Aufgabenblatt 6
Hashing und Graphen

- Abgabetermin: **Samstag, 16.07.2016 23:55 Uhr**
- Zur Prüfungszulassung müssen in einem Aufgabenblatt mind. 25% der Punkte erreicht werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bestanden werden.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Jede Hausaufgabe enthält eine optionale Zusatzaufgabe, die bei erfolgreicher Bearbeitungen 1 Klausurpunkt zählt.
- Abgabe:
  - Die Abgabe erfolgt über CodeOcean[1]. CodeOcean ermöglicht die Bearbeitung und Ausführung der Quelldateien im Browser. Bei Bedarf können die Sources (Quelltexte) und Unit Tests von der Übungs-Webseite[2] heruntergeladen werden, um lokal zu entwickeln.
  - CodeOcean benutzt Java Version 8.
  - Alle Aufgaben sind über das CodeOcean einzureichen. CodeOcean ermöglicht das Anfügen von Kommentaren bei der Abgabe, darüber hinaus können auch Kommentare in den Source-Code geschrieben werden.
  - Geben Sie bei **jeder** Aufgabe beide Namen Ihrer Gruppe an.
  - Achten Sie darauf, dass Ihr Quelltext hinreichend kommentiert ist.

Aufgabe 1: Hashing

8 P

Implementieren Sie eine HashMap, welche `Particle` auf Integer mappen kann. Ein `Particle` ist ein zweidimensionaler Vektor aus Integern.

a) Implementieren Sie eine Hashfunktion in der Klasse `assignment6.ParticleHashFunction`, welche ein `Particle` hasht. Achten Sie darauf, dass zufällig verteilte `Particle` möglichst unterschiedliche Hashwerte generieren.

b) Implementieren Sie die Klasse `ParticleToIntHashMap`. Ihre Hashmap soll über eine feste Anzahl an Buckets verfügen. Implementieren Sie die Funktionen `void put(Particle key, int value)` zum Einfügen eines Wertes und `Integer get(Particle key)` zum Auslesen eines Wertes. Überlegen Sie sich, wie Sie mit Hashkollisionen umgehen. Untesuchen Sie die Laufzeit für das Einfügen und Auslesen von n Werten im schlechtesten Fall in form eines Quellcodekommentars.

c) Implementieren Sie die Funktion int `ParticleToIntHashMap.largestBucketSize()`. Diese soll den größten Bucket in Ihrer HashMap finden und dessen Größe zurückgeben.

d) Überlegen sie sich wie `Particle` Objekte aussehen müssen, um mit der implementierten Hash-Funktion den selben Hash zu generieren. Implementieren Sie die statische Funktion `HashMapAdversary.generateWorstCase(int keyCount)`, welche eine Liste der Länge `keyCount` mit unterschiedlichen `Particle` erstellt, welche alle den selben Hash generieren.

e) Denken Sie sich eine zweite Hashfunktion aus, welche für den Fall der vorherigen Liste besser funktioniert. Implementieren Sie diese in `OtherParticleHashFunction.hash(Particle input)`.

[1] `https://open.hpi.de/courses/pt2-2016`
[2] `https://hpi.de/naumann/teaching/current-courses/ss-16/uebung-programmierechnik-ii.html`

Aufgabe 2: Bloomfilter

10 P

Bloomfilter[3] 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.

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

Sie haben nun die Aufgabe, einen Bloomfilter und Hashfunktionen über String-Elemente zu erstellen.

a) Implementieren Sie eigene Hashfunktionen in der Klasse `assignment6.CustomHashFunction`. Implementieren Sie mindestens 3 verschiedene Hashfunktionen, die über den Konstruktor spezifizierbar sind. Ändern Sie die Namen der ersten 3 Varianten nicht, da die Tests diese benutzen.[4] Seien Sie kreativ! Die zu verwendenden Hashfunktionen müssen alle das Interface `assignment6.JavaMDHashFunction` implementieren.

b) Implementieren Sie `assignment6.StringBloomfilter` als Bloomfilter über Strings. Die Länge des Bit-Arrays sei über den Parameter `m` einstellbar.

c) Machen Sie sich mit der Klasse `assignment6.JavaMDHashFunction` vertraut. Diese berechnet Hashes nach *MD5*, *SHA-1* und *SHA-256*, indem die Java-Funktionalität Message Digest[5] benutzt wird.

d) Evaluieren Sie Ihren Bloomfilter in der Klasse `assignment6.BloomfilterEvaluator`. Vergleichen Sie mindestens die Kombinationen von:

- Einen Bloomfilter, der ihre eigenen drei Hashfunktionen benutzt, einen, der die drei vorgegebenen kryptographischen Hashes benutzt und einen Bloomfilter, der sämtliche Hashfunktionen benutzt.
- Eine Bitfilterlänge von $\{10, 1.000, 100.000\}$.
- Eine lexigographisce Ordnung aller Wörter mit einer randomisierten Ordnung aller Wörter.

Führen Sie mindestens die folgenden Experimente durch:

- Messen Sie die Zeit, die ein Bloomfilter zum sukzessiven Einfügen aller gegebenen Wörter (durch die Methode `readWords`) benötigt.
- Messen Sie die danach die Zeit, um in diesem gefüllten Bloomfilter für alle gegebenen Wörter nachzuprüfen, ob sie in der Objektmenge enthalten sein könnten.
- Fügen sie in einen leeren Bloomfilter sukzessive alle gegebenen Wörter ein und ermitteln Sie vor dem Einfügen, ob ein *false positive* vorliegt. nach dem wievielten Wort tritt das erste false positive auf, nach dem wievielten Wort die erste Folge von 5 bzw. 20 false positives? Sie können annehmen, dass die Methode `readWords` keine Duplikate ausgibt.
- Treten false positives eher auf, wenn Sie sortierte Daten einfügen als wenn Sie zufällig sortierte Daten einfügen? Worauf würde das hindeuten, wenn dies bei vielen Testdaten konsistent auftritt?

[4] Normalerweise würde man jede Hashfunktion in einer eigenen Klasse implementieren. Da dies jedoch bei CodeOcean zu Problemen beim Anlegen von neuen Dateien und der Korrektur führen kann wählen wir hier bewusst diesen unschönen Weg.

[5] `https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html`

Aufgabe 3: Six Degrees of Bacon

8 P

Sie erhalten die Datei `actors.csv`, die Daten zu Schauspielern und Filmen enthält. Die Zeilen repräsentieren Filme, sind kommasepariert und enthalten eine `movieid`, einen `title` und das Feld `actors`. Letzteres besteht aus den Namen der am jeweiligen Film beteiligten Schauspieler, separiert durch das Verkettungszeichen | (senkrechter Strich, Pipe).

Implementieren Sie die Methode `readcsv` der Klasse `assignment6.ActorGraph`, in der Sie die betreffende Datei in eine Graphstruktur Ihrer Wahl einlesen. Ziel ist es, *Six Degrees of Bacon*[6] zu spielen. Es soll also anschließend mithilfe Ihres Graphen die Bacon-Zahl[7] eines Schauspielers ermittelt werden. Dazu implementieren Sie weiterhin die Methode `baconness`, in der genau dies geschieht.

Da Ihnen sowohl bei der Wahl des Graphen als auch beim Algorithmus zur Baconness-Bestimmung freie Wahl besteht, achten Sie bitte auf *gut nachvollziehbar kommentierten* und *für den Zweck effizienten* Code.

Beantworten Sie anschließend in einem Queltextkommentar die folgenden Fragen:

- Welche Bacon-Zahl hat "Hugo Weaving"?
- Wie viele Schauspieler haben die Bacon-Zahl 4?
- Wie lautet die durchschnittliche Bacon-Zahl aller Schauspieler (zwei Nachkommastellen)?

[6] `https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon`
[7] `https://de.wikipedia.org/wiki/Bacon-Zahl`

Zusatzaufgabe: EM Fernbus AG

Kurz vor Beginn der EM 2016 kam Ihnen eine geniale Geschäftsidee: Ein Fernbus-Unternehmen speziell für Fußballfans, um schnell von einem Spielort zum anderen zu kommen. Die Busse können Sie kostengünstig von Anbietern leihen, die Fluggäste von Tegel zum neuen Berliner Flughafen transportieren wollen, aber noch auf die Eröffnung nächsten Monat warten müssen. Daher ist Ihre einzige große Aufgabe, ein Streckennetz aufbauen, was genau alle Austragungsorte miteinander verbindet, und Live Übertragungen von Spielen während der Fahrten anbieten.

Da Sie am HPI studieren und während der Vorlesung *Wirtschaftliche Grundlagen* aufgepasst haben wissen Sie, dass es mit dieser groben Idee noch nicht getan ist. Sie machen sich deswegen an eine genauere Planung, wobei Sie unter anderem ihre Ausgaben einschätzen wollen. Ein hierbei zu beachtender Faktor ist eine neulich eingeführte Abgabe, mit der sich Busunternehmen an der Erhaltung der Autoroutes (französische Autobahnen) beteiligen sollen. Für jede Autoroute in ihrem Streckennetz müssen Sie eine jährliche Abgabe abhängig von der Anzahl der befahrenen Kilometer zahlen.

Da Sie französischen Landstaßen nicht trauen planen Sie, alle ihre Strecken über Autoroutes zu schicken - erahnen aber, dass diese Abgabe recht teuer werden könnte. Um die Planung zu erleichtern, wollen Sie wissen, wie viel Sie für diese mindestens einplanen müssen.

Implementieren Sie hierfür in der Funktion `minimumTaxableKilometers` einen effizienten Algorithmus, der Ihnen die Mindestzahl an Kilometern berechnet, die von der Abgabe betroffen sind. Als Parameter übergeben bekommen Sie die Anzahl der Austragungsorte, die Sie verbinden wollen, und eine Menge von Verbindungen zwischen diesen. Jede Verbindung verbindet zwei Austragungsorte, und ist in beide Richtungen befahrbar. Außerdem ist jeweils angegeben, wie viele Kilometer über Autoroutes führen. Sie dürfen davon ausgehen, dass es immer möglich ist, über die gegebenen Verbindungen von einem Austragungsort zu jedem anderen zu gelangen.