2. Übung zur Vorlesung "Internet-Suchmaschinen" im Wintersemester 2007/2008 – mit Musterlösungen –

Dr. Andreas Hotho, Prof. Dr. Gerd Stumme, MSc. Wi-Inf Beate Krause

15. November 2007

1 Tokenizing

1. Skizzieren Sie in Pseudocode oder einer Programmiersprache Ihrer Wahl, wie man aus einem HTML-Dokument den reinen Text extrahieren kann.

   Welche Probleme hat Ihre einfache Lösung noch?

   Eine einfache Variante in Perl (Java, andere Skriptsprachen äquivalent, nur etwas länger ;-):

   ```
   # iteriere über alle Zeilen
   while (<>) {
     # filtere alles zwischen spitzen Klammern, non-greedy!
     s/<.*?>//g;
     # gebe Resultat aus
     print;
   }
   ```

   Hier fehlt noch:

   - Auflösen von Entities (`&auml;` → ä usw.)
   - Schlägt fehl bei Tags, die über mehrere Zeilen gehen
   - ... und die Fragen aus der nächsten Aufgabe.

2. Berücksichtigt Ihr Verfahren auch

   - `ALT`-Tags von Bildern?
   - `TITLE`-Tags von Hyperlinks?
   - Keywords in `META`-Tags?
   - Kommentare und Skripte (sollen ausgeblendet werden!)

Wie aufwendig ist es, dies alles nachzurüsten?

Um diese in der Perl-Lösung von oben nachzurüsten, wäre es sinnvoll, zuerst das ganze Dokument in einen String einzulesen. Danach kann weitgehend wieder mit regulären Ausdrücken gearbeitet werden, die über mehrere Zeilen gehen können.

Problematisch wird allerdings, alle Spezialfälle abzufangen. Was ist z. B. mit den HTML-Konstrukten wie Kommentar-Trennern `<!--` und `-->`, wenn sie in Attributen oder in Skripten vorkommen, usw.?

3. Man will die oben genannten Features haben und zusätzlich auch noch Strukturinformation verarbeiten, also etwa die erste `<h1>`-Überschrift besonders gewichten, Hyperlinks extrahieren, usw.

   Finden Sie eine elegantere Möglichkeit, dies alles anzubieten, ohne von Hand einen entsprechenden Tokenizer zu bauen?

   Die eleganteste Variante ist, sich einen HTML-Parser wie z. B. JTidy zu nutze zu machen. Von diesem bekommt man das Dokument als Datenstruktur (sog. *Document Object Model, DOM*) und kann leicht auf alle Elemente, Attribute, Text-Elemente usw. gezielt zugreifen.

2 Indexstrukturen

Wir nehmen an, daß wir einen invertierten Index für folgenden Korpus gebaut haben:

- 1.000.000.000 Dokumente
- 2.000.000 Terme
- jeder Term komme im Mittel in 100.000 Dokumenten vor
- jeder Term und jeder Listeneintrag sei 16 Byte groß.

Der Index stehe als Aneinanderreihung der Terme und Dokument-Term-Gewichte auf dem Sekundärspeicher.
Weiterhin haben wir den Index auf einem RAID-Array mit

- 4 kB Blockgröße, 64 Bit breite Blocknummern
- 8 ms mittlerer Zugriffszeit
- 50 MB/s Übertragungsrate bei linearem Zugriff

1. Schätzen Sie ab, wie lange das Auffinden eines Term-Vorkommens bei linearer Suche im Mittel dauern würde.

   Gesamtgröße des Index: $2 \cdot 10^6$ Terme$\cdot 10^5$ Dokumente$\cdot 16$ Byte $= 3,2 \cdot 10^{12}$ Byte $= 3,2$ TB

Um ein Termvorkommen zu erhalten, müssen wir bei linearer Suche 1,000,000 Terme einlesen. Dies sind 10^6 Terme$\cdot 16$ Byte. Betrachten wir unseren Gesamtindex, sind dies ein Zweihunderttausendstel des Gesamtindex, denn: $3,2\,\text{TB}/(10^6\,\text{Terme}\cdot 16\,\text{Byte}) = 200,000$.

Damit wir also unseren Term bei linearer Suche finden, benötigen wir $\frac{1}{200000} \cdot 3,2\,\text{TB}/(50\,\text{MB/s}) = 0.32\,\text{s}$.

Um die 1,6 MB für den Eintrag des Terms zu lesen, sind $\frac{1,6\,\text{MB}}{50\,\text{MB/s}} = 32\,\text{ms}$ notwendig. Insgesamt brauchen wir also rund 350 ms.

Wollen wir allerdings *alle* Vorkommen aller Terme finden, müssen wir im Mittel (fast) den ganzen Index, also 3,2 TB lesen, und benötigen dazu $3,2\,\text{TB}/(50\,\text{MB/s}) = 17,8\,\text{h}$.

2. Kennen Sie eine Indexstruktur, um eine solche Suche auf dem Hintergrundspeicher zu beschleunigen? Schätzen Sie die Kosten mit einer solchen Datenstruktur ab.

   Wir können einen B-Baum benutzen, um die Suche im Index zu beschleunigen. In unserem B-Baum bestehen die Zeiger jeweils aus einem Term (16 Byte) und einer Blocknummer (8 Byte).

   Es passen also 4 kB / (16 Byte + 8 Byte) = 170 Zeiger in eine Seite; wir können also mit logarithmischer Tiefe zur Basis 170 rechnen. $\log_{170}(2 \cdot 10^6) \approx 2,8$, wir müssen also drei Indexseiten anschauen. Zusätzlich müssen wir den Eintrag für den Term lesen. Wir benötigen also 4 Plattenzugriffe und müssen $3 \cdot 4\,\text{kB} + 1 \cdot 1,6\,\text{MB}$ lesen:

   $$T = 4 \cdot 8\,\text{ms} + \frac{1,6\,\text{MB}}{50\,\text{MB/s}} = 32\,\text{ms} + 32\,\text{ms} = 64\,\text{ms}$$

   Insgesamt werden wir also etwa 64 ms benötigen, um die Vorkommen des Terms zu finden.

3 Evaluation

Ein Student soll ein Referat über den höchsten Vulkan der Welt, Ojos del Salado, schreiben. Dafür nutzt er zwei Suchmaschinen, und erhält jeweils eine Liste aus 30 URLs. Für diese erstellt er folgende Relevanzlisten (+ repräsentiert eine relevante URL, - repräsentiert eine nicht relevante URL):

$\triangle_1 = (+|+|-|+|-|-|-|+|-|-|-|+|-|-|-|-|-|-|+|-|-|-|-|-|-|-|-|+)$

$\triangle_2 = (-|+|+|-|-|-|+|-|-|+|+|-|+|-|-|-|+|-|-|+|-|-|+|-|+|-|-|-|-|-)$

- Zeichnen Sie für beide Systeme den Precision-Recall Graphen. Ermitteln sie die Precision und Recall Werte dabei in 5er-Abständen (also den 1., 5., 10. usw. Precision Wert). Gehen Sie davon aus, dass für jede Liste insgesamt 12 Dokumente relevant sind.

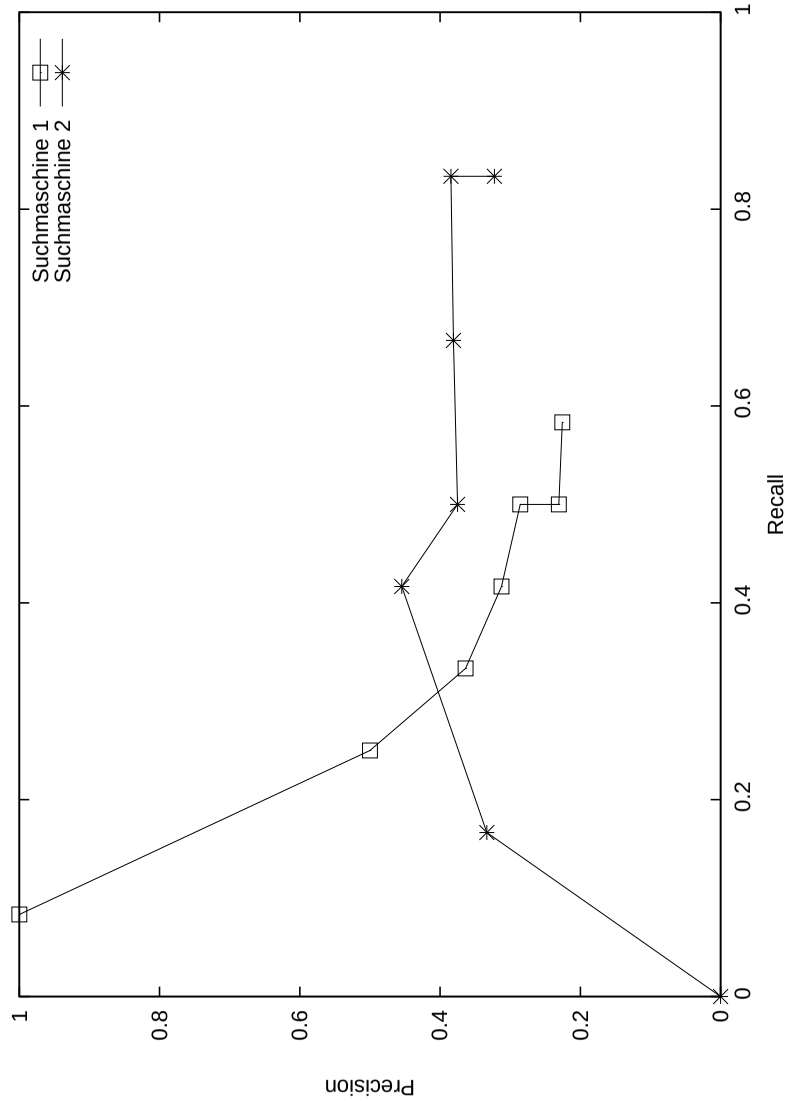

Figure 1: Recall-Precision-Diagramm für verschiedene Suchalgorithmen

- Welche Ranking-Liste ist in welchem Anwendungsszenario besser? Falls Wert auf die Precision gelegt wird, so schneidet Suchmaschine 1 besser ab, da viele relevante Dokumente in den ersten Antworten vorhanden sind. Falls dem Studenten wichtiger ist, alle möglichen relevanten Dokumente zu erhalten, schneidet System B besser ab, da sowohl Recall als auch Precision über die ganze Liste gesehen besser sind.
- Das F-Measure wird als harmonisches Mittel von Precision und Recall definiert. Welchen Vorteil hat die Benutzung des harmonischen Mittels gegenüber des arithmetischen Mittels? Beide Mittel sind 0, wenn Recall und Precision 0 sind, und 1 wenn Recall und Precision 1 sind. Allerdings ist der Verlauf des harmonischen Mittels für Recall- und Precision Werte anders: nur wenn beide Maße hohe Werte aufzeigen, nähert sich das harmonische Mittel dem Wert 1 an.

4 Praxisübung

Abgabe: 28.11.2007

Implementieren Sie einen invertierten Index mit TF-IDF-Gewichtung entsprechend dem Interface `InvertedIndex`! Die Termgewichte sollen wie in der Vorlesung skizziert normiert sein. Es gibt wieder eine Testklasse `IndexText`, mit der Sie Ihren Index ausprobieren können. Folgende Dokumente sind die am höchsten gerankten für die jeweils genannten Terme:

Term	Datei → Gewicht, ...
november	8683 → 0.5246, 3639 → 0.1749, ...
shipbuilding	1902 → 0.2623, 6541 → 0.2623, 5818 → 0.1749, ...
sugarcane	10306 → 0.2736, 11173 → 0.2736, 4630 → 0.1824, 259 → 0.1824, ...

Tip: Implementieren Sie die Postinglisten als absteigend sortierte `Collection` von `TokenOccurence`-Objekten. Es ist etwas trickreich, dazu das Interface `Comparable<TokenOccurrence>` in `TokenOccurrence` korrekt (!) zu implementieren. Lesen Sie zuerst die Java-Dokumentation zu `Comparable`!