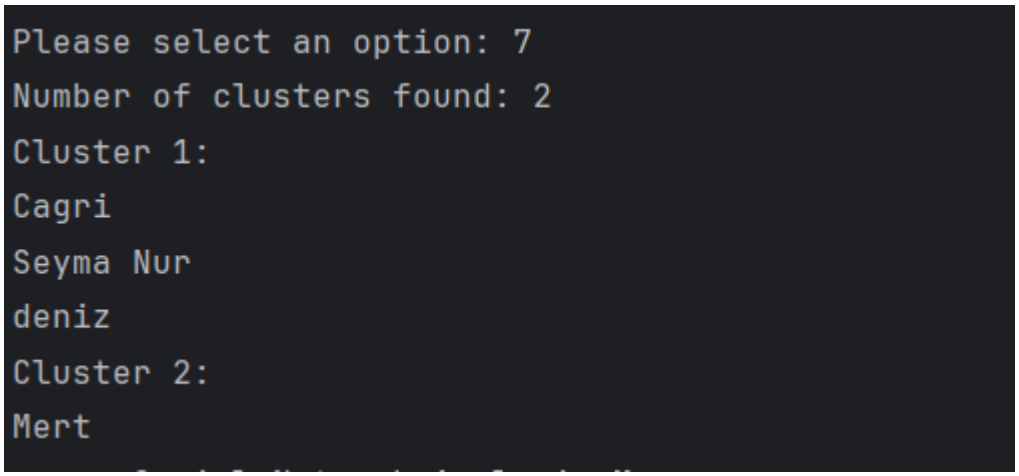
Number of clusters is downgraded in result of friendship between Deniz and Çağrı.

```
Please select an option: 5
Enter start person's name: Mert
Enter start person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:11:46
Enter end person's name: Seyma Nur
Enter end person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:14:12
Shortest path keys: Mert_2024-05-29 22:11:46, Seyma Nur_2024-05-29 22:14:12
Shortest path: Mert -> deniz -> Cagri -> Seyma Nur
```

Shortest path between Mert and Şeyma Nur

```
Please select an option: 4
Enter first person's name: deniz
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:10:00
Enter second person's name: Mert
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:11:46
Remove friendship keys: deniz_2024-05-29 22:10:00, Mert_2024-05-29 22:11:46
Friendship removed between deniz and Mert
```

Removing friendship between Mert and Deniz.



Final cluster appearance.

ANOTHER PROGRAM TO TEST UPDATED “SUGGEST FRIEND”

```
1
Enter name: a
Enter age: 10
Enter hobbies (comma-separated): 1,2,3
Person added: a (Timestamp: 2024-05-29 22:48:16)
===== 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: b
Enter age: 12
Enter hobbies (comma-separated): no
Person added: b (Timestamp: 2024-05-29 22:48:29)
===== 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: c
```

```
Please select an option: 1
Enter name: c
Enter age: 14
Enter hobbies (comma-separated): 2,3,4
Person added: c (Timestamp: 2024-05-29 22:48:35)
```

Persons a, b and c are added.

```
Please select an option: 3
Enter first person's name: a
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:48:16
Enter second person's name: b
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:48:29
Add friendship keys: a_2024-05-29 22:48:16, b_2024-05-29 22:48:29
Friendship added between a and b
===== 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 first person's name: b
Enter first person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:48:29
Enter second person's name: c
Enter second person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:48:35
Add friendship keys: b_2024-05-29 22:48:29, c_2024-05-29 22:48:35
Friendship added between b and c
```

Friendship betwenn a-b and b-c.

```
Please select an option: 6
Enter person's name: c
Enter person's timestamp (yyyy-MM-dd HH:mm:ss): 2024-05-29 22:48:35
Enter maximum number of friends to suggest: 5
Suggested friends: a (Score: 2.0, 1 mutual friends, 2 common hobbies)
```

Updated suggesting friend interface…(a hobbies: 1,2,3; c hobbies: 2,3,4)