Aufgabe 1: Superstar

Team-ID: 00361

Team-Name: bwinf

Bearbeiter dieser Aufgabe:
Yannik Schiebelhut

24. November 2018

Inhaltsverzeichnis

Lösungsidee

Vorsortierung

Zunächst wird eine Liste potentieller Superstars erstellt, die zu Beginn alle Namen enthält. Anschließend wird für jedes Mitglied im Netzwerk abgefragt, ob er/sie den anderen Mitgliedern folgt (Mitglied X folgt Mitglied Y?). Dabei ist jedoch zu beachten, dass diese Abfrage nur durchgeführt wird, wenn X ungleich Y ist und Y in der Liste der potentiellen Superstars enthalten ist. Gibt die Anfrage „true“ zurück, wird Mitglied X aus der Liste der potentiellen Superstars entfernt (ein Superstar darf niemandem folgen), gibt sie „false“ zurück, wird Mitglied Y aus der Liste entfernt (jeder muss einem Superstar folgen).

Finale Überprüfung

Nach der obigen Prozedur, enthält die Liste der potentiellen Superstars entweder noch ein Element oder sie ist leer. Falls sie noch ein Element enthält, ist es immer noch nicht sicher, ob dies auch ein Superstar ist. Deshalb muss noch überprüft werden, ob der potentielle Superstar wirklich niemandem im Netzwerk folgt. Dies ist notwendig, da bei der „Vorsortierung“ die Abfrage nur durchgeführt wurde, falls Mitglied Y in der Liste der potentiellen Superstars enthalten war. So ergibt sich zwar die Konstante Anzahl an Anfragen (Anzahl Mitglieder -1), die zusätzlich zur Vorsortierung bezahlt werden muss, allerdings ist dies immer noch deutlich billiger, als für jedes Mitglied direkt beim ersten Anlauf alle Folgemöglichkeiten zu überprüfen.

Umsetzung

Zunächst wird mithilfe von Scannern die Quelldatei, welche die Informationen über Mitglieder und Folgeeigenschaften enthält, auf drei ArrayListen aufgeteilt. Die erste enthält die vollständige Mitgliederliste. Liste zwei und drei hängen eng zusammen. Die Werte, die die beiden am jeweils identischen Index enthalten geben Informationen, welches Mitglied welchem anderen folgt (`ListeA.get(x) folgt ListeB.get(x)`).

Nun wird eine neue ArrayList mit dem selben Inhalt wie das Namensverzeichnis erstellt, welche die oben beschriebene Liste der potentiellen Superstars darstellt. Ebenfalls muss ein Zähler initialisiert werden, der die Anzahl der Anfrage festhält.

Jetzt folgt der eigentliche Vorgang. Mithilfe zweier ineinander verschachtelten for-Schleifen wird wie oben beschrieben die „Vorsortierung“ durchgeführt. Um trotz der vollständig vorliegenden Datenbank Anfragen wie durch die Aufgabenstellung gefordert durchführen zu können, erfolgt die Anfrage über eine eigens dafür konzipierte Methode. Diese nimmt die beiden Namen der zu überprüfenden Mitglieder (Mitglied X folgt Mitglied Y?) entgegen und prüft, ob es einen Index i gibt, bei dem in ListeA der Wert mit X und in ListeB der Wert mit Y übereinstimmt. Wird ein solcher gefunden, gibt die Methode true zurück, sonst false. In jedem Fall wird der Zähler für die Anfragen beim Methodenaufruf um 1 erhöht. Enthält die Liste der potentiellen Superstars vor Beginn einer Anfrage nur noch 1 Element, kann der Vorgang direkt abgebrochen und zum nächsten Schritt, der finalen Überprüfung übergegangen werden.

Wenn die ArrayList der potentiellen Superstars nicht leer ist, wird die obige Anfrage nach folgendem Muster erneut durchgeführt:
folgt(**verbleibendes Element**, Mitglied Y), wobei Y nacheinander die Werte aller anderen Mitglieder im Netzwerk annimmt. Gibt diese Anfrage niemals true zurück, ist der Superstar gefunden worden.

Beispiel (superstar2.txt)

Mitglieder: Turing Hoare Dijkstra Knuth Codd

	ListeA:	ListeB:
1.	Turing	Hoare
2.	Turing	Dijkstra
3.	Hoare	Turing
4.	Hoare	Djikstra
5.	Hoare	Codd
6.	Knuth	Turing
7.	Knuth	Dijkstra
8.	Codd	Turing
9.	Codd	Dijkstra
10.	Codd	Knuth

Potentielle Superstars: Turing Hoare Dijkstra Knuth Codd

Folgt Turing Hoare? // true

Potentielle Superstars: Hoare Dijkstra Knuth Codd

Folgt Turing Djikstra? // true

Folgt Turing Knuth? // false

Potentielle Superstars: Hoare Dijkstra Codd

Folgt Turing Codd? // flase

Potentielle Superstars: Hoare Dijkstra

Folgt Hoare Dijkstra? // true

Potentielle Superstars: Dijkstra

Dijkstra folgt Turing? // false

Dijkstra folgt Hoare? // false

Dijkstra folgt Knuth? // false

Dijkstra folgt Codd? // false

// Djikstra ist der Superstar

Quellcode

```
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;


public class A1 {
	// prepare counter for requests
	private static int sentFollowingRequests;


	/**
	 * The main method of the program. All essential work is done here.
	 *
	 * @param args database files given as command line arguments
	 */
	public static void main(String[] args) {
		// Verarbeitung der Komandozeilenargumente, siehe A1.java
		for (int i = 0; i < args.length; i++) {
			// no requests to pseudo network server yet
			sentFollowingRequests = 0;
			try { // needed in case a given file path is invalid
				// set source file to the parameter, currently being processed
```

```
File sourceFile = new File(args[i]);

// get the first line containing the members from sourceFile
Scanner sc = new Scanner(sourceFile);
sc.useDelimiter("\n");
String namesStr = sc.next();

// split string of network members and distribute them to an
ArrayList
List<String> names = new ArrayList<>();
Scanner nsc = new Scanner(namesStr);
nsc.useDelimiter(" ");
while (nsc.hasNext()) {
        names.add(nsc.next());
}
// process rest of the sourceFile and distribute data to two ArrayLists

// Format:
// users.get(x) follows targets.get(x)
List<String> users = new ArrayList<>();
List<String> targets = new ArrayList<>();
while (sc.hasNext()) {
        Scanner tmpSc = new Scanner(sc.next());
        tmpSc.useDelimiter(" ");
        users.add(tmpSc.next());
        targets.add(tmpSc.next());

        tmpSc.close();
}
// create a new ArrayList, containing all potential superstars
List<String> potSuperStar = new ArrayList<>(names);
// for now, all members are potential superstars
```

```
                // check for every user, if he follows one of the other users to find the
                // superstar in the group if he exists
                for (int userNum = 0; userNum < names.size(); userNum++) {
                    for (int targetNum = 0; targetNum < names.size();
targetNum++) {
                        // if there is only one potential superstar left, we
can directly check this one
                        if (potSuperStar.size() == 1)
                            break;
                        // to keep costs and requests low, check if requested
target is a potential
                        // superstar before sending a request
                        if (targetNum != userNum &&
potSuperStar.contains(names.get(targetNum))) {
                            // this is the actual request, whether user
X follows user Y
                            if
(checkFollowingStatus(names.get(userNum), names.get(targetNum), users, targets)) {
                                // if the request returns true, the
user X who follows user Y cannot be a
                                // superstar and can therefore be
removed from potential superstar list to
                                // reduce needed requests

        potSuperStar.remove(names.get(userNum));
                            } else {
                                // if the request returns true, user
Y cannot be followed by all users and
                                // therefore cannot be a superstar,
so he can be removed from the potential
                                // superstar list

        potSuperStar.remove(names.get(targetNum));
                            }
                        }
                    }
                }
```

```
                                        // make sure that superstar doesn't follow ANY member of the
network
                                        if (!potSuperStar.isEmpty()) {
                                                for (int targetNum = 0; targetNum < names.size();
targetNum++) {
                                                        if (!Objects.equals(names.get(targetNum),
potSuperStar.get(0))) {
                                                                if
(checkFollowingStatus(potSuperStar.get(0), names.get(targetNum), users, targets)) {
                                                                        potSuperStar.remove(0);
                                                                        break;
                                                                }
                                                        }
                                                }
                                        }

                                        System.out.println();

                                        // when there is a name left in the potential superstar list, it is the one
we
                                        // have been searching for
                                        if (!potSuperStar.isEmpty()) {
                                                System.out.printf("The superstar is %s.\t\t(%s)\n",
potSuperStar.get(0), sourceFile.getName());
                                        } else { // otherwise there is no superstar in the network
                                                System.out.printf("No superstar was found.\t\t(%s)\n",
sourceFile.getName());
                                        }
                                        System.out.printf("Needed %s requests.\n", sentFollowingRequests);
                                        System.out.println();
                                        System.out.println();

                                        // close resources
                                        sc.close();
```

```
                                        nsc.close();
                                } catch (FileNotFoundException e) { // needed if the given file path is invalid
                                        System.out.println("WARNING!!! Given file doesn't exists or
filepath is invalid!");
                                }
                        }
                }
        }

        /**
         * This method is used to sent the request to the pseudo network server. It
         * returns true if user follows target
         *
         * @param user    the name of the user you want to observe
         * @param target  the name of the user you want to know whether he's followed by
         *                the user from the first parameter
         * @param users   the database file, extracted in main, containing the list of
         *                users, who follow other users
         * @param targets the database file, extracted in main, containing the list of
         *                users, who are followed by other users
         * @return
         */
        public static boolean checkFollowingStatus(String user, String target, List<String> users, List<String>
targets) {
                // of course the has to be recorded as a request. Request cost money!!!
                sentFollowingRequests++;

                // print out information on the request to be sent
                System.out.printf("%s\t\tfollows\t\t%s?\t\t", user, target);

                // rush through the first list on search of the user
                for (int i = 0; i < users.size(); i++) {
                        // if the user has been found in the first list, check if the target is in the
```

```
                        // second list. at the same index as the user appears in the first list
                        if (Objects.equals(users.get(i), user) && Objects.equals(targets.get(i), target)) {
                                // the request returns true
                                System.out.println("true");
                                return true;
                        }
                }
                // the request returns false
                System.out.println("false");
                return false;
        }
}
```