Aufgabenblatt 6
Hashing und Graphen

- Abgabetermin: **Freitag, 11.07.14 23:59 Uhr**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Abgabe:
  - Geben Sie im Abgabesystem[1] eine ZIP-Datei ab, welche den Quelltext aller implementierten Klassen (Java 7), sowie entsprechende textuelle Lösungen enthält.
  - Achten Sie darauf, dass ihr ZIP-Archiv *keine* kompilierten Klassen (class-Dateien) enthält.
  - Textuelle Lösungen reichen Sie bitte in einer entsprechend benannten *PDF*-Datei (aufgabe_x.pdf) auf oberster Ebene im ZIP-Archiv ein.
  - Der Quelltext soll in einer Ordnerstruktur analog zu assignment6_sources.zip[2] vorliegen: Quellcode im `src` Ordner, etc.
  - Achten Sie darauf, dass Ihr Quelltext gut kommentiert ist.
  - Geben Sie bei der Einreichung Ihrer Lösung den Namen des anderen Gruppenmitglieds unter "Authors" an (pro Gruppe nur eine Abgabe).

Vorbereitung

Laden Sie sich das Archiv assignment6_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment6_sources.zip`

Aufgabe 1: Hashing

9 P

a) Implementieren Sie mit Klasse `HashMultiMap<K,V>` im Package `assignment6` ein assoziatives Array, das pro Schlüssel mehrere Elemente speichern kann[3]. Die Zuordnung von Schlüsselwerten soll nach der Funktionsweise einer Hashtabelle geschehen und dabei die im Konstruktor übergebene Hashfunktion `assignment6.hashing.HashFunction<T>` nutzen (eine vorhandene Implementierung finden Sie in Klasse `assignment6.hashing.DefaultHashFunction<T>`). **Tipp:** Damit die Hashtabelle gleichmäßig belegt ist achten sie darauf Tabellengrößen in 2er Potenzen zu wählen.

b) Implementieren Sie eine alternative Hashfunktion `assignment6.hashing.MyVeryOwnHashFunction` für `String`-Werte. Lassen Sie dabei Ihrer Phantasie freien Lauf.

c) Bloomfilter[4] sind probabilistische Datenstrukturen, die mit Hashfunktionen eng verwandt sind und eine sehr speicherschonende Repräsentation von Objektmengen ermöglichen. Bloomfilter lassen lediglich Aussagen zu, ob ein Element definitiv nicht in der Objektmenge enthalten ist (true negative), bzw. ob es enthalten sein könnte (true/false positive).

Ein Bloomfilter basiert auf k verschiedenen Hashfunktionen. Eine Objektmenge wird mit einem Array aus m Bits repräsentiert (`boolean[] bits`), die anfangs alle 0 (`false`) sind (der Filter ist leer). Wird nun ein Objekt zum Filter hinzugefügt (`add(String v)`), werden für Objekt `v` alle Hashwerte $h_1^{\mathtt{V}}, \ldots, h_k^{\mathtt{V}}$ der k Hashfuntionen berechnet. Anschließend wird für jeden Hashwert h_i das Bit an Stelle s auf 1 (`true`) gesetzt (`bits[`$s(h_i^{\mathtt{V}})$`] = true;`). Die Stelle ist definiert durch:

$$s(h_i^{\mathtt{V}}) = h_i^{\mathtt{V}} \bmod m.$$

Zur Überprüfung ob ein Element `w` in der Objektmenge enthalten ist (`mightContain(w)`), werden alle Hashwerte $h_1^{\mathtt{W}}, \ldots, h_k^{\mathtt{W}}$ berechnet, und es wird überprüft ob, *alle* zugehörigen Bits auf 1 gesetzt wurden:

$$\forall s \in \bigcup_{i=1}^{k} s(h_i^{\mathtt{W}}) : \texttt{bits[}s\texttt{] == true}$$

Ist dies nicht der Fall, kann ausgeschlossen werden, dass `w` in der Objektmenge enthalten ist.

Ansonsten kann lediglich vermutet werden, dass sich `w` in der Objektmenge befindet. Die Wahrscheinlichkeit einer Falschmeldung (false positive) liegt für einen Filter in dem sich bereits n Elemente befinden bei

$$P_{fp} \approx \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k$$

vorausgesetzt, die Hashfunktionen sind unabhängig und jede Array-Position wird von jeder Funktion gleicher Wahrscheinlichkeit adressiert.

Implementieren Sie `assignment6.StringBloomsfilter` als Bloomsfilter über String Elemente. Verwenden sie bei ihren Tests unterschiedliche Hashfunktionen.

[3] Sie dürfen für ihre Implementierung alle nicht-hashbasierten Java Collection-API Klassen verwenden (ausgeschlossen sind also: `java.util.HashSet`, `java.util.Hashtable`, `java.util.HashMap`, ...)

[4] `http://de.wikipedia.org/wiki/Bloomfilter`

Aufgabe 2: Graphen

6 P

a) Implementieren Sie die Klasse `assignment6.WeightedGraph`, eine Datenstruktur zur Speicherung von gerichteten Graphen mit gewichteten Kanten. Nutzen Sie zum Speichern eine Adjazenzlistenrepräsentation (Sie können auch gerne Ihre `HashMultiMap`-Implementierung aus Aufgabe 1 nutzen).

b) Diskutieren Sie kurz die Vor- und Nachteile der Speicherung eines Graphen in Form einer Adjazenzliste im Gegensatz zur Adjazenzmatrix einer PDF-Datei aufgabe_2.pdf und fügen Sie sie dem Abgabearchiv hinzu.

c) Implementieren Sie die Methode `connectedComponents` der Klasse `assignment6.ConnectedComponentFinder` zur Komponentenfindung (Zusammenhangskomponenten[5]. Laden Sie sich die beiden Dateien `cast.action.sample.txt` sowie `cast.action.txt` aus dem HPI-Materialien-Ordner[6]. Sie können diese Dateien mithilfe der Klasse `assignment6.GraphFileReader` einlesen. Die resultierenden Graphen beinhalten Schauspieler (Knoten) und deren Beziehungen zueinander, die bis 2006 in Actionfilmen mitgespielt haben. Eine Kante stellt dabei eine "haben in mindestens einem Film zusammen gespielt" Abhängigkeit dar. Finden Sie heraus, wie viele (schwache) Zusammenhangskomponenten im Graph der Datei `cast.action.txt` vorhanden sind. Ermitteln Sie zusätzlich wie viele Knoten in der größten Komponente liegen. Speichern sie beide Werte in der oben genannten PDF-Datei (aufgabe_2.pdf). Die Gewichte der Kanten spielen dabei keine Rolle.

Aufgabe 3: Kürzeste Pfade

5 P

Implementieren Sie den Dijkstra-Algorithmus zum finden den kürzesten Pfades zwischen zwei Knoten `start` und `end` (Methode `shortestPath(WeightedGraph g, String start, String end)` in Klasse `assignment6.Dijkstra`). Das erwartete Ergebnis ist ein Array mit dem Pfad (Knoten) zwischen `start` und `end` im Graphen.

Zum Beispiel ergibt sich aus `start = "Wilson, Owen (I)"` und `end = "Lee, Bruce (I)"` für den Graphen aus `cast.action.sample.txt` der Pfad:
`["Wilson, Owen (I)", "Schwarzenegger, Arnold", "Van Damme, Jean-Claude", "Lee, Bruce (I)"]`.

Die Gewichte der Graphen aus den bereits genannten Beispieldateien entsprechen den relativen Häufigkeiten von Filmen, die Schauspieler A (`from`) ohne Schauspieler B (`to`) gespielt hat. Es handelt sich also um Entfernungen.

Achten Sie bei Ihrer Implementierung darauf die Warteschlange der zu überprüfenden Knoten effizient zu speichern. Diskutieren Sie kurz für welche Speicherungsart der Warteschlange Sie sich entschieden haben und nennen Sie Vor- und Nachteile dieser Form im Vergleich zu einer einfachen Liste. Speichern Sie die Diskussion einer PDF-Datei aufgabe_3.pdf und fügen Sie sie dem Abgabearchiv hinzu.

[5] `http://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie)`
[6] `\\fs23\lehrveranstaltungen\FG_Informationssysteme\VL PT II\Uebung\`

Zusatzaufgabe: Fußball-Weltmeisterschaft

Die Crews von Qatar Airways[7] sind große Fußballfans. Aus diesem Grund sind sie natürlich an den Spielen der Fußball-Weltmeisterschaft interessiert und wollen keine spannende Szene verpassen. Leider werden die Spiele in Qatar nur im Pay-TV (bei beIN Sports) übertragen. Dieses Angebot können sich die Angestellten leider nicht leisten. Außerdem hat die Crew nur während der Flüge genug Zeit um alle Szenen zu sehen.

Zum Glück fliegt Qatar Airways auch 3 Flughäfen in Deutschland an (Berlin, Frankfurt und München)[8], denn in Deutschland werden die interessantesten Szenen kostenlos über die ZDF Mediathek ins Internet gestellt. Leider ist der Anbieter (ZDF) verpflichtet, die Videos nach einer gewissen Zeit wieder aus dem Online-Angebot zu löschen. Insgesamt werden während der Weltmeisterschaft N Szenen in das Angebot der Mediathek übernommen (Nummeriert von 0 bis $N-1$). Dabei ist das i-te Video nur in der Zeit zwischen s_i und e_i (in) online (Start- und Endzeit). Die Crews von Qatar Airways besuchen Deutschland nur sehr kurz und da der Rückflug direkt vorbereitet werden muss, haben die Mitarbeiter jeweils nur einen kurzen Moment (Zeitpunkt t) um alle verfügbaren Videos herunterzuladen. Durch die schnelle Internetverbindung am Flughafen kann der Download natürlich sofort abgeschlossen werden.

Sie erhalten nun die Besuchszeiten von Q unterschiedlichen Qatar Airways Crews sowie die Zeiten, in denen die N unterschiedliche Videoclips online sind (zwischen Startzeitpunkt $s_0, \ldots, s_{N-1}$ und Endzeitpunkt $e_0, \ldots, e_{N-1}$). Die Zahl der Besuche kann dabei natürlich von Crew zu Crew variieren, jedoch ist sie maximal K. Ihre Aufgabe ist es nun herauszufinden, wie viele Szenen die jeweilige Crew während der Flüge genießen kann.

Implementieren Sie dazu die Methode `numOfClips(int[] s, int[] e, int[][] visits)` der Klasse `assignment6.WorldCup` effizient. Die Parameter `s[i]` und `e[i]` entsprechen dem Start- bzw. Endzeitpunkt zu dem der i-te Clip online verfügbar ist ($0 \leq i < N$). Die Werte `visits[l]` entsprichen den Besuchszeiten der l-ten Qatar Airways Crew ($0 \leq l < Q$), wobei `visits[l][m]` dem Zeitpunkt t des m-ten Besuchs eben dieser Crew entspricht.

Wertebereiche
$1 \leq N \leq 100\,000$ (10^5)
$1 \leq Q \leq 5\,000$ $(5 \cdot 10^3)$
$1 \leq K \leq 20$
$1 \leq s_i, e_i, t \leq 1\,000\,000\,000$ (10^9)
$s_i \leq e_i$

Beispiel

```
// 4 clips (clip zero is online between time 1 and 4, ...)
int[] s = new int[] {1, 3,2,5};
int[] e = new int[] {4,10,6,8};
// 3 crews (crew zero visits Germany once at t=5, ...)
int[][] visits = new int[][] {{5},{2,6},{1,10,9}};

numOfClips(s, e, visits); // -> {3,4,2}
// crew zero: visits at t=5: 3 clips are online (1, 2, 3)
// crew one: visits at t=2 and t=6: all four clips are online
// crew two: visits at t=1, t=10 and t=9: 2 clips are online (0, 1)
```

[7] http://www.qatarairways.com/
[8] http://qatar.innosked.com/