Karlsruher Institut für Technologie

Programmieren – Sommersemester 2014
Forschungsgruppe Verifikation trifft Algorithmik
`http://baldur.iti.kit.edu/programmieren`
Dr. Carsten Sinz · Markus Iser

Abschlussaufgabe 1

Ausgabe: 04.08.2014 – 13:00
Abgabe: 01.09.2014 – 13:00

Allgemeine Hinweise

- Achten Sie darauf nicht zu lange Zeilen, Methoden und Dateien zu erstellen[1]
- Programmcode muss in englischer Sprache verfasst sein
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]
- Halten Sie alle whitespace Regeln ein[1]
- Halten Sie die Regeln zu Variablen-, Methoden und Paketbenennung ein und wählen Sie aussagekräftige Namen[1]
- Halten Sie die Regeln zu Javadoc Dokumentation ein[1]
- Nutzen Sie nicht das default package[1]
- Achten Sie auf die korrekte Sichtbarkeit Ihrer Klassen[1]
- Halten Sie auch alle anderen checkstyle Regeln ein[1]

Web-Suchmaschine

Die ACME Corporation betreibt zum internen Gebrauch eine Vielzahl von Internetseiten (nachfolgend „ACME-Web“ genannt), deren volles Ausmaß sie schon lange nicht mehr überblickt. Öffentliche Suchmaschinen sind den ACME-Mitarbeitern daher ein willkommenes Werkzeug, um schnell an gewünschte Informationen zu kommen. Allerdings stört sich ACME zunehmend an der durch Suchmaschinen hervorgerufenen Serverlast – mehr als 50% aller Serveranfragen können Suchmaschinen-Crawlern zugerechnet werden. ACME greift daher zu drastischen Maßnahmen: per IP-Filter verbieten sie allen gängigen Suchmaschinen-Crawlern den Zugriff auf das ACME-Web, das ohnehin uninteressant für Außenstehende ist. Doch die Mitarbeiter fordern unaufhörlich ihre Suchfunktionalität zurück, die sie so zu schätzen gelernt haben. Schließlich lenkt die Geschäftsleitung ein und bewilligt die Entwicklung einer internen Web-Suchmaschine, deren Arbeitstitel ACME.Search lautet. Ihr Aufgabe ist es, den Prototypen von ACME.Search zu entwickeln, welche aus den drei Hauptkomponenten Crawler, Indexer und Suche besteht.

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

A Crawler

Der Crawler hangelt sich von Seite zu Seite des ACME-Webs, indem ausgehende Links jeder Seite verfolgt werden. Den Inhalt jeder besuchten Seite legt der Crawler in einer geeigneten Datenstruktur im Hauptspeicher[2] ab. Aus dem Seiteninhalt extrahiert der Crawler alle ausgehenden Links. Ausgehende Links werden anschließend besucht, allerdings nur solche Links die zuvor noch nicht besucht wurden. Kein Link darf mehrmals besucht werden, um eine endlose Ausführung zu verhindern.

Zu Entwicklungszwecken wird Ihnen ein Web in Form eines Emulators zur Verfügung gestellt. Nutzen Sie die Methode `getWeb` der beigefügten Klasse `WebFactory`, um den Emulator zu instanziieren. Anschließend können Sie die `request`-Methode des Emulators verwenden, um den zu einer URL gehörenden Seiteninhalt abzurufen. Die Klasse `WebFactory` erzeugt im Ursprungszustand Instanzen der Klasse `TestWeb`. Es genügt aber, wenn die zurückgelieferten Klassen das `WebEmulator`-Interface implementieren, so können Sie auch mit eigenen `WebEmulator`-Implementierungen experimentieren. Achten Sie darauf, dass Ihr Programm nicht von der Klasse `TestWeb` abhängt und dass Sie die Signatur und Pakete der Klassen `WebEmulator` und `WebFactory` nicht verändern, da auch wir diese zum Testen verwenden.

Sie dürfen in dieser Aufgabe davon ausgehen, dass immer ein Dokument mit der URL `Start.sml` existiert, und dass Sie, wenn Sie von diesem Dokument aus anfangen zu crawlen, das gesamte Netz erreichen. URLs sind in diesem Zusammenhang einfache eindeutige Dokument*bezeichner*, die nicht notwendigerweise Protokollangaben und Servernamen einschließen.

B Simple Markup Language (SML)

Während gewöhnliche Webseiten im (X)HTML-Format vorliegen, nutzen wir für dieses Aufgabenblatt eine stark vereinfachte Variante, die wir nachfolgend Simple Markup Language (SML) nennen. Alle Seiten des ACME-Webs liegen als SML-Dokumente vor. Ein SML-Dokument enthält eine beliebige Anzahl an Links, welche in einen Text eingebettet werden können. Ein Link wird mit der Zeichenfolge `[[` eingeleitet und mit der Zeichenfolge `]]` abgeschlossen (doppelte öffnende bzw. doppelte schließende eckige Klammer). Der Text zwischen den Klammern ist eine URL mit der ein weiteres SML-Dokument aufgerufen werden kann. Weitere Schlüsselwörter bzw. Auszeichnungselemente existieren nicht. Neben den Links darf ein SML-Dokument beliebige Folgen aus lateinischen Buchstaben, Umlauten, Zahlen und Leerzeichen enthalten. Nur folgende Sonderzeichen sind erlaubt: `. , ; : - _ ? !` Abbildung B zeigt ein SML-Beispieldokument.

> Das TF-IDF-Maß (von englisch `[[Englische_Sprache.sml]]` Term Frequency und Inverse Document Frequency `[[Inverse_Dokumenthäufigkeit.sml]]`) wird im Information Retrieval `[[Information_Retrieval.sml]]` zur Beurteilung der Relevanz von Termen in Dokumenten einer Dokumentenkollektion eingesetzt.

Abbildung 1: Beispiel SML-Dokument „TF-IDF-Maß.sml“ (vgl. Wikipedia)

C Indexer

Damit die Dokumentensammlung nicht bei jeder Suchanfrage erneut durchsucht werden muss, soll ein *Invertierter Index* aufgebaut werden. Ein Invertierter Index beinhaltet alle *verschiedenen* Wörter, die innerhalb der Dokumentensammlung vorkommen. Zu jedem Wort speichert der Invertierte Index zudem eine Liste der Vorkommen (d.h. Dokumente) genau dieses Wortes. Jedes durch den Crawler entdeckte Dokument wird in diesen Index eingepflegt. Dazu wird der Text in seine Bestandteile (Worte) zerlegt und Wort für Wort dem Index hinzugefügt.

Der Text wird so in Worte aufgetrennt, dass nach jedem Leerzeichen oder Satzzeichen ein neues Wort beginnt, wobei ein Wort aus mindestens einem Buchstaben bestehen muss. Dabei müssen die Sonderzeichen zur Interpunktion ignoriert werden. Zusammengesetzte Worte mit Bindestrich werden als *ein* Wort erkannt, falls keine

[2] Sie dürfen davon ausgehen, dass die Menge aller Dokumente in den Hauptspeicher passt. Es sollen keine Dateien angelegt werden.

zusätzlichen Leerzeichen vor und/oder nach dem Strich eine Interpretation als Gedankenstrich nahelegen. Worte werden unabhängig von Ihrer Groß-/Kleinschreibung miteinander verglichen.

D Suche

ACME.Search nimmt Suchanfragen vom Benutzer über die interaktive Benutzerschnittstelle entgegen (siehe Abschnitt E) und liefert die zehn relevantesten Dokumente, absteigend sortiert nach Relevanz. Die Relevanz (ein `double`-Wert) eines Dokumentes bezüglich einer Suchanfrage wird im Folgenden besprochen.

D.1 Ranking

Der Inverse Index liefert zu einem Suchwort eine möglicherweise sehr große Menge an Dokumenten zurück, welche das Suchwort enthalten. Ranking-Verfahren ordnen jedem Dokumenten nach bestimmten Regeln einen Relevanzwert zu. Nachfolgend sind zwei Ranking-Verfahren beschrieben. ACME möchte es offen lassen, welches Ranking-Verfahren am Ende tatsächlich benutzt wird, und so soll es die Möglichkeit geben durch einfache Weise auch weitere Ranking-Methoden hinzuzufügen.

D.1.1 TF-Ranking

Beim Term-Frequency Ranking (kurz TF-Ranking) wird gezählt, wie häufig ein Suchterm t in einem Dokument d auftaucht. Je häufiger der Suchbegriff t im Dokuments enthalten ist, desto höher ist dessen Relevanz. Das Maß wird üblicherweise mit der Häufigkeit des häufigsten Wortes normalisiert.[3]

Sei $|d_t|$ die Häufigkeit des Wortes t im Dokument d und sei $|d|$ die Anzahl aller Worte des Dokuments d, dann ist die `TF`-basierte Relevanz des Dokuments d bezüglich des Suchbegriffs t wie in Gleichung 1 gegeben.

$$\texttt{TF}(t,d) = \frac{|d_t|}{\texttt{max}\{|\texttt{d}_\texttt{w}|, \texttt{w} \in \texttt{d}\}} \tag{1}$$

D.1.2 TF-IDF-Ranking

Man kann leicht einsehen, dass ein seltener Begriff wie z.B. „Programmieren-Vorlesung“ besser dafür geeignet ist ein Dokument zu charakterisieren als inflationär gebrauchte Worte wie Pronomen oder Artikel, die in praktisch allen Dokumenten der Dokumentensammlung auftauchen.

Wie wichtig ein Begriff t innerhalb der Menge aller Dokumente ist, gibt die Inverse Dokumentenhäufigkeit (IDF) an. Sei N die Anzahl aller Dokumente der Dokumentensammlung und n_t die Anzahl der Dokumente, in denen das Wort t mindestens einmal vorkommt, dann ist die Inverse Dokumentenhäufigkeit wie in Gleichung 2 gegeben. Verwenden Sie für die Berechnung des natürlichen Logarithmus die Methode `Math.log`.

$$\texttt{IDF}(t) = \texttt{ln}\left(\frac{N}{n_t}\right) \tag{2}$$

Um das Vorkommen eines seltenen Begriffes innerhalb eines Dokumentes stärker zu gewichten als das Vorkommen eines inflationär gebrauchten Begriffs, kombiniert man beim TF-IDF-Ranking die Gleichungen 1 und 2.

$$\texttt{TF-IDF}(t,d) = \texttt{TF}(t,d) \cdot \texttt{IDF}(t) \tag{3}$$

[3] Die benötigten Maße extrahiert man am besten schon während der Indexierung.

D.2 Zusammengesetzte Suchanfragen

Eine zusammengesetzte Suchanfrage ist eine Suchanfrage, die aus zwei oder mehreren Wörtern besteht. Zunächst ist undefiniert, wie mehrere Suchwörter miteinander verknüpft werden: liefert die Suche nach „programmieren vorlesung“ nur solche Dokumente, in denen sowohl *programmieren* als auch *vorlesung* vorkommt? Oder werden alle Dokumente zurückgeliefert, die entweder *programmieren*, oder *vorlesung* enthalten, aber nicht notwendigerweise beide Begriff zusammen? Die erste Verknüpfungsart wird als `Und`-Verknüpfung (Konjunktion) bezeichnet. Die zweite Verknüpfungsart wird als `Oder`-Verknüpfung (Disjunktion) bezeichnet.

Sei ρ ein beliebige Ranking-Funktion, wie in D.1 beschrieben, dann erweitern wir diese zu einer rekursiven Funktion κ, die die Komposition umfasst, wie folgt:

$$\kappa(t) = \begin{cases} \rho(t), & \text{falls } t \text{ ein einfacher Suchbegriff ist} \\ \top(\kappa(t_1), \kappa(t_2)), & \text{falls } t = \text{„AND}(t_1,\, t_2)\text{“} \\ \bot(\kappa(t_1), \kappa(t_2)), & \text{falls } t = \text{„OR}(t_1,\, t_2)\text{“} \end{cases} \tag{4}$$

Für die Kompositionsfunktionen $\top$ und $\bot$ gibt es genau wie beim Ranking verschiedene Möglichkeiten der Implementierung. Ihr Ranking-Algorithmus soll auch unterschiedliche Arten der Komposition unterstützen. Berücksichtigen Sie bei der Modellierung, dass weitere Kompositionsmethoden hinzukommen könnten. Implementieren Sie die beiden unten genannten Methoden der Komposition.

D.2.1 Komposition mit Maximum und Minimum

Hier wird für die Realisierung der Konjunktion die Minimums- und für die Realisierung der Disjunktion die Maximumsfunktion verwendet.

$$\top(a, b) = \texttt{min}\{a, b\} \tag{5}$$

$$\bot(a, b) = \texttt{max}\{a, b\} \tag{6}$$

D.2.2 Komposition mit Produkt und Summe

Hier wird für die Realisierung der Konjunktion der Produkt- und für die Realisierung der Disjunktion der Summenoperator verwendet.

$$\top(a, b) = a * b \tag{7}$$

$$\bot(a, b) = a + b \tag{8}$$

E Benutzerschnittstelle

Sämtliche Fehlermeldungen müssen mit dem Präfix „Error, “ gekennzeichnet werden.

E.1 Konfiguration

Ihr Programm soll als Konsolen-Parameter die zu verwendende Ranking- und Kompositions-Methode entgegennehmen.

Wird das Programm mit „-ranking=TFIDF“ (**default**) gestartet, so wird das Ranking mit Formel 3 durchgeführt, mit „-ranking=TF“ wird das Ranking mit Formel 1 durchgeführt. Mit „-composition=PRODSUM“ (**default**) wird die Kompositionsmethode mit Produkt und Summe ausgewählt, mit „-composition=MAXMIN“ wird die Kompositionsmethode mit Maximium und Minimum ausgewählt.

Wird keine Kompositions- bzw. Ranking-Methode angegeben, so wird die mit **default** gekennzeichnete Methode gewählt. Es dürfen keine konkurrierenden Kompositions- bzw. Ranking-Methoden angegeben werden. Wird eine ungültige Kombination von Parametern angegeben, beendet sich das Programm sofort mit einer aussagekräftigen Fehlermeldung.

E.2 Interaktive Benutzerschnittstelle

Ihr Programm nimmt nach dem Crawlen und Indexieren drei Arten von Befehlen entgegen. Benutzten Sie für alle Ein- und Ausgaben die `Terminal`-Klasse. Verändern Sie diese nicht und schieben Sie sie auch nicht in ein anderes Paket. Verwenden Sie in keinem Fall `System.in` oder `System.out`.

Wir wollen uns nicht mehr als nötig mit Ausgabe-Formaten aufhalten, daher sind die einzigen Ausgaben, die das Programm tätigt, die Ergebnisse zu den eingegebenen Befehlen.

E.2.1 Der `search`-Befehl

Gesucht wird mit „`search` *term*“, wobei *term* entweder für ein zu suchendes Wort oder für einen zusammengesetzten Ausdruck steht. Ausdrücke werden wie folgt zusammengesetzt:

term ::= *final* | `AND`(*term*, *term*) | `OR`(*term*, *term*)
final ::= (a-zA-Z0-9-_)*

Wie Sie sehen, bestehen einfache Suchworte aus Ziffern, Zahlen, Binde- und/oder Unterstrichen. Größere Terme können aber auch mittels der *binären* Operatoren `AND` und `OR` zusammengesetzt werden.

Überflüssige Leerzeichen sollen entfernt werden, außerdem sollen `String`-Vergleiche unter Nichtbeachtung der Groß- und Kleinschreibung erfolgen.

Beispielausdrücke:

- `Katzenbild` (einfaches Suchwort)
- `OR(Katze, Kater)` (einfach zusammengesetzter Term)
- `AND(Katze, OR(goldig, lieb))` (zweifach zusammengesetzter Term)

Als Ausgabe erfolgen zeilenweise die URLs der zehn besten Treffer, absteigend nach Ranking-Wert sortiert. Geben Sie am Ende jeder Zeile den Ranking-Wert des Dokuments (auf die zweite Nachkommastelle gerundet) in Klammern mit aus.[4]

Beispielausgabe:

```
Inverse_Dokumenthäufigkeit.sml (1,01)
Information_Retrieval.sml (0,20)
Englische_Sprache.sml (0,00)
```

E.2.2 Der `info`-Befehl

Die Ausgabe besteht aus zwei Zeilen. Die erste Zeile enthält Informationen über die eingelesene Dokumentmenge, (d.h. Anzahl der indizierten Worte und Dokumente), und die zweite Zeile enthält Informationen über die gewählte Konfiguration (d.h. welche Kompositions- bzw. Ranking-Methode ausgewählt ist).

[4] Die Darstellung des Dezimaltrenners ist `Locale`-abhängig und muss nicht speziell angepasst werden.

Halten Sie sich an die im Beispiel verwendete Formatierung und Reihenfolge der Ausgabe. Verwenden Sie für die Konfigurationen dieselben Bezeichner, wie sie auch in der Eingabe verwendet werden.

Beispielausgabe:

```
Documents: 3, Words: 120
Ranking: TFIDF, Composition: MAXMIN
```

E.2.3 Der `quit`-Befehl

Beendet das Programm

F Aufgabenstellung

Schreiben Sie ein Programm, das die oben genannten Spezifikationen erfüllt. Zunächst soll das vorgegebene Web mittels `WebFactory` geholt, vollständig gecrawlt und indexiert werden. Dann werden in einer Schleife Suchanfragen und weitere Befehle entgegengenommen und beantwortet.

Parsen Sie die zusammengesetzten Terme rekursiv, indem Sie den Suchterm von links nach rechts „abfrühstücken“. Überlegen Sie sich eine geeignete Datenstruktur für die geparsten Terme.

Achten Sie auf eine saubere Modellierung, so dass der Ranking-Algorithmus unabhängig von der speziellen (gerade ausgewählten) Kompositions- und Ranking-Methode arbeitet und nicht verändert werden muss, wenn eine neue Kompositions- und Ranking-Methode hinzukommt.

Laden Sie ihre Lösung nicht erst kurz vor Abgabeschluss hoch. Die öffentlichen Praktomat-Tests geben Ihnen wertvolle Hinweise zur Kompatibilität mit der vorgegebenen Spezifikation und damit auch mit den bei der Korrektur verwendeten Tests.