Aufgabe 1
„Superstar“- Dokumentation

37. Bundeswettbewerb Informatik 2018/19 - 1. Runde

Sebastian Baron, Simon Fiebich, Lukas Rost

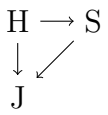
Team-ID: 00036

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)
    else
        for all Knoten u in visited do
            if hasEdge(start,u) then
                return kein Superstar
            end if
        end for
        for all Knoten k (ohne start) do
            if not hasEdge(k,start) then
                return kein Superstar
            end if
        end for
        return start
    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*.

Literatur

[1] Wikipedia-Artikel zur Tiefensuche, `https://de.wikipedia.org/wiki/Tiefensuche`

[2] Steven S. Skiena: The Algorithm Design Manual, ISBN 978-1-84800-069-8

2 Umsetzung

3 Beispiele

4 Quellcode

```
1  package de.lukasrost.bwinf2019.superstar;
2
3  import javax.swing.*;
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10
11 class SuperstarHelper {
12     private File inputFile;
13     private ArrayList<Vertex> vertices = new ArrayList<>();
14     private HashMap<String,Vertex> nameToVertex = new HashMap<>();
15     private Graph graph;
16     private String superStar;
17
18     void showFileSelectionWindow(){
19         //Benutzerauswahl der einzulesenden Datei und Umwandlung in ein
           ↪  File-Objekt
20         JFileChooser chooser = new JFileChooser();
21         File file = null;
22         int rueckgabeWert = chooser.showOpenDialog(null);
23         if (rueckgabeWert == JFileChooser.APPROVE_OPTION){
24             file = chooser.getSelectedFile();
25         } else {
26             System.exit(0);
27         }
28         inputFile = file;
29     }
30
31     void readToGraph(){
32         try (BufferedReader br = new BufferedReader(new
           ↪  FileReader(inputFile)))
33         {
34             boolean first = true;
35             for (String line; (line = br.readLine()) != null;)
36             {
37                 if (first){
```

```
                    first = false;
                    for (String name : line.split(" ")){
                        Vertex v = new Vertex(name);
                        vertices.add(v);
                        nameToVertex.put(name,v);
                    }
                } else {
                    String[] edge = line.split(" ");
                    nameToVertex.get(edge[0]).addAllToAdjacency(nameToVertex.get(edge[1]));
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        graph = new Graph(vertices.toArray(new Vertex[0]));
    }

    void generateSolution(){
        superStar = graph.modifiedDFS();
    }

    String getOutput(){
        if (!"".equals(superStar)){
            return "Superstar ist " + superStar + ".\nAnzahl der Anfragen: " + graph.getAnfrageCounter();
        } else {
            return "Es gibt keinen Superstar.\nAnzahl der Anfragen: " + graph.getAnfrageCounter();
        }
    }
}
```

Quellcode 1: Ein- und Ausgabe: `SuperstarHelper.java`

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

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

class Graph {
    private ArrayList<Vertex> vertices = new ArrayList<>();
    private Vertex current;

    private int anfrageCounter = 0;

    Graph(Vertex... nodes1){
        vertices.addAll(Arrays.asList(nodes1));
        current = vertices.get(0);
    }

    int getAnfrageCounter() {
        return anfrageCounter;
```

```
19      }
20
21      private boolean hasEdge(Vertex start, Vertex end){
22          anfrageCounter++;
23          return start.getAdjacency().contains(end);
24      }
25
26      String modifiedDFS(){
27          return modifiedDFS(current,new ArrayList<>(),null);
28      }
29
30      private String modifiedDFS(Vertex start, ArrayList<Vertex> visited, Vertex
        ↪ parent){
31          visited.add(start);
32          Vertex vt = null;
33
34          for (Vertex vertex : vertices) {
35              if (!vertex.equals(start) && !visited.contains(vertex) &&
                ↪ hasEdge(start,vertex)){
36                  vt = vertex;
37                  break;
38              }
39          }
40
41          if (vt != null){
42              return modifiedDFS(vt,visited,start);
43          } else {
44              for (Vertex vertex : vertices){
45                  if (!vertex.equals(start) && !vertex.equals(parent) &&
                    ↪ !hasEdge(vertex,start)){
46                      return "";
47                  }
48              }
49              return start.getContent();
50          }
51      }
52  }
```

Quellcode 2: Implementierung des ADT Graph: `Graph.java`