Inverted Indexes: Alternative Queries

Yufei Tao

KAIST

April 2, 2013

Remember that our discussion of inverted indexes so far aims at accelerating a specific type of queries (see the slides of an earlier lecture for the query definition). This lecture will discuss two other types of common queries. As we will see, one of them requires a more sophisticated version of the inverted index.

Type 1: Conjunctive Queries

Let S be a set of documents $D_1, ..., D_n$. Given a set Q of query terms $\{t_1, ..., t_m\}$, we want to report the k documents with the largest scores, among those documents $D \in S$ such that D contains all the terms in Q.

The score of a document D is defined as:

$$score(D, Q) = A(D) \cdot \sum_{t_j \in Q} tf(D, t_j) \cdot idf(t_j)^2$$

where

$$A(D) = rank(D)/|p|$$

where p is the point converted from D.

See the next slide for an example.

Example (Excerpted from [Zobel and Moffat, 2006]

Suppose that our document collection is:

document ID	content
1	the old night keeper keeps the keep in the town
2	in the big old gown in the big old house
3	the house in the town had the big old keep
4	where the old night keeper never did sleep
5	the night keeper keeps the keep in the night
6	and keeps in the dark and sleeps in the light

If $Q = \{\text{in, town}\}$, then only documents 1 and 3 may be returned, because none of the other documents contains both terms in Q.

Conjunctive queries are supported by an id-sorted inverted index using the following algorithm (assume that $Q = \{t_1, ..., t_m\}$):

algorithm conjunctive-query(Q)
1. $count \leftarrow 0$, $lastid \leftarrow -1$, and $s \leftarrow 0$
2. **while** none of $list(t_1), ..., list(t_m)$ has been exhausted
3. let $(i, tf(D_i, t_j))$ be the pair with the smallest document id i among all the pairs in $list(t_1), ..., list(t_m)$ that have not been examined
4. **if** $lastid \neq i$ **then**
5. $count \leftarrow 0$, $s \leftarrow 0$, and $lastid \leftarrow i$
6. **else**
7. $count++$ and $s \leftarrow s + tf(D_i, t_j) \cdot idf(t_j)^2$
8. **if** $count = m$ **then**
9. $score(D_i, Q) \leftarrow s$

/* at this point, $(i, tf(D_i, t_j))$ is said to have been examined */
10. **for** each $D \in S$
11. $score(D, Q) \leftarrow score(D, Q)/A(D)$
12. sort the documents in S by score
13. **return** the k documents with the highest scores

Example (Excerpted from [Zobel and Moffat, 2006]

term w	inverted list for w
and	(6, 2)
big	(2, 2), (3, 1)
dark	(6, 1)
did	(4, 1)
gown	(2, 1)
had	(3, 1)
house	(2, 1), (3, 1)
in	(1, 1), (2, 2), (3, 1), (5, 1), (6, 2)
keep	(1, 1), (3, 1), (5, 1)
keeper	(1, 1), (4, 1), (5, 1)
keeps	(1, 1), (5, 1), (6, 1)
light	(6, 1)
never	(4, 1)
night	(1, 1), (4, 1), (5, 2)
old	(1, 1), (2, 2), (3, 1), (4, 1)
sleep	(4, 1)
sleeps	(6, 1)
the	(1, 3), (2, 2), (3, 3), (4, 1), (5, 3), (6, 2)
town	(1, 1), (3, 1)
where	(4, 1)

To answer query $Q = \{in, town\}$, only the red pairs are examined.

Type 2: Phrase Queries

Let S be a set of documents $D_1, ..., D_n$, each of which is a sequence of terms. Given a sequence Q of terms, a query returns all the document ids i such that Q is a subsequence of D_i. Formally, let $Q = (t_1, t_2, ..., t_m)$ and $D_i = (w_1, ..., w_x)$. Then, there exists a $j \in [1, x]$ such that $t_1 = w_j$, $t_2 = w_{j+1}$, ..., $t_m = w_{j+m-1}$.

For instance, on the document collection in Slide 4, if $Q =$ "the night keeper", only document 5 may be returned. Note that "the night keeper" is a subsequence of "the night keeper keeps the keep in the night".

None of the inverted indexes we have seen is able to support phrase queries efficiently. Intuitively, this is because those indexes do not have positional information regarding where a term appears in a document.

Next, we will discuss word-level inverted indexes that contain additional features for accelerating phrase queries.

Inverted Index

A word-level inverted index consists of:

- For every term w in $DICT$, the value of $idf(w)$.
- For every term w in $DICT$, an inverted list, denoted as $list(w)$, which contains a tuple

$$(i, f, p_1, p_2, ..., p_f)$$

for every document D_i that contains w, where $f = tf(D_i, w)$, and p_j $(1 \leq j \leq f)$ indicates that the j-th term of D_i is w.

Example

The word-level inverted index for the example in Slide 4:

term w	inverted list for w
and	(6, 2, 1, 6)
big	(2, 2, 3, 8), (3, 1, 8)
dark	(6, 1, 5)
did	(4, 1, 7)
gown	(2, 1, 5)
had	(3, 1, 6)
house	(2, 1, 10), (3, 1, 2)
in	(1, 1, 8), (2, 2, 1, 6), (3, 1, 3), (5, 1, 7), (6, 2, 3, 8)
keep	(1, 1, 7), (3, 1, 10), (5, 1, 6)
keeper	(1, 1, 4), (4, 1, 5), (5, 1, 3)
keeps	(1, 1, 5), (5, 1, 4), (6, 1, 2)
light	(6, 1, 10)
never	(4, 1, 6)
night	(1, 1, 3), (4, 1, 4), (5, 2, 2, 9)
old	(1, 1, 2), (2, 2, 4, 9), (3, 1, 8), (4, 1, 3)
sleep	(4, 1, 8)
sleeps	(6, 1, 7)
the	(1, 3, 1, 6, 9), (2, 2, 2, 7), (3, 3, 1, 4, 7), (4, 1, 2), (5, 3, 1, 5, 8), (6, 2, 4, 9)
town	(1, 1, 10), (3, 1, 5)
where	(4, 1, 1)

A word-level inverted index provides all the information needed to answer a phrase query. For example, from tuple (5, 3, 1, 5, 8) of *list*(the), tuple (5, 2, 2, 9) of *list*(night), and tuple (5, 1, 3) of *list*(keeper), we know that there is a subsequence "the night keeper" starting at position 1 of document 5.

Think

- How would you design an algorithm to answer a phrase query using a word-level inverted index?
- How would you compress a word-level inverted index?