Aufgabe 1: Superstar

Team: Pied Piper
Team-ID: 00342

23. November 2018

Inhalt

Lösungsidee

Die Aufgabe impliziert im Prinzip folgende zwei Bedingungen, die für ein Mitglied gelten muss, um als „Superstar“ gewertet zu werden:

1. Der potenzielle Superstar folgt keinem Mitglied der Gruppe (im folgenden Bedingung 1)
2. Der potenzielle Superstar, wird von jedem anderen Mitglied der Gruppe gefolgt (im folgenden Bedingung 2)

Diese zwei Bedingungen sind mit „Anfragen“ herauszufinden. Die Anfragenanzahl entspricht der benötigten Anzahl an Anfragen zum Erkennen eines Superstars. Eine Anfrage entspricht der Antwort, ob das Mitglied Y, dem Mitglied Z folgt. Grundlegend kann man einfach linear durch alle Namen durchgehen, und überprüfen ob die Bedingungen, für einen vorher bestimmten potenziellen Superstar, zutreffen. Findet man ein Mitglied, für das beide Bedingungen zustimmen, wird das Mitglied als Superstar deklariert und die Suche wird abgebrochen, da es nur ein Mitglied geben kann, für das beide Bedingungen gelten.

Laufzeitoptimierung

Es ist effizienter Bedingung 2 vor Bedingung 1 zu stellen, da diese Anfragen schon nächste potenzielle Superstars ausschließen könnten, wenn mitten in der aktuellen Überprüfung klar wird, dass das momentan überprüfte Mitglied kein Superstar mehr sein kann. In der anderen Anordnung der Reihenfolge der Abfrage, hat man keinen Sinn aus den Anfragen mehr bekommen, wenn mitten in der Überprüfung abgebrochen wird. Mit dieser Anordnung der Bedingungen kann man dann jedes Mitglied nehmen und die Bedingungen, wie am Anfang schon erwähnt, überprüfen. Um so viele verschiedene Mitglieder wie möglich nach dieser Bedingung abzufragen und damit, möglichst viele Mitglieder auf die Liste der nicht mehr relevanten Mitglieder zu setzen, wird jedes Mal, wenn ein Mitglied auf die Liste gesetzt wird, dieses ans Ende dieser Liste gesetzt.

Umsetzung

Die Lösungsidee wurde in Java implementiert. Mithilfe der java.nio Bibliothek wird die Datei mit den Mitgliedern in eine ArrayList gespeichert und die einzelnen Teilnehmernamen. Diese werden dann gespeichert in eine eigene ArrayList gespeichert und bilden somit den Grundbaustein des Vorgangs (im Programm macht das die splitFile Methode).

Um überhaupt die Bedingungen überprüfen zu können, müssen entsprechende Anfragen gestellt werden, sprich ob Mitglied A dem Mitglied B folgt. Dies wird dadurch bewerkstelligt, dass die zu überprüfenden Mitglieder als Paramater, sowie das Array das die Textdatei enthält, an eine Methode übergeben werde. Zu allererst wird in der Methode eine globale Variable um 1 erhöht, die am Ende anzeigt, wie viele Anfragen gestellt wurden. Anschließend geht die Methode dann mithilfe einer for-Schleife durch das Array und schaut in jeder Zeile ob die Zeile dem String: Mitglied A plus einem Leerzeichen plus Mitglied B gleicht. Falls durch die ganze Datei gegangen ist, und keine solche Zeile gefunden worden ist, wird *false* zurückgegeben, das Mitglied A folgt also nicht dem Mitglied B. Falls eine Zeile gefunden wurde, dass dem angegebenen String gleicht, wird *true* zurückgegeben, das Mitglied A folgt also dem Mitglied B.

Insgesamt sind 3 verschiedene for-Schleifen für das Finden des Superstars verbaut. Es sind außerdem zwei Listen mit allen Mitgliedern vorhanden, eine zum Auswählen des zu überprüfenden Mitglieds, die anderen zwei zum Überprüfen der Bedingungen. Die erste gibt vor, welches Mitglied überprüft werden soll, während die anderen beiden jeweils die Bedingungen für das in der ersten for-Schleife ausgewählte Mitglied überprüfen. Die Reihenfolge des Vorgangs ist somit von der ersten Liste abhängig.

Anfangs wählt die erste for-Schleife das erste Mitglied in der Liste aus.

Ist man nun in der zweiten for-Schleife angelangt wird zuerst die Bedingung 2 überprüft. Diese Schleife läuft solange bis entweder rauskommt, dass ein Mitglied nicht dem potenziellen Superstar folgt, also die Anfrage, ob das Mitglied dem potenziellen Superstar folgt, ist *false*, oder der Vorgang läuft bis zum Ende und die Variable die Aussagt, ob jedes Mitgli. Wird bekannt, dass ein Mitglied, dem potentiellen Superstar nicht folgt, wird der Vorgang vorzeitig abgebrochen, die Variable *getsFollowed*, die Aussagt ob X von allen Mitgliedern gefolgt wird, wird auf *false* gesetzt und der nächste potenzielle Superstar X wird von der ersten for-Schleife ausgewählt. Läuft der Vorgang erfolgreich bis zum Ende, ohne abzubrechen, sprich die Bedingung wurde mit jedem Mitglied und X überprüft, bleibt *getsFollowed* auf *true*. Außerdem wird nach jedem erfolgreichen Durchlauf dieser Schleife, das dem potenziellen Superstar folgende Mitglied auf einer Liste gespeichert, falls es nicht schon auf der Liste ist, um folgende Iterationen zu überspringen. Das Mitglied wird ebenfalls mithilfe der Methoden *list.add* und *list.remove* ans Ende der Liste gesetzt. Ist *getsFollowed* nach der Schleife *false* wird sofort der nächste potenzielle Superstar von der ersten for-Schleife gewählt. Sonst wird die nächste Bedingung, die Bedingung 1, mit der dritten for-Schleife überprüft.

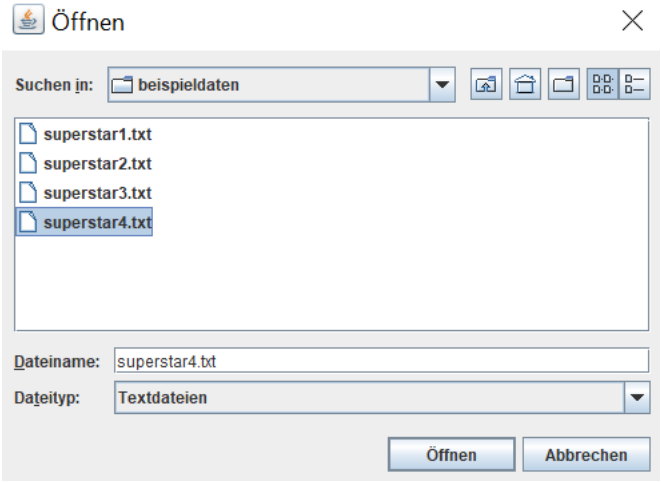
Diese Bedingung wird fast genauso wie in der ersten Schleife geprüft. Die Schleife läuft solange, bis erkannt wird, dass X einem Mitglied folgt, also bis die Anfrage *true* ist oder bis die Bedingung mit jedem Mitglied und X erfolgreich überprüft worden ist. Folgt X einem Mitglied, wird der Vorgang abgebrochen und die Variable, die aussagt, ob X keinem Mitglied folgt *doesntFollow* wird auf *false* gesetzt. Wurde erfolgreich überprüft, bleibt die Variable auf *true*.

Wurden die beiden Variablen bzw. Bedingungen nicht verändert, sind sie beide auf *true*. Somit müssen beide Bedingungen bei dem aktuellen potenziellen Superstar gelten. Die Suche wird erfolgreich abgebrochen, da es nur einen Superstar geben kann. Der Superstar wird dann an die main-Methode zurückgegeben, danach wird der Superstar sowie die benötigten Anfragen ausgegeben und das Programm wird beendet. Wurde die Suche erfolglos abgebrochen, werden die beiden Variablen wieder zurück auf *true* gesetzt und die Schleife beginnt wieder mit dem nächsten potenziellen Superstar.

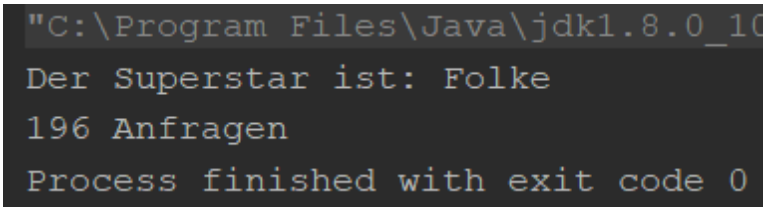
Wird am Anfang einer Schleife erkannt, dass der aktuelle potenzielle Superstar ein Mitglied der Liste ist, die vorher schon durch das Überprüfen der Bedingung 2 in der zweiten for-Schleife, ausgeschlossen wurden, wird die komplette Iteration übersprungen und der nächste potenzielle Superstar ausgewählt.

Beispiele

Das Programm wird aufgerufen und die zu überprüfende Datei wird ausgewählt.



Anschließend wird in der Konsole das Ergebnis präsentiert.



Die Ergebnisse der Beispiele:

1. Superstar ist Justin, mit 6 Anfragen herausgefunden
2. Superstar ist Dijkstra, mit 14 Anfragen herausgefunden
3. Superstar ist Niemand, mit 14 Anfragen herausgefunden
4. Siehe Bild

Quellcode

Die globale Variable und die main-Methode

```
private static int inquiriesAmount = 0;
//Um zu speichern, wie viele Anfragen benötigt wurden

public static void main(String args[]) throws IOException {
        System.out.print("Der Superstar ist: " + findsuperstar() + "\n" + inquiriesAmount + " Anfragen");
}
```

Aufgabe 1: Superstar

Hauptblock zum Finden des Superstars

```
for (int i = 0; allNames.size() - 1 >= i; i++) {

    if (searchInListFor(nonRelevantMembers,allNames.get(i))){
        continue;
    }

    doesntFollow = true;
    getsFollowed = true;

    for (int j = 0; allNames.size() - 1 >= j; j++) {
        //Das gleiche wie vorher, nur mit der nächsten Bedingung
        if (!allNamesSorted.get(j).contains(allNames.get(i)) && !anfrage(allNamesSorted.get(j), allNames.get(i), allData)) {
            //Falls eine Person nicht dem nun potentiellen Superstar folgt, wird abgebrochen
            // Die zweite Bedingung ist da um auszuschließen dass abgebrochen wird, wenn die Person nicht sich selber folgt
            getsFollowed = false;
            break;
        }
        if (!allNamesSorted.get(j).contains(allNames.get(i)) && !searchInListFor(nonRelevantMembers, allNamesSorted.get(j))){
            nonRelevantMembers.add(allNamesSorted.get(j));
            //Name zur Liste der nicht relevanten Mitglieder hinzufügen
            nameBuffer = allNamesSorted.get(j);
            allNamesSorted.remove(j);
            allNamesSorted.add(nameBuffer);
            //Ans Ende der Liste setzten

        }
    }

    if (getsFollowed) {
        //Falls die Bedingung 2 (getsFollowed) nicht vorher von der vorherigen for-schleife auf false gesetzt worden ist,
        //wird die Überprüfung fortgesetzt
        for (int j = 0; allNames.size() - 1 >= j; j++) {
            if (!allNamesSorted.get(j).contains(allNames.get(i)) && anfrage(allNames.get(i), allNamesSorted.get(j), allData)) {
                //Falls die Person irgendjemandem folgt, wird direkt abgebrochen und die nächste Person ausprobiert
                doesntFollow = false;
                break;
            }
        }
    } else continue;

    if (doesntFollow) {
        //Falls die Bedingungen 1 (doesntFollow) nicht vorher von der vorherigen for-Schleife auf false gesetzt worden ist,
        // wird der von der ersten for-schleife ausgewählte potenzielle Superstar zurückgegeben, da dieses Mitglied der Superstar
        //sein muss
        return allNames.get(i);
    }
}
return "Niemand";
```

Die Methode zum Anfragen stellen

```
private static boolean anfrage(String name1, String name2, List<String> allData) throws IOException { //Anfrage: ob X, Y folgt
    inquiriesAmount++;
    for (int i = 2; allData.size() - 1 >= i; i++) {
        //Durch alle Zeilen durchgehen und schauen ob X und Y, in dieser Reihenfolge, mit Leerzeichen dazwischen, stehen
        if ((allData.get(i)).contains(name1 + " " + name2)) {
            return true;
        }
    }
    return false;

}
```

Aufgabe 1: Superstar

Die Methode zum Entnehmen aller Mitglieder

```
private static List<String> splitFile(String filepath) throws IOException {
    //String in Array aufteilen -> jeder Name hat eine Zeile
    List<String> lines = Files.readAllLines(Paths.get(filepath), StandardCharsets.ISO_8859_1);
    int listLocation = 0;
    String names = lines.get(0);

    List<StringBuilder> allNames = new ArrayList<>();
    allNames.add(new StringBuilder());
    for (int i = 0; names.length() - 1 >= i; i++) {
        if (names.charAt(i) == ' ') {
            //Bei Leerzeichen in nächste Zeile
            listLocation++;
            allNames.add(new StringBuilder());
        } else {
            allNames.set(listLocation, allNames.get(listLocation).append(names.charAt(i)));
            //Sonst wird der aktuelle Buchstabe der aktuellen Stelle angehängt
        }
    }
    return toStringList(allNames);
}
```

Die Methode zum Auswählen der Datei

```
private static String chooseFile() {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter( description: "Textdateien", ...extensions: "txt");
    //Nur .txt Dateien werden anezeigt
    chooser.setFileFilter(filter);
    int chosen = chooser.showOpenDialog( parent: null);

    if (chosen == JFileChooser.APPROVE_OPTION) {
        //Wenn Ok gedrückt wird, wird der Pfad der Datei zurückgegeben
        return chooser.getSelectedFile().getPath();
    } else System.exit( status: 0);
    //Sonst wird beendet
    return null;
}
```