Information Retrieval 2011, Aufgabenblatt 3, Lösungsvorschlag

Aufgabe 9 Positionelle Indexes

Nachfolgend ein Ausschnitt eines positionellen Indexes der Form „Term: doc1: position1, position2, ...; doc2: position1, position2, ... ; etc.":
angels: 2: (36,174,252,651); 4: (12,22,102,432); 7: (17);
fools: 2: (1,17,74,222); 4: (8,78,108,458); 7: (3,13,23,193);
fear: 2: (87,704,722,901); 4: (13,43,113,433); 7: (18,328,528);
in: 2: (3,37,76,444,851); 4: (10,20,110,470,500); 7: (5,15,25,195);
rush: 2: (2,66,194,321,702); 4: (9,69,149,429,569); 7: (4,14,404);
to: 2: (47,86,234,999); 4: (14,24,774,944); 7: (199,319,599,709);
tread: 2: (57,94,333); 4: (15,35,155); 7: (20,320);
where: 2: (67,124,393,1001); 4: (11,41,101,421,431); 7: (16,36,736);
1. Welche Dokumente, falls überhaupt, passen auf folgende Suchanfragen? (Ausdrücke in Anführungszeichen sind Phrasensuchen)
a. „fools rush in"
b. „fools rush in" AND „angels fear to tread"
2. Rekonstruieren Sie den Inhalt des Dokumentes 2 anhand des Indexes.
3. Erstellen Sie einen Teil des Biword Indexes, der für die Beantwortung der ersten Anfrage notwendig ist.
4. Nehmen Sie an, die Terme „in" und „to" sind als Stop-Words aus den o.g. Suchanfragen herausgefiltert worden. Wie kann ein positioneller Index in einem Suchsystem mit Stop-Word Filterung kombiniert werden?

9.1)

a: „fools rush in"; {Doc2, Doc4, Doc7}

b: „fools rush in" AND „angels fear to tread": {Doc4}

9.2)

fools rush in...fools...angels in...to...tread...rush
where...fools...in...to...fear...tread...where...angels...rush...fools...to...angels...rush...tread...where...in...angels...rush...fear...in... fear...to...where

9.3)

fools rush: 2:<(1,2)>; 4:<(8,9)>; 7: <(3,4), (13,14)>

rush in: 2: <(2,3)>; 4: <(9,10)>; 7: <(4,5),(14,15)>

9.4)

Es gibt zwei Arten zum Kombinieren des positionellen Index mit Stop-Word-Filterung: Skip Modus und Position-Increment-Modus:

1) Skip Modus: Bei der Indexierung werden die Stopwörter komplett ignoriert und

die Positionen der anderen Terme werden entsprechend angepasst. Im Skip Modus ist eine Suchanfrage wie etwa „angels fear to tread“ äquivalent zur Anfrage „angels fear tread“ und es wird das Dokument 4 gefunden.

2) Position-Increment-Modus: Bei der Indexierung werden die Stopwörter nicht gespeichert aber gleichzeitig erhöhen sich die Positionen der Terme, die nach den Stopwörtern stehen. Damit werden die Phrasen gematcht, deren Terme vor und nach den Stopwörtern stehen. Im Position-Increment-Modus ist die Suchanfrage „angels fear to tread“ äquivalent zu „angels fear * tread“ und es werden die Dokumente {Doc4, Doc7} gefunden.

Zur weiteren Verdeutlichung Zitat aus Lucene Implementierung:

Suppressing Stop Word Indexing

Skip to end of metadata

-
- Added by Cassandra Targett, last edited by Drew Wheeler on Mar 15, 2011 (view change)

Go to start of metadata

By default, LucidWorks Enterprise indexes all stop words. Modern data storage is very cheap and even the simplest of stop words provide additional context that boosts relevancy and enables more precise queries. By default, the Lucid query parser eliminates stop words from basic queries, including them only when they are used in quoted phrases, or when a query term list consists only of stop words. In addition, the Lucid query parser uses query stop words to construct relevancy boosting phrase terms (bigram and trigram phrases) to supplement the basic query. Still, there may be applications and environments where the choice is to suppress the indexing of stop words. Although this is not the preferred choice, it is supported by the Lucid query parser.

There are two modes for suppressing stop word indexing:

1. **Skip mode**: Completely ignore or skip them, as if they were not present.
2. **Position increment mode**: Do not store them in the index, but increment the position counter so as to leave a hole at the position of each stop word.

When skip mode is selected, the query parser will ignore or skip stop words in quoted phrases.

When position increment mode is selected, the query parser will also skip each stop word, but will increment the position of the next term in the phrase so as to allow any term to match between the previous term and the next term after the stop word. This will allow for more precise query matching than the first mode where stop words are simply discarded.

Examples

Given these mini documents:

- Doc #1: Buy the time for the test.
- Doc #2: Buy more time for the test.
- Doc #3: Buy time for test.

A query of `Buy the time` regardless of the stop word indexing mode will be equivalent to `Buy AND time` and match all three documents.

A query of `"buy the time"` in normal indexing mode will match exactly that phrase and match only the first document. In skip mode it is equivalent to `"buy time"` and will match the first and third documents. In position increment mode the query is equivalent to `"buy * time"` which is not a valid query format but indicates that `"time"` will match the second word after `"buy"` regardless of the intervening word. This will match the first and second documents, but not the third document.

Aufgabe 10
Skip-Lists
Gegeben sei eine Suchanfrage bestehend aus zwei Wörtern.
Für den einen Term besteht die Posting-Liste aus folgenden 16 Einträgen:
(4, 6, 10, 12, 14, 16, 18, 20, 22, 32, 47, 81, 120, 122, 157, 180)
Für den anderen Term besteht die Posting-Liste nur aus einem Eintrag:

(47)
Bestimmen Sie die Anzahl notwendiger Vergleichsoperationen für
1. Herkömmliche Posting-Listen
2. Posting-Listen mit Skip-Pointers und einer Skip-Länge von Wurzel P. Begründen Sie Ihre Antwort.

1. Herkömmliche Posting-Listen: Vergleichsoperationen = 11.

Nach dem Algorithmus der INTERSECT(p1,p2) ist die Laufzeit linear O(m+n) m für die Länge der Liste p1, n für die Liste p2. Jeder Eintrag von der Liste 1 wird verglichen. Bis der Eintrag 47 gefunden wird, ist 11 mal eine Vergleichsoperation durchgeführt.

2. Posting-Listen mit Skip- Pointern: Vergleichsoperationen = 6

Weil die Länge von Skip-Pointer Wurzel aus p = Wurzel aus 16 = 4 für die längere Liste und 1 für die kurze Liste ist, werden die folgenden Vergleiche angestellt: 1. 4 & 47 → 2. 14 & 47 → 3. 22& 47 → 4. 120&47 → 5. 32 & 47 → 6. 47 & 47

Aufgabe 11
Wildcard Query

Gegeben sei die Wildcard Suchanfrage Sh*sp*re (Shakespeare).
a) Erstellen Sie für diese Query, die Anfragen an einen Bigram Index.
b) Geben Sie die Anfrage bei Google ein. Wie erklären wir uns das Ergebnis?

a) $s AND sh AND sp AND re AND e$

b) In „www.google.de“ werden ungefähr 1.340.000.000 Ergebnisse für die Anfrage Sh*sp*re gefunden, darin selten um Shakespeare geht.

Aufgabe 12
Levenshtein-Distanz
Erklären Sie was die Levenshtein-Distanz ist und zu welchem Zweck man sie berechnet? Berechnen Sie die Levenshtein-Distanz zwischen den Wörtern "glue" und "bleu". Stellen Sie dazu die Matrix auf und nehmen Sie folgende Kosten an: match 0 substitute: 1 insert, delete: 2

1. Die Levenshtein-Distanz zwischen zwei Zeichenketten ist die minimale Anzahl von Einfüge-, Lösch- und Ersetz-Operationen um die erste Zeichenkette in die zweite umzuwandeln.
2. In der Praxis dient die Levenshtein-Distanz der Berechnung von Wortähnlichkeiten beispielsweise zur Rechtschreibprüfung oder bei der Duplikaterkennung.

		B	L	E	U
	0	2	4	6	8
G	2	1	3	5	7
L	4	3	1	3	5
U	6	5	3	2	3
E	8	7	5	3	3

Aufgabe 13
Matrizenprodukt

Schauen Sie Definition für die Multiplikation von Matrizen nach und berechnen Sie folgende Produkte:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 3 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 2 \\ 1 & 3 & 3 \end{pmatrix} = \begin{pmatrix} 6 & 6 & 9 \\ 6 & 9 & 12 \\ 5 & 9 & 12 \end{pmatrix}$$

b) Funktioniert nicht, AB ist nur definiert wenn B so viele Zeilen, wie A Spalten hat. Das Ergebnis hat dann so viele Zeilen wie A und so viele Spalten wie B

$$\begin{pmatrix} 1 & 2 & 1 & 3 \\ 2 & 1 & 2 & 0 \\ 3 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 1 \end{pmatrix}$$

Das Produkt einer mxn Matrix und einer nxl Matrix ergibt eine mxl Matrix.