Bundeswettbewerb Informatik: Beispiellösung

Aufgabenstellung: tratsCH trATsch

Die Leute vom Planeten Chator schreiben gern Schlechtes übereinander. Wer vielen über andere Schlechtes schreibt, gilt als besonders charmant. Aber natürlich nur, wenn die Kompromittierten nichts davon erfahren.

Chatonen schreiben nur an Leute, die Ihnen sympathisch sind. Doch die können den Tratsch weitertragen, und eventuell genau an den Falschen. Ein Chatone muss also gut aufpassen, dass er keinen Charmefehler macht. Dieses Missgeschick passierte unlängst Ator, als er Btor Schlechtes über Dtor schrieb. Zu dumm: Dtor ist dem Ctor sympathisch, der wiederum Btor sympathisch ist. Und so landete der Tratsch bei Dtor, der über Ator verständlicherweise sehr verärgert war. Dies hätte Ator mit ein wenig Übersicht vermeiden können, denn schließlich wissen alle Chatonen voneinander, wer wem sympathisch ist.

Aufgabe

Programmiere einen Charminator, der einliest, welche Chatonen welchen anderen sympathisch sind. Er soll auf dieser Grundlage möglichst kompakt für alle Chatonen ausgeben, wem der Betreffende über wen Schlechtes schreiben kann, ohne einen Charmefehler zu riskieren.

Teste dein Programm an drei Beispielen für mindestens 5 und höchstens 10 Chatonen; verwende auf jeden Fall das folgende Beispiel:

Dem Chatonen	sind sympathisch:
A	B E
B	C
C	D G
D	C
E	F
F	B E G
G	H
H	D

Bedenke dabei, dass eine Tratschnachricht von Chatone x nach Chatone y weder x noch y betrifft und dass ein Chatone eine Nachricht, die ursprünglich von ihm selbst stammt und wieder bei ihm ankommt, nicht weiterleiten wird – so dumm sind Chatonen nicht.

Lösungsidee *nach Benjamin Berger*

Die Sympathiebeziehungen der Chatonen werden als ein gerichteter Graph aufgefasst: die Chatonen sind die Knoten, die direkten Sympathiebeziehungen die gerichteten Kanten. Dann wird der Weg einer Nachricht von einem Chatonen S (Sender) an einen dem S sympathischen Chatonen E (Empfänger) für alle möglichen Paare (S,E) wie folgt nachvollzogen: Der Graph wird beginnend bei E rekursiv durchsucht (Tiefensuche), bis entweder S selbst erreicht wird (der die Nachricht nicht weiterleitet) oder ein gefundener Chatone schon besucht wurde, die Nachricht

also schon erhalten hat. Über alle Chatonen, die *nicht* besucht wurden, kann S bei E tratschen – da E an sie den Tratsch eben nicht weiterleiten kann.

Programm-Dokumentation

Die Lösungsidee wurde in Java umgesetzt. Das Programm besteht aus zwei Klassen: Die Klasse `Chatone` stellt Variablen für den Namen eines Chatonen und die Liste der ihm sympathischen Chatonen sowie die Methode für die Suche nach vom Empfänger einer Nachricht aus erreichbaren Chatonen zur Verfügung. Die Klasse `Charminator` verwaltet alle während des Programmlaufs erzeugten Chatonen-Objekte und enthält die Hauptprozedur.

Die Hauptprozedur liest eine Sympathietabelle als Zeilen von Zeichenketten (Strings) ein, wobei der erste String den Namen eines Chatonen und alle folgenden Strings die Namen der ihm sympathischen Chatonen angibt. Nach dem Einlesen einer Zeile werden für neue Namen die passenden Chatonen-Objekte angelegt; dies regelt die Prozedur `getChatone`. Nach der Eingabe werden für alle Chatonen S die Lästermöglichkeiten berechnet und ausgegeben: Für jeden sympathischen Chatonen E wird die Suchmethode `sucheErreichbareVon` aufgerufen; falls das Komplement der zurückgegebenen Menge nicht leer ist, wird ausgegeben, dass S bei E über die darin enthaltenen Chatonen lästern kann.

Die Suchmethode setzt die Lösungsidee direkt um. Die Variable erreichbar speichert die bei der Suche vom Empfänger E aus besuchten und damit von E aus erreichbaren Chatonen. Zunächst wird der Sender S in dieser Variable gespeichert, damit bei S die Suche endet. Nun wird die rekursive Prozedur `sucheErreichbare` für E aufgerufen. Sie beendet den Suchzweig, wenn der aktuelle Chatone schon besucht wurde. Ansonsten speichert sie diesen als erreichbar und setzt die Suche durch rekursiven Aufruf für alle sympathischen Chatonen fort.

Programm-Ablaufprotokoll

Aus Platzgründen kann hier nur ein Ablaufprotokoll abgedruckt werden. Das Programm liest die Sympathietabelle der Aufgabenstellung ein und gibt die Lästermöglichkeiten aus:

```
$ java Charminator
Gib die Zeilen der Sympathietabelle ein:
Chatone Sympathisch1 Sympathisch2 ...
End-of-File (Ctrl-D) beendet die Eingabe.
A B E
B C
C D G
D C
E F
F B E G
G H
H D
A kann bei B tratschen ueber: E F
E kann bei F tratschen ueber: A
F kann bei E tratschen ueber: A G B D C H
F kann bei G tratschen ueber: A E B
```

```
F kann bei B tratschen ueber: A E
G kann bei H tratschen ueber: A E F B
B kann bei C tratschen ueber: A E F
D kann bei C tratschen ueber: A E F B
C kann bei G tratschen ueber: A E F B
C kann bei D tratschen ueber: A E F G B H
H kann bei D tratschen ueber: A E F B
$
```

Programm-Text

```
import java.util.*;
import java.io.*;

/** Diese Klasse enthält die main()-Methode. **/
public class Charminator{
 //Die Menge aller erzeugten Chatonen-Objekte.
 static Set chatonen=new HashSet();
 //Die Zuordnung von Namen zu Chatonen.
 static Map namen=new HashMap();

 public static void main(String[] args)throws IOException{
  //Begrüßung & Instruktionen
  System.out.println("Gib die Zeilen der Sympathietabelle ein: ");
  System.out.println("Chatone Sympathisch1 Sympathisch2 ...");
  System.out.println("End-of-File (Ctrl-D) beendet die Eingabe.");

  BufferedReader inReader=
    new BufferedReader(new InputStreamReader(System.in));
  //Einleseschleife
  while(true){
   String zeile=inReader.readLine();
   if(zeile==null)break;  //End-Of-File beendet das Einlesen.
   //Zum Zerlegen der Eingabe in die Namen-Strings:
   StringTokenizer stizer=new StringTokenizer(zeile, " ");
   String chatName=stizer.nextToken(); //erster Name: Chatone
   while(stizer.hasMoreTokens()) //alle weiteren: sympathisch
      getChatone(chatName).
        sympathisch.add(getChatone(stizer.nextToken()));
  } //Ende der Einleseschleife

  //Alle Chatonen können Sender sein.
  for(Iterator i=chatonen.iterator(); i.hasNext(); ){
   Chatone sender=(Chatone)i.next();
   //Alle Sympathischen können Empfänger sein.
   for(Iterator j=sender.sympathisch.iterator(); j.hasNext(); ){
    Chatone empfaenger=(Chatone)j.next();
    //Alle vom Empfänger Erreichbaren sollten ...
    Set erreichbar=sender.sucheErreichbareVon(empfaenger);
    Set laesternUeber=new HashSet(chatonen); //(kopieren)
    //... kein Lästerobjekt sein:
    laesternUeber.removeAll(erreichbar);
    //Ausgabe, falls der Sender lästern darf:
    if(laesternUeber.size() != 0){
     System.out.print(sender + " kann bei " +
```

```
                    empfaenger + " tratschen ueber: ");
     for(Iterator k=laesternUeber.iterator(); k.hasNext(); )
                 System.out.print(k.next() + " ");
     System.out.println();
    }
   }
  }
 } //end Main

 /* Gibt ein Chatone-Objekt aus "namen" zurück. Falls nötig,
    wird der Eintrag neu erzeugt (auch in "chatonen"). */
 static Chatone getChatone(String name){
  Chatone ergebnis=(Chatone)namen.get(name);
  if(ergebnis==null){ //Name unbekannt
   ergebnis=new Chatone(name);
   namen.put(name, ergebnis);
   chatonen.add(ergebnis);
  }
  return ergebnis;
 }
}

/** Diese Klasse realisiert die wichtigen Eigenschaften
    eines Chatonen: Name, sympathische Chatonen und
    Erreichbarkeitssuche. **/
class Chatone{
 //Diejenigen, denen dieser Chatone schreiben kann
 HashSet sympathisch=new HashSet();
 //Der Name dieses Chatonen
 final String name;
 public Chatone(String name_){ name=name_; }
 //Gibt den Namen des Chatonen zurück
 public String toString(){ return name; }

 /* Bestimme alle Chatonen, die eine Nachricht dieses
    Chatonen an einen Empfänger erreichen kann. */
 public Set sucheErreichbareVon(Chatone empfaenger){
  Set erreichbar=new HashSet(); //besuchte=erreichbare Chatonen
  erreichbar.add(this); //Beim Sender soll die Suche enden.
  //nun startet die Suche
  empfaenger.sucheErreichbare(erreichbar);
  return erreichbar;
 }

 /* rekursive Suchprozedur */
 void sucheErreichbare(Set erreichbar){
  //Wenn dieser Chatone schon erreicht wurde, endet die Suche.
  if(erreichbar.contains(this)) return;
  //Sonst ist er nun erreicht; die Suche wird bei ihm fortgesetzt.
  erreichbar.add(this);
  for(Iterator i=sympathisch.iterator(); i.hasNext(); )
   ((Chatone)i.next()).sucheErreichbare(erreichbar);
 }
}
```