Aufgabe 1: Superstar

Team-ID: 00381

Team-Name: EinSteine

Bearbeiter/-innen dieser Aufgabe:
Oliver Schirmer, Konrad Walter, Paskal Paesler, Corey Attireh

20. November 2018

Inhaltsverzeichnis

Lösungsidee

Aus der Aufgabenstellung gehen folgende zwei Bedingungen hervor, damit es sich bei einem Mitglied um den Superstar handelt:

1. Der vermeintliche Superstar darf keinem anderen Mitglied in der Gruppe folgen.
2. Jedes Mitglied der Gruppe muss dem vermeintlichen Superstar folgen.

Aus diesen beiden Bedingungen ergibt sich auch schon der Lösungsansatz. Es werden alle Mitglieder nacheinander durchgegangen und auf beide Bedingungen hin überprüft, angefangen bei der ersten Bedingung. Das ist insofern wichtig, da dadurch die Anzahl der benötigten Anfragen, durch ein „Fail-Fast“ System, stark minimiert werden kann. Wird ein Mitglied gefunden, dem das momentan getestete Mitglied folgt, wird direkt zum nächsten möglichen Superstar gesprungen. Selbiges gilt für die Überprüfung der 2. Bedingung.

Wichtig, um die benötigte Anzahl nochmals zu minimieren, ist ein Speicher, der bereits durchgeführte Anfragen hält und bei nochmaliger Anfrage das Ergebnis sofort liefert, ohne weitere Kosten zu verursachen.

Um die maximale nötige Anzahl an Anfragen zu bestimmen, muss man für die Verbindung jedes Mitglieds 2 Anfragen zu jedem anderen Mitglied mit einberechnen. Damit lässt sich folgende Formel bilden:

$$2*((n-1)*n/2)$$

Der Teil in Klammer stellt die Gaußsche Summenformel bezogen auf n-1 dar, wobei n der Anzahl der Mitglieder entspricht, wodurch die Anzahl der Verbindungen zwischen n Mitgliedern berechnet werden kann. Diese wird dann verdoppelt, da es jeweils eine Anfrage hin und zurück geben kann.

Umsetzung

Das grundsätzliche Überprüfen der Bedingungen erfolgt durch eine Verschachtlung von for-Schleifen. Dabei gibt es eine übergeordnete Schleife, die durch alle Mitglieder iteriert und eine weitere Schleife, die sich in ihr befindet und die beiden Bedingungen überprüft.

In der untergeordneten Schleifen wird geprüft, ob es sich um das Mitglied in der übergeordneten Schleife handelt, welches logischerweise übersprungen wird, da dieses für die Bedingungen nicht relevant ist.

Wird in der untergeordneten Schleife ermittelt, dass der vermeintliche Superstar jemandem folgt, wird in der übergeordneten Schleife zum nächsten Mitglied gesprungen und dieses überprüft. Selbiges passiert, wenn ermittelt wird, dass ein Mitglied dem vermeintlichen Superstar nicht folgt.

Beispiele

In diesem Abschnitt gehe ich auf die Originalbeispiele der Webseite (https://bwinf.de/bundeswettbewerb/37-bwinf/1-runde/material-371/) ein:

Beispiel 1:

An diesem relativ simplen Beispiel lässt sich das praktische Arbeiten des Algorithmus sehr gut verbildlichen. Die Mitglieder Selena, Justin und Hailey wurden in der gleichen Reihenfolge, wie in der Datei vorgegeben eingelesen. Daraus folgt, dass der Algorithmus bei der Überprüfung mit Selena beginnt. Zuerst muss dieser Ermitteln, ob Selena jemandem folgt, und da das Mitglied selbst übersprungen wird, ist die 1. Anfrage, ob Selena Justin folgt. Diese wird mit false, also nein beantwortet, weshalb der Algorithmus auch noch die Anfrage stellt, ob Selena Hailey folgt. Dies ist true, also wahr, weshalb die übergeordnete Schleife mit der Überprüfung Justin‘s fortführt. Dieser folgt zwar nicht Selena, jedoch Hailey. Somit ist auch er kein Superstar.
Bei der Überprüfung Hailey‘s stellt sich heraus, dass sie keinem der beiden anderen folgt, jedoch werden die beiden Anfragen zur Überprüfung, ob die beiden anderen Mitglieder Hailey folgen nicht gestellt. Das liegt an dem bereits angesprochenen Speicher (Cache), der die zuvor gestellten Anfragen (Selena→Hailey & Justin → Hailey), sofort mit ja beantwortet.
Damit ist Hailey nach 6 Anfragen als Superstar ermittelt worden.

Eingabe:	Ausgabe:
Selena Justin Hailey Selena Hailey Justin Hailey	---ALGORITHMUS--- Selena->Justin? false Selena->Hailey? true Justin->Selena? false Justin->Hailey? true Hailey->Selena? false Hailey->Justin? false ---ERGEBNIS--- User: 3 Superstar: Hailey Maximale Kosten: 6 Anfragen/Kosten: 6

Bei den folgenden Beispiel wurde auf eine detailliertere Erläuterung verzichtet, da diese nach dem selbigen Verfahren verläuft.

Beispiel 2:

```
---ERGEBNIS---
User: 5
Superstar: Dijkstra
Maximale Kosten: 20
Anfragen/Kosten: 10
```

Beispiel 3:

```
---ERGEBNIS---
User: 8
Superstar: /
Maximale Kosten: 56
```

Anfragen/Kosten: 8
Beispiel 4:

---ERGEBNIS---
User: 80
Superstar: Folke
Maximale Kosten: 6320
Anfragen/Kosten: 204

Quellcode

```
package de.ncrypted.superstar;

/**
 * @author ncrypted
 */
public class Superstar {

    private static List<User> users = new ArrayList();
    private static List<Connection> followCache = new
ArrayList<>();
    private static int costs = 0;

    public static void main(String[] args) {
        System.out.println("---INFOS---");
        // Lese die Daten aus der Datei ein
        readUsersData(userDataFileName);
        System.out.println("---ALGORITHMUS---");
        // Fuehre den Algorithmus aus
        User superstar = getSuperstar();
        System.out.println("---ERGEBNIS---");
        System.out.println("User: " + users.size());
        System.out.println("Superstar: " + (superstar == null ?
"/" : superstar.getName()));
        System.out.println("Maximale Kosten: " + (2 *
((users.size() - 1) * users.size() / 2)));
        System.out.println("Anfragen/Kosten: " + costs);
    }

    private static User getSuperstar() {
        first:
        for (User user : users) {
            // Pruefe ob der User jemandem folgt & ob alle dem
User folgen
            for (User other : users) {
                if (other == user) {
                    continue;
                }
                // Falls er jemandem folgt, kann er kein Superstar
sein -> Gehe zum nächsten User
                if (doesFollow(user, other)) {
                    continue first;
                }
                // Falls jemand ihm nicht folgt, kann er kein
Superstar sein -> Gehe zum nächsten User
                if (!doesFollow(other, user)) {
                    continue first;
                }
            }
            return user;
        }
```

```
            return null;
        }

        public static class User {
            private String name;
            private List<User> following = new ArrayList<>();

            public User(String name) {
                System.out.println("neuer User: " + name);
                this.name = name;
            }

            public String getName() {
                return name;
            }

            public void follow(User toFollow) {
                System.out.println(name + " folgt " +
    toFollow.getName());
                following.add(toFollow);
            }
        }

        private static class Connection {
            private User user1;
            private User user2;
            private boolean follows;

            public Connection(User user1, User user2, boolean follows)
    {
                this.user1 = user1;
                this.user2 = user2;
                this.follows = follows;
            }

            public User getUser1() {
                return user1;
            }

            public User getUser2() {
                return user2;
            }

            public boolean isFollowing() {
                return follows;
            }
        }

        private static boolean doesFollow(User user1, User user2) {
            // Pruefe ob die Abfrage bereits gemacht wurde, um Kosten
    zu vermeiden
            for (Connection connection : followCache) {
                if (connection.getUser1() == user1 &&
    connection.getUser2() == user2) {
                    return connection.isFollowing();
                }
            }

            costs += 1;
            boolean follows = user1.following.contains(user2);
```

```
        System.out.println(user1.getName() + "->" +
user2.getName() + "? " + follows);
        // Fuege die Anfrage dem Cache hinzu, um Kosten zu sparen
(theoretisch reichen positiven Anfragen, aufgrund des Algorithmus)
        followCache.add(new Connection(user1, user2, follows));
        return follows;
    }

}
```