Aufgabe 1: Superstar

Team-ID: 00772

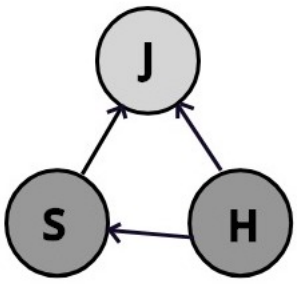
Team-Name: T.S

26. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Da TeeniGram ein soziales Netzwerk ist, liegt es nahe, die Daten als **gerichteten Graphen** G zu betrachten. Dabei werden die n Mitglieder einer Gruppe als n Knoten dargestellt (durchnummeriert mit 1,2,3,...,n nach Eingabereihenfolge) und eine gerichtete Kante von Mitglied X zu Mitglied Y bedeutet, dass Mitglied X Mitglied Y folgt. Das Beispiel aus der Aufgabe würde folgendermaßen aussehen werden:



Man sieht sofort, dass ein Superstar (bezeichnet mit der Nummer s) dadurch erkannt werden kann, dass sein Eingangsgrad $d_G^-(s) = n-1$ beträgt und sein Ausgangsgrad $d_G^+(s) = 0$. Wird der Graph als **Matrix** G dargestellt (bei der $G[x][y] = 1$ bedeutet, dass X Y folgt und $G[x][y] = 0$, dass X Y nicht folgt), enthält in dem Fall also die s-te Zeile nur Nullen und die s-te Spalte nur aus Einsen (abgesehen vom Eintrag $G[s][s] = 0$, da sich eine Person nicht selber folgen kann).

So würde die Matrix von Beispiel 1 aussehen, wobei Selena Nummer 0, Justin Nummer 1, und Hailey Nummer 2 ist:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Würde man um den Superstar zu finden für jedes einzelne Paar von Mitgliedern die Fragen „Folgt Mitglied X Mitglied Y?“ und „Folgt Mitglied Y Mitglied X?“ stellen (d.h die Positionen $G[x][y]$ und $G[y][x]$ der Matrix abfragen), müsste man mit dieser Brute-Force-Methode $n^2 - n$ (n^2 Einträge in der Matrix insgesamt ohne n Einträge $G[x][x]$) Fragen stellen, was sehr kostenaufwändig wäre.

Deswegen macht an sich folgende Eigenschaften zu Nutze:

- falls $G[x][y] = 1$ dann X $\neq$ Superstar (weil X jemandem folgt) (1)
- falls $G[x][y] = 0$ dann Y $\neq$ Superstar (weil Y von jemandem nicht gefolgt wird) (2)

Dadurch kommt folgendes **Verfahren** zustande (mit einer Matrix zur Darstellung der Daten):

1. Man fängt in der obersten Zeile der Matrix links (theoretisch $G[0][0]$ aber da eine Person sich nie selber folgt, praktisch bei $G[0][1]$) an, abzufragen ob X Y folgt
   a) Ist die Antwort ja ($G[x][y] = 1$), dann geht man eine Zeile nach unten ($x + 1$)
   b) Ist die Antwort ja ($G[x][y] = 0$), dann geht man eine Spalte nach rechts ($y + 1$)
2. Erreicht man an einem Punkt $x > n$, gibt es keinen Superstar und das Programm wird beendet, erreicht man $y > n$, kann nur noch das Mitglied mit dem aktuellen Wert von x (folgend x_s benannt) als Nummer ein Superstar sein
3. Man kontrolliert ob die x_s-te Zeile nur Nullen und die x_s-te Spalte nur Einsen enthält, wenn ja ist das Mitglied mit der Nummer x_s der Superstar und das Programm wird beendet, sonst gibt es keinen Superstar und das Programm wird beendet

Warum kann in diesem Verfahren im Punkt 2. nur x_s der Superstar sein? Es kann zum einem kein weiterer Knoten k mit $k < x_s$ sein aufgrund von Eigenschaft (1), da alle Zeilen vor Zeile x_s mindestens eine 1 hatten, die der Grund für den Zeilenwechsel $x + 1$ war (und eine 1 in der Zeile k bedeutet, dass des Mitglied mit der Nummer k kein Superstar ist). Es kann aber auch kein weiterer Knoten k mit $k > x_s$ sein aufgrund von Eigenschaft (2), da alle Spalten vor der letzten Spalte $n - 1$ mindestens eine 0 hatten, die der Grund für den Spaltenwechsel $y + 1$ war (und eine 0 in der Spalte k die auch nicht in der Zeile k ist (weil $k > x_s$) bedeutet, dass des Mitglied mit der Nummer k kein Superstar ist.

Die maximale Anzahl von Abfragen lässt sich folgendermaßen berechnen:

- Phase 1 (Finden eines potentiellen x_s): Im worst case ist entweder $x_s = n - 1$ oder es gibt keinen Superstar und die Einsen und Nullen sind in der Matrix so verteilt, dass man bis vor $G[n - 1][n - 1]$ abfragen muss. In beiden Fällen muss der von $G[0][0]$ am weitesten entfernte Eintrag mit $2(n-1) = 2n-2$ Abfragen erreicht werden ($n-1$ statt n weil $G[k][k]$ nie abgefragt werden muss)

- Phase 2 (Kontrollieren des potentiellen x_s): Immer $2(n-1) = 2n-2$ Abfragen, da eine Spalte und eine Zeile (ohne $G[x_s][x_s]$) kontrolliert wird

Insgesamt führt das bei n Mitgliedern der Gruppe also zu einer **maximalen Anzahl** von $4n-4$ Abfragen. Diese Methode ist also um eigenes kostengünstiger als die Brute Force Methode mit n^2-n Abfragen, da für $n > 4 : n^2-n > 4n-4$ gilt (algorithmisch gesehen kann man auch sagen, dass $O(n^2)$ schneller steigt als $O(n)$). Man sollte auch beachten, dass die Brute-Force-Methode immer genau so viele Abfragen braucht, während es bei dem Verfahren bei $4n-4$ nur um die maximale Anzahl handelt und man oft noch weniger Fragen stellen muss.

Es wäre auch möglich die Anzahl der Fragen noch weiter zu verringern, indem man die Antworten zu allen Fragen speichert, sodass man Fragen eventuell nicht doppelt stellen muss weil man zunächst die Antworten durchsucht bevor man fragt, dies würde jedoch einen höheren Speicher- und Zeitaufwand bedeuten.

2 Umsetzung

Die Lösungsidee wurde in Java implementiert.

Zunächst werden die Namen aller Teilnehmer aus der Eingabedatei (im Code veränderbar) eingelesen und dabei wird (abgesehen von einer allgemeinen Liste aller Teilnehmernamen) zum einem eine HashMap, die jedem Teilnehmernamen eine Nummer zuordnet, zum anderen wird eine Adjazenzmatrix des Graphen als zweidimensionaler Array erstellt. Anschließend werden die Folgebeziehungen eingelesen und in die Matrix eingetragen. Bei der darauf folgenden Ausführung des Verfahrens wird jede Frage (mit der Antwort) in der Konsole ausgegeben, und am Ende der Superstar falls es einen gibt. Zusätzlich kann auch die Matrix und die Anzahl der gestellten Fragen ausgegeben werden.

3 Beispiele

Beispiel 1 (superstar1.txt):

```
Frage 1: Folgt Selena Justin? ja
Frage 2: Folgt Justin Hailey? nein
Frage 3: Folgt Selena Justin? ja
Frage 4: Folgt Hailey Justin? ja
Frage 5: Folgt Justin Selena? nein
Frage 6: Folgt Justin Hailey? nein
Der Superstar ist Justin
Gefunden in 6 Fragen (maximale Anzahl waere 8)
```

Beispiel 2 (superstar2.txt):

```
Frage 1: Folgt Turing Hoare? ja
Frage 2: Folgt Hoare Dijkstra? ja
Frage 3: Folgt Dijkstra Knuth? nein
Frage 4: Folgt Dijkstra Codd? nein
```

```
Frage 5: Folgt Turing Dijkstra? ja
Frage 6: Folgt Hoare Dijkstra? ja
Frage 7: Folgt Knuth Dijkstra? ja
Frage 8: Folgt Codd Dijkstra? ja
Frage 9: Folgt Dijkstra Turing? nein
Frage 10: Folgt Dijkstra Hoare? nein
Frage 11: Folgt Dijkstra Knuth? nein
Frage 12: Folgt Dijkstra Codd? nein
Der Superstar ist Dijkstra
Gefunden in 12 Fragen (maximale Anzahl waere 16)
```

Beispiel 3 (superstar3.txt):

```
Frage 1: Folgt Edsger Jitse? ja
Frage 2: Folgt Jitse Jorrit? nein
Frage 3: Folgt Jitse Peter? nein
Frage 4: Folgt Jitse Pia? nein
Frage 5: Folgt Jitse Rineke? nein
Frage 6: Folgt Jitse Rinus? nein
Frage 7: Folgt Jitse Sjoukje? nein
Frage 8: Folgt Edsger Jitse? ja
Frage 9: Folgt Jorrit Jitse? nein
Frage 10: Folgt Peter Jitse? nein
Frage 11: Folgt Pia Jitse? nein
Frage 12: Folgt Rineke Jitse? nein
Frage 13: Folgt Rinus Jitse? nein
Frage 14: Folgt Sjoukje Jitse? nein
Es gibt keinen Superstar in dieser Gruppe
Gefunden in 14 Fragen (maximale Anzahl waere 28)
```

Beispiel 4 (superstar4.txt):
(Bei diesem Beispiel würden die einzelnen Fragen hier zu viel Platz einnehmen)

```
...
Der Superstar ist Folke
Gefunden in 256 Fragen (maximale Anzahl waere 316)
```

4 Quellcode

```
public class Superstar {

    static int n; //Gruppengroesse
    static HashMap<String, Integer> mitglieder; //ordnet Teilnehmern Nummern zu
    static String[] names; //Teilnehmernamen
    static int[][] matrix; //Adjazenzmatrix
    static int fragen; //Anzahl der benoetigten Fragen

    public static void main(String[] args) throws IOException {
        buildMatrix("superstar4.txt"); //hier filename veraendern
        findSuperstar();
```

```
    }

    //fuehrt Verfahren aus
    public static void findSuperstar(){
        fragen = 0;
        int x = 0;
        int y = 0;
        while(x<n && y<n){
            if(x != y){
                if(XfollowsY(x, y)){
                    x++;
                }else{
                    y++;
                }
            }else{
                y++;
            }
        }
        if(x == n-1){
            System.out.println("Es gibt keinen Superstar in dieser Gruppe");
        }else{
            if(checkSuperstar(x)){
                System.out.println("Der Superstar ist "+names[x]);
            }else{
                System.out.println("Es gibt keinen Superstar in dieser Gruppe");
            }
        }
        System.out.println("Gefunden in "+fragen+" Fragen (maximale Anzahl waere"
        +(4*n-4)+")");
    }

    //Kontrolliert ob gefundener moeglicher Superstar wirklich Superstar ist
    public static boolean checkSuperstar(int x){
        boolean s = true;
        for(int i = 0; i<n; i++){
            if(i != x){
                s = XfollowsY(i, x);
            }
        }
        if(!s){
            return s;
        }else{
            for(int i = 0; i<n; i++){
                if(i != x){
                    if(XfollowsY(x, i)){
                        s = false;
                    }
                }
            }
            return s;
        }
    }

    //fraegt Position G[x][y] ab und gibt die dazugehoerige Frage aus
    public static boolean XfollowsY(int x, int y){
```

```
67          fragen++;
            System.out.print("Frage␣"+fragen+":␣Folgt␣"+names[x]+"␣"+names[y]+"?␣");
69          boolean antwort = (matrix[x][y]==1);
            if(antwort){
71              System.out.println("ja");
            }else{
73              System.out.println("nein");
            }
75          return antwort;
        }
77
        //erstellt Adjazenzmatrix (und weitere Listen) aus Eingabe
79      public static void buildMatrix(String fn) throws IOException {
            mitglieder = new HashMap<>();
81          String filename = fn;
            String currentDirectory;
83          File file = new File(filename);
            currentDirectory = file.getAbsolutePath();
85          final BufferedReader in = new BufferedReader(new FileReader(currentDirectory));
            String name = "";
87          String line = in.readLine();
            names = line.split("␣");
89          n = names.length;
            for(int i = 0; i<names.length; i++){
91              mitglieder.put(names[i], i);
            }
93          String[] connection;
            matrix = new int[n][n];
95          while((line = in.readLine()) != null){
                connection = line.split("␣");
97              matrix[mitglieder.get(connection[0])][mitglieder.get(connection[1])] = 1;
            }
99      }
    }
```