Aufgabe 1
„Superstar“- Dokumentation

37. Bundeswettbewerb Informatik 2018/19 - 1. Runde

Sebastian Baron, Simon Fiebich, Lukas Rost

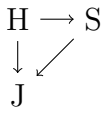
Team-ID: 00036

26. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Zunächst bietet es sich an, die Eingabe in einen Graphen umzuwandeln. Die Knoten entsprechen dabei den TeeniGram-Mitgliedern. Eine gerichtete Kante verläuft von einem Knoten x zu einem Knoten y genau dann, wenn x y im TeeniGram-Netzwerk folgt. Für das in der Aufgabenstellung gegebene Beispiel ergibt sich folgender Graph (Namen abgekürzt):



Mithilfe dieses Graphen kann nun ein Superstar gefunden werden. Dafür verwendet man eine modifizierte Version der Tiefensuche. Dazu wählt man einen beliebigen Knoten (z.B. den ersten) als Startpunkt *start* aus. Man setzt außerdem die Existenz einer Funktion $hasEdge(x, y)$ voraus, die angibt, ob es eine an x beginnende und an y endende Kante gibt. Diese Funktion stellt die einzige Zugriffsmöglichkeit auf die Kanten dar, während die Knoten vorher bekannt sind. Außerdem muss eine Liste *visited* vorhanden sein, die die schon besuchten Knoten angibt und am Anfang leer ist.

Dann ruft man folgenden Algorithmus mit den Parametern *start* und `null` auf:

```
function modifiedDFS(start,parent)
    Füge start zu visited hinzu
    next ← erster noch nicht besuchter Knoten, für den hasEdge(start,next) gilt
    if next existiert then
        return modifiedDFS(next,start)      ▷ rekursiv zum nächsten Knoten absteigen
    else                                    ▷ start möglicherweise Superstar
        for all Knoten u in visited do      ▷ keine Kanten zu besuchten Knoten existieren
            if hasEdge(start,u) then
                return kein Superstar
            end if
        end for
        for all Knoten k (ohne start und parent) do      ▷ alle k haben Kante zu start
            if not hasEdge(k,start) then
                return kein Superstar
            end if
        end for
        return start                        ▷ start ist Superstar
    end if
end function
```

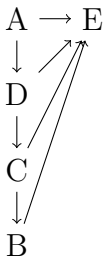
Dieser Algorithmus steigt sozusagen immer weiter über die erste ausgehende Kante des aktuellen Knotens zum nächsten Knoten hinab. Dabei vermeidet er Kanten, die zu schon besuchten Knoten führen. Hat der aktuelle Knoten keine ausgehenden Kanten (unabhängig davon, ob diese zu schon besuchten Knoten führen), könnte dieser Knoten (und nur dieser Knoten) ein Superstar sein.

Andere Knoten kommen dafür nicht in Frage, da ihnen dann auch der aktuelle Knoten folgen müsste. Dies tut er jedoch nicht, da er keine ausgehenden Kanten hat. Nun muss man den aktuellen Knoten nur noch darauf überprüfen, ob alle anderen Knoten ihm folgen (bzw. er von allen Knoten eine eingehende Kante besitzt). Ist auch dies der Fall, handelt es sich beim aktuellen Knoten tatsächlich um einen *Superstar*.

1.1 Anzahl der Anfragen

Die Anzahl der gestellten Anfragen kann als ungefähr äquivalent bzw. proportional zur Laufzeit betrachtet werden. Somit handelt es sich hier zugleich um eine Laufzeitbetrachtung.

Zunächst betrachten wir die Phase bis zum Auffinden des Superstars, d.h. bis $modifiedDFS()$ als Parameter *start* den Superstar enthält. Da die maximal notwendige Anfragenzahl gesucht ist, ist es sinnvoll, einen Worst-Case-Fall aufzubauen. Für ein Beispiel aus 5 Knoten könnte dieser so aussehen:



Gehen wir nun davon aus, dass die Knoten in der Reihenfolge A, B, C, D, E geordnet sind, ergibt sich der Worst Case für die Anfragenanzahl. Dabei werden folgende Anfragen gestellt:

1. $A->B$
2. $A->C$
3. $A->D$
4. $D->B$
5. $D->C$
6. $C->B$
7. $B->E$

Ein ähnlicher Worst Case lässt sich auch für jede beliebige andere Knotenanzahl erstellen. Allgemein werden dabei $(n-2)+(n-3)+(n-4)+...+2+1+1$ Anfragen gestellt.[1] Insgesamt ergibt dies:

$$1+\sum_{i=2}^{n-1} n-i = \frac{n^2-3\cdot n+4}{2}$$

Die Anfragen für die anderen Bestandteile der Suche lassen sich leichter berechnen. Die Überprüfung, dass der mögliche Superstar keine ausgehenden Kanten hat, benötigt insgesamt $n-1$ Anfragen, denn dass der Superstar keine Kante zu sich selbst hat, ist offensichtlich. Dass alle anderen dem Superstar folgen, ist in $n-2$ Anfragen sichergestellt, denn der *parent* des Superstars hat offensichtlich eine Kante zu ihm. Es ergibt sich also insgesamt folgende Maximalanzahl an Anfragen:

$$\frac{n^2-3\cdot n+4}{2}+(n-1)+(n-2)=\frac{n^2+n-2}{2}$$

Dies ist jedoch wirklich nur eine Maximalanzahl, die in den meisten Fällen erheblich unterschritten wird. Nimmt man die Anfragenanzahl als Grundlage für die Laufzeitbetrachtung, so liegt der Algorithmus in $\mathcal{O}(n^2)$.

[1] Vom ersten Knoten aus $n-2$ usw.

2 Umsetzung

Das Programm wurde in Java geschrieben und mit Java 10 kompiliert. Die im Unterordner `out/artifacts` bereitstehende .jar-Datei lässt sich mit einer Java-Version größer oder gleich 10 ausführen. Bei Neukompilierung läuft das Programm aber auch ab Java 8. Bei der jar-Datei handelt es sich um ein Konsolenprogramm.

Das Programm besteht aus vier Klassen.

- `Main`. Diese Klasse ist die Hauptklasse, die alle anderen Klassen aufruft.
- `SuperstarHelper`. Diese Klasse übernimmt Eingabe und Ausgabe.
- `Graph`. Diese Klasse implementiert den abstrakten Datentyp Graph und beherbergt den eigentlichen Algorithmus.
- `Vertex`. Diese Klasse repräsentiert einen Knoten mitsamt Inhalt und Adjazenzliste.

Zunächst wird beim Aufruf des Programms die `main`-Methode der Klasse `Main` aufgerufen. Diese erzeugt einen `SuperstarHelper` und führt dessen Methoden in der richtigen Reihenfolge aus.

Der `SuperstarHelper` erfragt zunächst mithilfe eines grafischen `JFileChooser`-Dialogs die Eingabedatei. Anschließend wird diese durch einen `BufferedReader` eingelesen. Zunächst wird eine Liste der Knoten erstellt. Gleichzeitig wird eine `HashMap` erstellt, die beim Hinzufügen der Kanten eine einfache Zuordnung von Name zu Knoten ermöglicht. Aus den Knoten wird nun ein `Graph`-Objekt erzeugt. Anschließend wird dessen `modifiedDFS()`-Methode aufgerufen und auf Grundlage dieses Algorithmus eine Ausgabe generiert.

Die `Graph`-Klasse repräsentiert einen Graphen und behrebegt gleichzeitig den eigentlichen Algorithmus. Sie enthält eine Knotenliste, eine Liste der besuchten Knoten, einen Verweis auf den Startknoten sowie einen Anfragezähler. Wichtig ist hier die Methode `hasEdge()`, die eine Anfrage durchführt und dabei den Anfragezähler inkrementiert sowie die Anfrage ausgibt. Die Methode `modifiedDFS()` enthält den Hauptalgorithmus, wie er unter „Lösungsidee “ vorgestellt wurde.

Schließlich gibt es noch die `Vertex`-Klasse, die einen Knoten darstellt. Sie enthält eine Adjazenzliste sowie Funktionen zum Hinzufügen zu dieser und Auslesen dieser. Außerdem ist ein Feld für den Inhalt, in diesem Fall also den Namen des Mitglieds, vorhanden. Die `equals()`-Methode wurde ebenfalls implementiert, da der Algorithmus sie häufig benutzt.

3 Beispiele

Ausgabe für Beispiel 1

```
1  [Anfrage] Folgt Selena Justin? Antwort: Ja
2  [Anfrage] Folgt Justin Hailey? Antwort: Nein
3  [Anfrage] Folgt Justin Selena? Antwort: Nein
4  [Anfrage] Folgt Hailey Justin? Antwort: Ja
5  Superstar ist Justin.
6  Anzahl der Anfragen: 4
```

Ausgabe für Beispiel 2

```
[Anfrage] Folgt Turing Hoare? Antwort: Ja
[Anfrage] Folgt Hoare Dijkstra? Antwort: Ja
[Anfrage] Folgt Dijkstra Knuth? Antwort: Nein
[Anfrage] Folgt Dijkstra Codd? Antwort: Nein
[Anfrage] Folgt Dijkstra Turing? Antwort: Nein
[Anfrage] Folgt Dijkstra Hoare? Antwort: Nein
[Anfrage] Folgt Turing Dijkstra? Antwort: Ja
[Anfrage] Folgt Knuth Dijkstra? Antwort: Ja
[Anfrage] Folgt Codd Dijkstra? Antwort: Ja
Superstar ist Dijkstra.
Anzahl der Anfragen: 9
```

Ausgabe für Beispiel 3

```
[Anfrage] Folgt Edsger Jitse? Antwort: Ja
[Anfrage] Folgt Jitse Jorrit? Antwort: Nein
[Anfrage] Folgt Jitse Peter? Antwort: Nein
[Anfrage] Folgt Jitse Pia? Antwort: Nein
[Anfrage] Folgt Jitse Rineke? Antwort: Nein
[Anfrage] Folgt Jitse Rinus? Antwort: Nein
[Anfrage] Folgt Jitse Sjoukje? Antwort: Nein
[Anfrage] Folgt Jitse Edsger? Antwort: Ja
Es gibt keinen Superstar.
Anzahl der Anfragen: 8
```

Ausgabe für Beispiel 4 (gekürzt)

```
[Anfrage] Folgt Hanna Melker? Antwort: Nein
[Anfrage] Folgt Hanna Liv? Antwort: Nein
[Anfrage] Folgt Hanna Ellen? Antwort: Ja
[Anfrage] Folgt Ellen Melker? Antwort: Ja
[Anfrage] Folgt Melker Liv? Antwort: Nein
[Anfrage] Folgt Melker Ali? Antwort: Ja
[Anfrage] Folgt Ali Liv? Antwort: Nein
[Anfrage] Folgt Ali Lova? Antwort: Ja
[Anfrage] Folgt Lova Liv? Antwort: Ja
[Anfrage] Folgt Liv Vide? Antwort: Nein
[Anfrage] Folgt Liv Freja? Antwort: Nein
[Anfrage] Folgt Liv Melvin? Antwort: Nein
[Anfrage] Folgt Liv Loke? Antwort: Nein
[Anfrage] Folgt Liv Sigge? Antwort: Nein
[Anfrage] Folgt Liv Milton? Antwort: Nein
[Anfrage] Folgt Liv Sofia? Antwort: Ja
[...]
[Anfrage] Folgt August Folke? Antwort: Ja
[Anfrage] Folgt Folke Tuva? Antwort: Nein
[Anfrage] Folgt Folke Isabelle? Antwort: Nein
[Anfrage] Folgt Folke Wilmer? Antwort: Nein
[...]
[Anfrage] Folgt Folke Jonathan? Antwort: Nein
[Anfrage] Folgt Folke Colin? Antwort: Nein
[Anfrage] Folgt Folke Frank? Antwort: Nein
[Anfrage] Folgt Folke August? Antwort: Nein
[Anfrage] Folgt Hanna Folke? Antwort: Ja
[Anfrage] Folgt Melker Folke? Antwort: Ja
[Anfrage] Folgt Liv Folke? Antwort: Ja
[Anfrage] Folgt Ellen Folke? Antwort: Ja
[Anfrage] Folgt Ali Folke? Antwort: Ja
```

```
[...]
[Anfrage] Folgt Sebastian Folke? Antwort: Ja
[Anfrage] Folgt Charlie Folke? Antwort: Ja
[Anfrage] Folgt Penny Folke? Antwort: Ja
[Anfrage] Folgt Rut Folke? Antwort: Ja
Superstar ist Folke.
Anzahl der Anfragen: 211
```

Eigenes Beispiel 5 (ebsp_superstar5.txt)

```
Neville Ron Hermine Harry
Neville Ron
Ron Hermine
Hermine Harry
Ron Harry
Neville Harry
Harry Neville
Harry Ron
```

Ausgabe für eigenes Beispiel 5

```
 [Anfrage] Folgt Neville Ron? Antwort: Ja
 [Anfrage] Folgt Ron Hermine? Antwort: Ja
 [Anfrage] Folgt Hermine Harry? Antwort: Ja
 [Anfrage] Folgt Harry Neville? Antwort: Ja
 Es gibt keinen Superstar.
 Anzahl der Anfragen: 4
```

Das eigene Beispiel Nr. 5, für welches ich Figuren einer bekannten Fantasy-Buchreihe benutzt habe, testet einen bestimmten Sonderfall. Testet man nämlich nicht, ob vom aktuellen Knoten aus auch keine Kanten zu bereits besuchten Knoten existieren[2], kann bei diesem Knoten ein falsches Ergebnis ausgegeben werden. Startet man nämlich bei *Neville*, bewegt sich dann zu *Ron* und über *Hermine* zu *Harry*, gibt es keinen noch nicht besuchten Knoten mehr, den man von *Harry* aus erreichen kann. Alle anderen Knoten folgen *Harry*, also könnte er theoretisch der Superstar sein. Dies ist er jedoch nicht, da er *Ron* und *Neville* folgt und somit nicht alle Kriterien erfüllt. Dies wird nur durch den genannten Test entdeckt.

4 Quellcode

```java
package de.lukasrost.bwinf2019.superstar;

import java.util.ArrayList;
import java.util.Arrays;

class Graph {
    private ArrayList<Vertex> vertices = new ArrayList<>(); //Knotenliste
    private ArrayList<Vertex> visited = new ArrayList<>(); //besuchte Knoten
    private Vertex current; //Startknoten
    private int anfrageCounter = 0;

```

[2] siehe Kommentar „keine Kanten zu besuchten Knoten existieren“ im Pseudocode

```
12      Graph(Vertex... nodes1){
13          vertices.addAll(Arrays.asList(nodes1));
14          current = vertices.get(0);
15      }
16
17      int getAnfrageCounter() {
18          return anfrageCounter;
19      }
20
21      private boolean hasEdge(Vertex start, Vertex end){ //Anfragemethode als
        ↪  einziger Zugriff auf Adjazenzliste
22          anfrageCounter++;
23          boolean hasEdge = start.getAdjacency().contains(end);
24          System.out.println("[Anfrage] Folgt "+start.getContent()+"
            ↪  "+end.getContent()+"? Antwort: "+(hasEdge ? "Ja" : "Nein"));
25          return hasEdge;
26      }
27
28      String modifiedDFS(){
29          return modifiedDFS(current,null);
30      }
31
32      private String modifiedDFS(Vertex start, Vertex parent){
33          visited.add(start);
34          Vertex next = null;
35
36          for (Vertex vertex : vertices) { //Knoten zum rekursiven Abstieg
            ↪  bestimmen
37              if (!vertex.equals(start) && !visited.contains(vertex) &&
                ↪  hasEdge(start,vertex)){
38                  next = vertex;
39                  break;
40              }
41          }
42
43          if (next != null){ //Rekursion
44              return modifiedDFS(next,start);
45          } else {
46              for (Vertex vis: visited){ //start auf Abwesenheit von Kanten zu
                ↪  besuchten Knoten überprüfen
47                  if (!vis.equals(start) && hasEdge(start,vis)){
48                      return "";
49                  }
50              }
51              for (Vertex vertex : vertices){ //Vorhandensein von eingehenden
                ↪  Kanten von allen Knoten
52                  if (!vertex.equals(start) && !vertex.equals(parent) &&
                    ↪  !hasEdge(vertex,start)){
53                      return "";
54                  }
55              }
56              return start.getContent(); //Superstar gefunden
57          }
```

```
    }
}
```

Quellcode 1: Implementierung des ADT Graph: `Graph.java`

```
package de.lukasrost.bwinf2019.superstar;

import javax.swing.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

class SuperstarHelper {
    private File inputFile;
    private ArrayList<Vertex> vertices = new ArrayList<>();
    private HashMap<String,Vertex> nameToVertex = new HashMap<>();
    private Graph graph;
    private String superStar;

    void showFileSelectionWindow(){
        //Benutzerauswahl der einzulesenden Datei und Umwandlung in ein
        ↪ File-Objekt
        JFileChooser chooser = new JFileChooser();
        File file = null;
        int rueckgabeWert = chooser.showOpenDialog(null);
        if (rueckgabeWert == JFileChooser.APPROVE_OPTION){
            file = chooser.getSelectedFile();
        } else {
            System.exit(0);
        }
        inputFile = file;
    }

    void readToGraph(){ //Einlesen der Eingabedatei
        try (BufferedReader br = new BufferedReader(new
        ↪ FileReader(inputFile)))
        {
            boolean first = true;
            for (String line; (line = br.readLine()) != null;)
            {
                if (first){ //erste Zeile: Knoten
                    first = false;
                    for (String name : line.split(" ")){
                        Vertex v = new Vertex(name);
                        vertices.add(v);
                        nameToVertex.put(name,v);
                    }
                } else { //andere Zeilen: Kanten
                    String[] edge = line.split(" ");
```

```
46                      nameToVertex.get(edge[0]).
                        ↪ addAllToAdjacency(nameToVertex.get(edge[1]));
47                  }
48              }
49          } catch (IOException e) {
50              e.printStackTrace();
51          }
52          graph = new Graph(vertices.toArray(new Vertex[0]));
53      }
54
55      void generateSolution(){ //Tiefensuche durchführen
56          superStar = graph.modifiedDFS();
57      }
58
59      String getOutput(){ //Ausgabe auf die Konsole
60          if (!"".equals(superStar)){
61              return "Superstar ist " + superStar + ".\nAnzahl der Anfragen: " +
                ↪ graph.getAnfrageCounter();
62          } else {
63              return "Es gibt keinen Superstar.\nAnzahl der Anfragen: " +
                ↪ graph.getAnfrageCounter();
64          }
65      }
66  }
```

Quellcode 2: Ein- und Ausgabe: `SuperstarHelper.java`