Lehrstuhl für Informatik 2
Modellierung und Verifikation von Software

Prof. Dr. Ir. Joost-Pieter Katoen Christian Dehnert, Friedrich Gretz, Benjamin Kaminski, Thomas Ströder

Datenstrukturen und Algorithmen

Softwarewettbewerb

1 Einleitung

Im Bereich der automatischen Verarbeitung natürlicher Sprache spielen sogenannte n-Gramme — Tupel von n aufeinanderfolgenden Wörtern — eine wichtige Rolle. So kann man zum Beispiel für die deutsche Sprache statistisch ermitteln, dass auf die Worte "Sehr geehrter" fast immer das Wort "Herr" folgt, oder dass den Worten "freundlichen Grüßen" praktisch immer ein "Mit" vorangeht.

Die statistische Verarbeitung der n-Gramme basiert im Wesentlichen darauf, dass man auf großen Mengen geeigneter Texte — dem sogenannten Trainingscorpus — zählt, wie oft jedes Tupel von n Worten innerhalb dieses Textes vorkommt. Diese Zahl nennt man dann auch den **Count** $\text{count}(w)$ des entsprechenden n-Gramms w.

Allerdings gibt es hier ein Problem: bereits für ein verhältnismäßig kleines Vokabular der Größe $|V| = 10.000$ Wörter gibt es $|V|^2 = 100.000.000$ möglicher Bigramme (2-Gramme) und 1 Billion möglicher Trigramme (3-Gramme). In der Praxis werden die meisten möglichen n-Gramme zwar gar nicht vorkommen; dennoch muss man sich Gedanken machen, wie man möglichst effizient auf die Counts zugreift. Im Rahmen dieses Software-Wettbewerbs sollen Sie eine Datenstruktur entwerfen, die diesen Zugriff so effizient wie möglich umsetzt.

2 Aufgabenstellung

Die Eingabe Ihrer Datenstruktur sind eine Menge von Tupeln $D = \{\langle w, c\rangle\}$ wobei w ein Trigramm und c der entsprechende Count ist. Ihre Datenstruktur soll diese Daten geeignet **speichern**, um anschließend **Anfragen** zu beantworten.

Eine Anfrage $q = \langle w_1, w_2\rangle$ besteht aus zwei Trigrammen w_1 und w_2. Sei $D_{\geq w_1}^{\leq w_2}$ die Menge aller Trigramme aus D, die lexikographisch zwischen w_1 und w_2 liegen. Dann ist die Antwort auf die Anfrage q wiederum ein Tupel $\langle c_1, c_2\rangle$, wobei c_1 das Maximum der Counts und c_2 die Summe der Counts aller Trigramme in $D_{\geq w_1}^{\leq w_2}$ sind. Es soll also gelten:

$$c_1 = \max_{w \in D_{\geq w_1}^{\leq w_2}} \text{count}(w) \quad \text{und} \quad c_2 = \sum_{w \in D_{\geq w_1}^{\leq w_2}} \text{count}(w)$$

Beispiel Ist D durch die Tabelle 1 gegeben, so bedeutet dies, dass der Count des Trigramm ⟨sehr,geehrte,damen⟩ 130 beträgt. Die Anfrage $\langle w_1, w_2\rangle$ mit $w_1 = \langle$sehr,geehrte,damen$\rangle$ und $w_2 = \langle$sehr,geehrter,herr$\rangle$ sollte korrekterweise mit dem Tupel $\langle 130, 240\rangle$ beantwortet werden, da in diesem Fall

$$D_{\geq w_1}^{\leq w_2} = \{\langle\text{sehr,geehrte,damen}\rangle, \langle\text{sehr,geehrte,frau}\rangle, \langle\text{sehr,geehrter,herr}\rangle\}$$

und damit

$$c_1 = \max_{w \in D_{\geq w_1}^{\leq w_2}} \text{count}(w) = 130$$

und

$$c_2 = \sum_{w \in D_{\geq w_1}^{\leq w_2}} \text{count}(w) = 240$$

Bitte beachten Sie, dass Anfragen auch bisher unbekanntes Vokabular umfassen können. Beispielsweise ist die Anfrage $\langle w_3, w_4\rangle$ mit $w_3 = \langle$mit,unfreundlichen,gruessen$\rangle$ und $w_4 = \langle$sehr,geehrte,kollegin$\rangle$ legal und muss mit dem Tupel $\langle 130, 190\rangle$ beantwortet werden.

Trigramm w	count(w)
⟨mit,freundlichen,gruessen⟩	300
⟨sehr,geehrte,damen⟩	130
⟨sehr,geehrte,frau⟩	60
⟨sehr,geehrter,herr⟩	50

Tabelle 1: Eine Beispieleingabe D.

Ihre Aufgabe lautet: schreiben Sie eine Java-Klasse, die die genannten Eingaben geeignet speichert, um die Anfragen so effizient wie möglich zu beantworten.

3 Framework und Beispiel-Implementierung

Auf der Website

`http://moves.rwth-aachen.de/teaching/ss-15/dsal/`

finden Sie ein Archiv mit einem Code-Gerüst, mehreren Beispieleingaben und einer Beispiel-Implementierung. Die Klasse, die die Main-Methode enthält, heißt "TrigramMain" und kann mit `javac TrigramMain.java` kompiliert werden. Die Main-Methode liest zuerst eine Trigram-Datei und fügt alle gelesenen Trigramme mit dem entsprechenden Count in eine Instanz s des Interfaces `TrigramStorage` ein, indem die Methode

```
insert(TrigramAndCount trigram)
```

aufgerufen wird. Nachdem alle Trigramme gelesen wurden, wird s darüber informiert, indem dessen Methode `void build()` aufgerufen wird. Schlussendlich wird eine Query-Datei gelesen und alle gefundenen Queries an s gestellt mit Hilfe der Methode

```
QueryResult query(Trigram trigram1, Trigram trigram2)
```

Als (sehr naive) Beispielimplementierung des Interfaces `TrigramStorage` ist eine Klasse `SimpleTrigramStorage` mitgeliefert. Diese speichert alle eingehenden Trigramme mit ihrem Count in einer Liste. Wird eine Anfrage gestellt, so wird die Liste sequentiell durchlaufen und für jedes Trigramm der Liste geprüft, ob es lexikographisch zwischen den beiden Trigrammen der Anfrage liegt. Ist dies der Fall, wird der Count für die Berechnung des Maximums und des Durchschnitts benutzt. Um die naive Implementierung zu testen, können Sie die mitgelieferten Beispieleingaben benutzen. Beispielsweise sollte der Aufruf

```
java TrigramMain ../input/ulysses.in ../input/ulysses-queries1.in
```

im Verzeichnis `src` mit Hilfe der naiven Implementierung die folgende Ausgabe produzieren

```
Inserted all items and built data structure in 444ms.
WARNING: there were queries for which the returned result was not verified. [...]
All queries were answered successfully in 98ms.
```

Implementieren Sie Ihre Lösung als Klasse, welche das Interface `TrigramStorage` implementiert (also analog zu `SimpleTrigramStorage`).

4 Hinweise

- Ihre Lösung ist bis zum 5. Juli 2015 als Java-Quellcode mit dem Betreff "Softwarewettbewerb" an

  `dsal15@i2.informatik.rwth-aachen.de`

- Die gültigen Zeichen der Eingabe sind beschränkt auf Kleinbuchstaben und Ziffern. Alle Trigramme haben einen Count von mindestens eins.
- Es werden maximal $2 \cdot 10^6$ verschiedene Wörter in allen Trigrammen vorkommen.
- Ihre Implementierung sollte mit einer "großen" Datenmenge umgehen können (bis zu ca. $100 \cdot 10^6$ Datensätze und $100 \cdot 10^3$ Anfragen).
- Die Zeit, die ihre Implementierung benötigt, um die Datenstruktur aufzubauen, wird nicht gewertet. Das heißt, dass die Zeitmessung erst mit der ersten Anfrage beginnt. Die Dauer für die Aufbauphase (inklusive des Aufrufs von `TrigramStorage.build()`) ist jedoch auf maximal 20 Minuten beschränkt; ein Überschreiten führt zur Disqualifikation (für den aktuellen Lauf).
- Es darf kein Sourcecode anderer Gruppen oder anderer Quellen übernommen werden.
- Ebenso dürfen Sie keine nicht-elementaren Bibliotheksfunktionen verwenden. Insbesondere dürfen Methoden wie `java.util.sort` oder Klassen wie `java.util.hash` nicht verwendet werden.
- Aus diesem Grund sind imports mit * (also z.B. der Form `import java.util.*`) nicht zulässig.
- Die Einreichungen werden von uns auf einem Rechner mit der Java-Version 7 ausgeführt. Der verfügbare Heap-Speicherplatz von Java wird von uns auf 180GB begrenzt.
- Für die Gruppe, die die **schnellste korrekte** Implementierung einreicht, gibt es kleine Sachpreise und natürlich den versprochenen endlosen Ruhm!
- Teilnehmen dürfen alle Studierende der Vorlesung "Datenstukturen und Algorithmen". Der Rechtsweg ist natürlich ausgeschlossen.
- Die Abgabe kann in Gruppen von 1 bis 4 Personen erfolgen.

Update:

- Die Implementierung darf **nicht mehr als einen Thread** benutzen.
- Der Source-Code muss **vollständig** in Java vorliegen. Das bedeutet: kein JNI oder ähnliches.