2024

CSE222 – HW8 Report

ELIFNUR KABALCI
1801042617

Test - Outputs:

```
/* Adding 10 people */
network.addPerson(name:"John Doe", age:25, Arrays.asList(...a:"reading", "hiking", "cooking"));
network.addPerson(name:"Jane Smith", age:22, Arrays.asList(...a:"swimming", "cooking"));
network.addPerson(name:"Alice Johnson", age:27, Arrays.asList(...a:"hiking", "painting"));
network.addPerson(name:"Bob Brown", age:30, Arrays.asList(...a:"reading", "swimming"));
network.addPerson(name:"Emily Davis", age:28, Arrays.asList(...a:"running", "swimming"));
network.addPerson(name:"Frank Wilson", age:26, Arrays.asList(...a:"reading", "hiking"));
network.addPerson(name:"George White", age:23, Arrays.asList(...a:"gaming", "swimming"));
network.addPerson(name:"Hannah Green", age:29, Arrays.asList(...a:"reading", "cooking"));
network.addPerson(name:"Irene Black", age:24, Arrays.asList(...a:"hiking", "dancing"));
network.addPerson(name:"Jack Brown", age:31, Arrays.asList(...a:"running", "gaming"));
```

```
Person added: John Doe (Age: 25, Hobbies: [reading, hiking, cooking]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Jane Smith (Age: 22, Hobbies: [swimming, cooking]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Alice Johnson (Age: 27, Hobbies: [hiking, painting]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Bob Brown (Age: 30, Hobbies: [reading, swimming]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Emily Davis (Age: 28, Hobbies: [running, swimming]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Frank Wilson (Age: 26, Hobbies: [reading, hiking]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: George White (Age: 23, Hobbies: [gaming, swimming]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Hannah Green (Age: 29, Hobbies: [reading, cooking]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Irene Black (Age: 24, Hobbies: [hiking, dancing]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
Person added: Jack Brown (Age: 31, Hobbies: [running, gaming]) (Timestampt: Wed May 29 16:12:03 TRT 2024)
```

/**

```
System.out.println(x:"After adding 10 people");
network.displayPeople();
```

```
After adding 10 people



People in the network:
Emily Davis (Age: 28, Hobbies: [running, swimming])
Alice Johnson (Age: 27, Hobbies: [hiking, painting])
Irene Black (Age: 24, Hobbies: [hiking, dancing])
Hannah Green (Age: 29, Hobbies: [reading, cooking])
John Doe (Age: 25, Hobbies: [reading, hiking, cooking])
Bob Brown (Age: 30, Hobbies: [reading, swimming])
Frank Wilson (Age: 26, Hobbies: [reading, hiking])
George White (Age: 23, Hobbies: [gaming, swimming])
Jane Smith (Age: 22, Hobbies: [swimming, cooking])
Jack Brown (Age: 31, Hobbies: [running, gaming])
```

/**

```
/* Adding friendships */
network.addFriendship(name1:"John Doe", name2:"Jane Smith", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"Jane Smith").getTime());
network.addFriendship(name1:"John Doe", name2:"Alice Johnson", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"Alice Johnson").getTime());
network.addFriendship(name1:"Jane Smith", name2:"Bob Brown", network.people.get(key:"Jane Smith").getTime(),
        network.people.get(key:"Bob Brown").getTime());
network.addFriendship(name1:"Emily Davis", name2:"Frank Wilson", network.people.get(key:"Emily Davis").getTime(),
        network.people.get(key:"Frank Wilson").getTime());
network.addFriendship(name1:"George White", name2:"Hannah Green", network.people.get(key:"George White").getTime(),
        network.people.get(key:"Hannah Green").getTime());
network.addFriendship(name1:"Irene Black", name2:"Jack Brown", network.people.get(key:"Irene Black").getTime(),
        network.people.get(key:"Jack Brown").getTime());
network.addFriendship(name1:"John Doe", name2:"George White", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"George White").getTime());
network.addFriendship(name1:"Emily Davis", name2:"Hannah Green", network.people.get(key:"Emily Davis").getTime(),
        network.people.get(key:"Hannah Green").getTime());
```

```
Friendship added between John Doe and Jane Smith
Friendship added between John Doe and Alice Johnson
Friendship added between Jane Smith and Bob Brown
Friendship added between Emily Davis and Frank Wilson
Friendship added between George White and Hannah Green
Friendship added between Irene Black and Jack Brown
Friendship added between John Doe and George White
Friendship added between Emily Davis and Hannah Green
```

/**

```
System.out.println(x:"After adding friendships");
network.displayFriendships();
```

```
After adding friendships


Friendships in the network:
Alice Johnson is friends with [John Doe (Age: 25, Hobbies: [reading, hiking, cooking])]
Jack Brown is friends with [Irene Black (Age: 24, Hobbies: [hiking, dancing])]
Jane Smith is friends with [John Doe (Age: 25, Hobbies: [reading, hiking, cooking]), Bob Brown (Age: 30, Hobbies: [reading, swimming])]
George White is friends with [Hannah Green (Age: 29, Hobbies: [reading, cooking]), John Doe (Age: 25, Hobbies: [reading, hiking, cooking])]
Irene Black is friends with [Jack Brown (Age: 31, Hobbies: [running, gaming])]
Hannah Green is friends with [George White (Age: 23, Hobbies: [gaming, swimming]), Emily Davis (Age: 28, Hobbies: [running, swimming])]
Emily Davis is friends with [Frank Wilson (Age: 26, Hobbies: [reading, hiking]), Hannah Green (Age: 29, Hobbies: [reading, cooking])]
Bob Brown is friends with [Jane Smith (Age: 22, Hobbies: [swimming, cooking])]
Frank Wilson is friends with [Emily Davis (Age: 28, Hobbies: [running, swimming])]
John Doe is friends with [Jane Smith (Age: 22, Hobbies: [swimming, cooking]), Alice Johnson (Age: 27, Hobbies: [hiking, painting]), George White (Age: 23, Hobbies: [gaming, swimming])]
```

/**

```
/* Removing a person */
network.removeperson(name:"Jack Brown", network.people.get(key:"Jack Brown").getTime());

System.out.println(x:"After deleting Jack Brown");
network.displayPeople();
```

```
/* Removing a friendship */
network.removeFriendship(name1:"John Doe", name2:"Jane Smith", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"Jane Smith").getTime());

System.out.println(x:"After removing friendships - John Due - Jane Smith");
network.displayFriendships();
```

```
People in the network:
Emily Davis (Age: 28, Hobbies: [running, swimming])
Alice Johnson (Age: 27, Hobbies: [hiking, painting])
Irene Black (Age: 24, Hobbies: [hiking, dancing])
Hannah Green (Age: 29, Hobbies: [reading, cooking])
John Doe (Age: 25, Hobbies: [reading, hiking, cooking])
Bob Brown (Age: 30, Hobbies: [reading, swimming])
Frank Wilson (Age: 26, Hobbies: [reading, hiking])
George White (Age: 23, Hobbies: [gaming, swimming])
Jane Smith (Age: 22, Hobbies: [swimming, cooking])

Friendship deleted between John Doe and Jane Smith
After removing friendships - John Due - Jane Smith



Friendships in the network:
Alice Johnson is friends with [John Doe (Age: 25, Hobbies: [reading, hiking, cooking])]
Jane Smith is friends with [Bob Brown (Age: 30, Hobbies: [reading, swimming])]
George White is friends with [Hannah Green (Age: 29, Hobbies: [reading, cooking]), John Doe (Age: 25, Hobbies: [reading, hiking, cooking])]
Irene Black is friends with []
Hannah Green is friends with [George White (Age: 23, Hobbies: [gaming, swimming]), Emily Davis (Age: 28, Hobbies: [running, swimming])]
Emily Davis is friends with [Frank Wilson (Age: 26, Hobbies: [reading, hiking]), Hannah Green (Age: 29, Hobbies: [reading, cooking])]
Bob Brown is friends with [Jane Smith (Age: 22, Hobbies: [swimming, cooking])]
Frank Wilson is friends with [Emily Davis (Age: 28, Hobbies: [running, swimming])]
John Doe is friends with [Alice Johnson (Age: 27, Hobbies: [hiking, painting]), George White (Age: 23, Hobbies: [gaming, swimming])]
```

/**

```
/* Finding shortest path - Cannot find */
network.findShortestPath(startName:"John Doe", endName:"Bob Brown", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"Bob Brown").getTime());

/* Finding shortest path - can find */
network.findShortestPath(startName:"John Doe", endName:"Alice Johnson", network.people.get(key:"John Doe").getTime(),
        network.people.get(key:"Alice Johnson").getTime());
```

```
No path found between John Doe and Bob Brown
Shortest path: John Doe -> Alice Johnson
```

/**

```
/* Suggesting friends */
List<Person> l = network.suggestFriends(name:"John Doe", maxSuggestions:3, network.people.get(key:"John Doe").getTime());
System.out.println(x:"\nJohn Doe friend suggestions: ");
for (int i = 0; i < l.size(); i++) {
    System.out.println(l + "\n");
}
```

```
John Doe friend suggestions:
[Hannah Green (Age: 29, Hobbies: [reading, cooking])]
```

/**

```java
/* Counting clusters */
System.out.println("Number of clusters found: " + network.countClusters());
```

```
Cluster 1:
Emily Davis
Frank Wilson
Hannah Green
George White
John Doe
Alice Johnson

Cluster 2:
Irene Black

Cluster 3:
Bob Brown
Jane Smith

Number of clusters found: 3
```

Menu – Outputs:

```
' 'Main'
===== 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: elif
Enger age:
13
Enter hobbies (comma-separated):
kosma,oynama,dinleme
Person added: elif (Age: 13, Hobbies: [kosma, oynama,
 dinleme]) (Timestampt: Wed May 29 16:58:55 TRT 2024)
===== 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: nur
Enger age:
15
Enter hobbies (comma-separated):
kosma,dinleme,halay
Person added: nur (Age: 15, Hobbies: [kosma, dinleme,
 halay]) (Timestampt: Wed May 29 16:59:27 TRT 2024)
```

```
===== 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: elif
Enter first person's timestamp: Wed May 29 16:58:55 T
RT 2024
Enter second person's name: nur
Enter second person's timestamp: Wed May 29 16:59:27
TRT 2024
Friendship added between elif and nur
===== 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 first person's name: elif
Enter first person's timestamp: Wed May 29 16:58:55 T
RT 2024
Enter second person's name: nur
Enter second person's timestamp: Wed May 29 16:59:27
TRT 2024
Shortest path: elif -> nur
```

```
===== 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
Counting clusters int the social network
Cluster 1:
nur
elif

Number of clusters found: 1
===== 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:
8
Exiting..
```

Implementations:

```java
public class Person {
    String name;
    int age;
    List<String> hobbies;
    Date timestamp;
    String time;

    /* constructor */
    public Person(String name, int age, List<String> hobbies) {
        this.name = name;
        this.age = age;
        this.hobbies = new ArrayList<>(hobbies);
        this.timestamp = new Date();
        time = timestamp.toString();
    }
```

I added time data in the Person class. Since the timestamp received from the user as a string was a string, its conversion was difficult. However, string conversion was easier with toString. That's why I converted the automatically generated date data in the add people method to string and made the checks with it.

AddPerson:

```java
/* Method to add a person */
public void addPerson(String name, int age, List<String> hobbies) {
    /*
     * if we try to add exatly same people, they have different timestamp , so they
     * become not same
     */
    Person person = new Person(name, age, hobbies);
    people.put(name, person); /* add person list */
    friendships.put(person, new ArrayList<>()); /* add friendship key list */
    System.out.println("Person added: " + person + " (Timestampt: " + person.gettimestamp() + ")");
    /* print added person name and added time that we create in person class */

}
```

When we want to add someone, we add them as keys to both the people map and the friendship map. When we create the person, the addition time is automatically assigned. Person class I above was created with this logic.

RemovePerson:

```java
/* Method to remove a person */
public void removeperson(String name, String time) {
    Person person = people.get(name);
    people.keySet().remove(name); /* delete from people's */
    friendships.keySet().remove(person); /* delete from friendship's key */
    if (!person.getTime().equals(time)) {
        System.out.println(x:"Given time is not equal with eixst one.");
        return;
    }
    /* delete from friends to other's */
    for (Person p : friendships.keySet()) {
        if (friendships.get(p).contains(person)) {
            /* if is exist and given time and person's added time are equal */
            friendships.get(p).remove(person);
        }
    }
}
```

Since we are processing a previously added person here, we check whether the received username matches the received time data. Then we delete the person from the first person map. Then we delete it from the friendship key.

AddFriendship:

```java
/* Method to add a friendship */
public void addFriendship(String name1, String name2, String time1, String time2) {
    Person person1 = people.get(name1);
    Person person2 = people.get(name2);
    if (!time1.equals(person1.getTime()) || !time2.equals(person2.getTime())) {
        System.out.println(x:"Times are not equal with given by user.");
        return;
    }
    if (person1 != null && person2 != null) {
        /* if persons are exist and person's time are equal with given time values */
        friendships.get(person1).add(person2);
        friendships.get(person2).add(person1);
        /*
         * key:person1 value:person2
         * key:person2 value:person1
         */
        System.out.println("Friendship added between " + person1.name + " and " + person2.name);
    } else {
        System.out
                .println(x:"One or both persons not found in the network.");
    }
}
```

Name data is first searched in the people map. If not found, null is returned, if not null, the process begins. Before this, it is checked whether the person's creation time matches the time data received from the user. Later, the people were already added as keys to the friendship map in the add person section. Everyone adds friends at their own discretion. We describe friendship for both because it is reciprocal.

RemoveFriendship:

```java
/* Method to remove a friendship */
public void removeFriendship(String name1, String name2, String time1, String time2) {
    Person person1 = people.get(name1);
    Person person2 = people.get(name2);
    if (!time1.equals(person1.getTime()) || !time2.equals(person2.getTime())) {
        System.out.println(x:"Time's are not equal with exist ones.");
        return;
    }
    if (person1 != null && person2 != null) {
        friendships.get(person1).remove(person2);
        friendships.get(person2).remove(person1);
        System.out.println("Friendship deleted between " + person1.name + " and " + person2.name);
    } else {
        System.out.println(x:"One or both persons not found in the network.");
    }
}
```

Similar to the Add method, time and existence checks are made. Then, we give the key data for each person and delete the object from the relevant key's list.

SuggestFriends:

```java
public List<Person> suggestFriends(String name, int maxSuggestions, String time) {
    Person person = people.get(name);
    Map<Person, Double> scores = new HashMap<>();

    if (!person.getTime().equals(time)) {
        System.out.println(x:"Person's added time is wrong");
        return null;
    }
    for (Person friend : friendships.get(person)) {
        for (Person friendOfFriend : friendships.get(friend)) {
            if (!friendOfFriend.equals(person) && !friendships.get(person).contains(friendOfFriend)) {
                double score = calculateScore(person, friendOfFriend);
                scores.put(friendOfFriend, scores.getOrDefault(friendOfFriend, defaultValue:0.0) + score);
            }
        }
    }

    return scores.entrySet().stream()
            .sorted(Map.Entry.<Person, Double>comparingByValue().reversed())
            .limit(maxSuggestions)
            .map(Map.Entry::getKey)
            .collect(Collectors.toList());
}

private double calculateScore(Person person, Person candidate) {
    long mutualFriends = friendships.get(person).stream().filter(friendships.get(candidate)::contains).count();
    long commonHobbies = person.getHobbies().stream().filter(candidate.getHobbies()::contains).count();
    return mutualFriends * 1.0 + commonHobbies * 0.5;
}
```

First of all, time matching is checked, as in other methods. Inside the double for loop, it is checked whether two people are friends of each other and whether they have mutual friends. Having common hobbies and common friends among the people examined affects the score. A score is calculated according to the suitability of each person examined. The number of people you want to recommend is printed on the screen.

BFS- CountClusters:

```java
// Method to count clusters using BFS
public int countClusters() {
    Set<Person> visited = new HashSet<>();
    List<List<Person>> clusterList = new ArrayList<>();

    for (Person person : people.values()) {
        if (!visited.contains(person)) {
            List<Person> cluster = new ArrayList<>();
            bfs(person, visited, cluster);
            clusterList.add(cluster);
        }
    }
    for (int i = 0; i < clusterList.size(); i++) {
        System.out.println("Cluster " + (i + 1) + ":");
        for (Person person : clusterList.get(i)) {
            System.out.println(person.getname());
        }
        System.out.println();
    }
    return clusterList.size();
}

private void bfs(Person start, Set<Person> visited, List<Person> cluster) {
    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);
            }
        }
    }
}
```

In the bfs method, the general definition method of bfs was written. The count cluster method also uses this method. This method primarily contains the for loop of the application method of bfs. It checks whether the nodes have been visited or not and applies this order. While doing this, the clusters sent to the bfs method are also kept in a list. In the secondary part, these are pressed onto the screen.

DisplayPeople:

```
public void displayPeople() {
    /*
     * for printing people map
     */
    System.out.println(x:"\n\n\nPeople in the network:");
    for (Person person : people.values()) {
        System.out.println(person);
    }
    System.out.println();
}
```

While doing my tests, I wrote a method in which I printed the entire map on the screen to examine the operations of the add remove person methods.

DisplayFriendship:

```
public void displayFriendships() {
    /* for printing friendship map */
    System.out.println(x:"\n\n\nFriendships in the network:");
    for (Map.Entry<Person, List<Person>> entry : friendships.entrySet()) {
        System.out.println(entry.getKey().getname() + " is friends with " + entry.getValue());
    }
    System.out.println();
}
```

I wrote a method to control the add and remove friendship methods during tests.

FindShortestpath:

Since this method is long, I did not add a screenshot. Here, I made time and existence checks at the beginning. Then I used map, queue and set to keep the previous map contacts, to show the general contact list and those visited. A for while loop works by adding nodes to the list and adding others as you visit.

PrintPath:

We specify the starting and ending nodes and use the prev map to determine the path from end to end, and then we reverse this path. In this path I, we press it to the screen with a for loop.

Class Diagram: