Data-analysis and Retrieval
Boolean retrieval, posting lists and dictionaries

Hans Philippi
(based on the slides from the Stanford course on IR)

April 26, 2023

Basics of text searching

1. *Collection*: fixed set of documents
2. Goal: retrieve documents that are relevant to the user's information need
3. Practice: user's information need is expressed by one or more search terms
4. Example: you want to book a room in a Hilton hotel for a trip to Paris

Information need?

paris hilton book

Web Afbeeldingen Maps Shopping Meer Zoekhulpmiddelen

Ongeveer 50.900.000 resultaten (0,28 seconden)

Tip: Alleen in het **Nederlands** zoeken. U kunt uw zoektaal instellen in de Voorkeuren

Confessions of an Heiress: A Tongue-in-Chic Peek Behind the Pose ...
www.amazon.com › ... › Fashion › Models ▾ Vertaal deze pagina
Paris Hilton is exactly what I thought she was, a spoiled little daddy's girl who thinks everyone should be like her, and it shows through in this **book**.

Afbeeldingen van **paris hilton book** - Afbeeldingen melden

Book:Paris Hilton - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Book:Paris_Hilton** ▾ Vertaal deze pagina
This is a Wikipedia **book**, a collection of Wikipedia articles that can be easily saved, rendered electronically, and ordered as a printed **book**. For information and ...

Basics

Quality measures for retrieval

1. *Precision*: fraction of retrieved docs that are relevant to user's information need (also called *selectivity*)
2. *Recall*: fraction of relevant docs in collection that are retrieved (also called *sensitivity*)

Examples of collections

WestLaw (http://en.wikipedia.org/wiki/Westlaw)

1. Largest commercial legal search service (started 1975; ranking added 1992)
2. Tens of terabytes of data; 700,000 users
3. Majority of users still use boolean queries
4. Example query:
    - What is the *statute* of *limitations* in cases involving the *federal tort claims act*?
    - LIMIT! /3 STATUTE /S FEDERAL /2 TORT /3 CLAIM (! = trailing wildcard, /3 = within 3 words, /S = in same sentence)

Collections for research purposes

RCV1, RCV2 (Reuters Corpus Volume 1, 2)

1. In 2000 Reuters released a corpus of Reuters News stories for use in research and development of natural language processing, information retrieval or machine learning
2. RCV1 covers 800,000 news articles in English (2.5 GB)
3. RCV2 covers 487,000 articles in thirteen languages
4. More recently: Reuters-21578 for text categorization

Boolean retrieval

1. Basic model for IR
2. Matching of keywords, using logical connectives: AND, OR, NOT and brackets
3. Still used, e.g. in library catalogs

Boolean retrieval

1. Which plays of Shakespeare contain the words **Brutus AND Caesar** but **NOT Calpurnia**?
2. One could *grep* all of Shakespeare's plays for **Brutus** and **Caesar**, then strip out plays containing **Calpurnia** . . .
3. . . . but smarter approaches may be ahead

Boolean retrieval: term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus* AND *Caesar* BUT NOT *Calpurnia

1 if play contains word, 0 otherwise

1. We have a 0/1 vector for each term
2. To answer query: apply a bitwise AND to the vectors for Brutus, Caesar and Calpurnia (complemented)
3. 110100 AND 110111 AND 101111 = 100100

Indexing: term-document incidence matrix?

Can we use the term-document incidence matrix for indexing purposes?

Some typical parameters:

1. number of documents: thousands (libraries) to billions (www)
2. number of terms per document: possibly several thousands
3. number of terms in a language (English, Dutch): tens of thousands (note that the web is multilingual)
4. on average 6 bytes/word

For the web, we have the following orders of magnitude:

1. 10^{10} for the number of web sites
2. roughly 10^{11} to 10^{12} for the number of web pages

Indexing: dictionary and postings lists

- *sparse matrix* approach
- documents are identified by a unique number: the *docID*
- terms are organized in a *dictionary*, supporting quick searching
- each term has a *postings list*: an ordered list of docs containing this term

Calpurnia $\Longrightarrow$	2	31	45	101	112	154	181	...
Brutus $\Longrightarrow$	1	2	4	11	31	45	173	...
Caesar $\Longrightarrow$	1	2	4	5	6	16	45	...

$\uparrow$ *Dictionary* $\uparrow$ *Postings lists*

Implementation of dictionary and postings lists

As always: optimality depends on read - update ratio.

Internal memory, static situation:

- hash table or tree like structure for dictionary
- arrays for postings lists: good cache behaviour [1]

Internal memory, dynamic situation:

- hash table or tree like structure for dictionary
- linked lists for postings lists

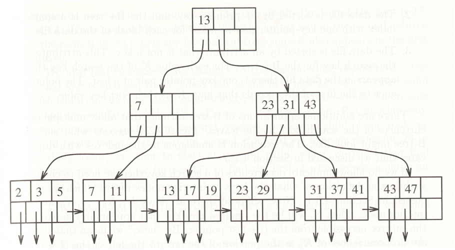
External memory:

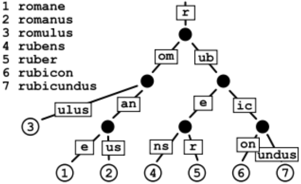
- tree like structure or hash table for dictionary
- linked lists (block structure) for postings lists

General observation: hash table does not support range queries

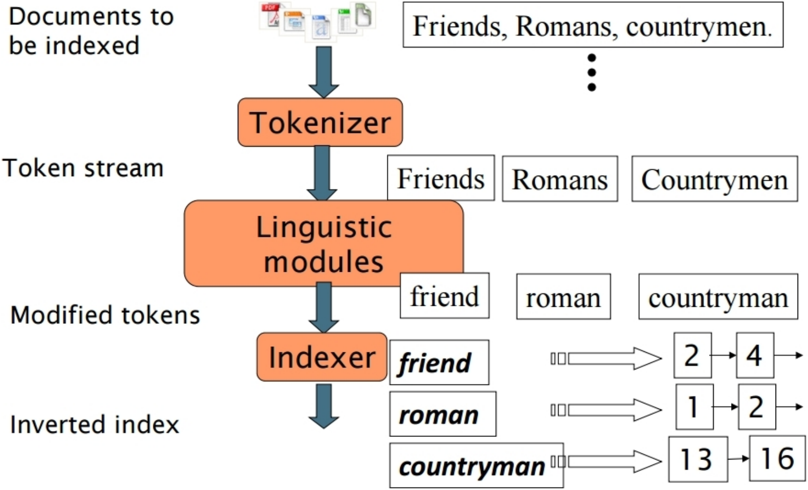
[1] MSc thesis Matthijs Meulenbrug (Mininova)

Tree like structures: B-tree and Trie (prefix tree)





Indexing process



Boolean query processing

Query = $term_1$ AND $term_2$

1. locate postings list p_1 for $term_1$
2. locate postings list p_2 for $term_2$
3. calculate the intersection of p_1 and p_2 by list merging

$term_1 \Longrightarrow$	1	3	7	**11**	37	**44**	**58**	112	...
$term_2 \Longrightarrow$	2	4	**11**	25	**44**	54	55	**58**	...

Boolean query processing: list merging

INPUT: postings lists p_1 and p_2
OUTPUT: a sorted list representing the intersection of p_1 and p_2
METHOD:

```
result = empty list;
while not (IsEmpty(p1) or IsEmpty(p2)) {
    if (docID(p1) == docID(p2))
    then {
        append(result, docID(p1));
        p1 = next(p1); p2 = next(p2);
    } else if (docID(p1) < docID(p2))
    then p1 = next(p1);
    else p2 = next(p2);
}
```

INTERMEZZO: Boolean query processing

Query $= term_1$ AND NOT $term_2$

1. locate postings list p_1 for $term_1$
2. locate postings list p_2 for $term_2$
3. ?

$p_1 \Longrightarrow$	**1**	**3**	**7**	11	**37**	44	58	**112**	...
$p_2 \Longrightarrow$	2	4	11	25	44	54	55	58	...

INTERMEZZO: Boolean query optimization

Query $= term_1$ AND $term_2$ AND ... AND $term_n$

- How do we process this query?

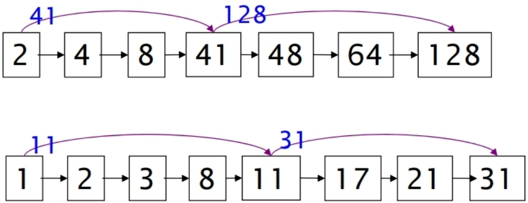
INTERMEZZO: Boolean query optimization

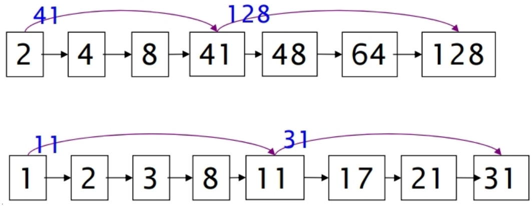
Query $= term_1$ AND $term_2$ AND ... AND $term_n$

- How many possibilities do we have?
- Analogy with join order problem in database query processing
- Heuristic?

Boolean query processing: skip pointers

Skip pointers may speed up merge process

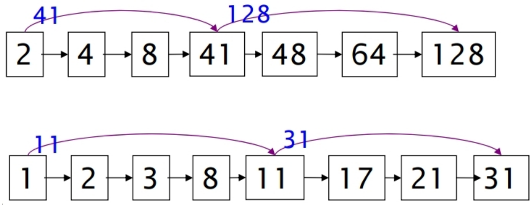




… but what are suitable skip spans?

- many skip pointers: …
- less skip pointers: …

Boolean query processing: skip pointers



... but what are suitable skip spans?

- many skip pointers: more comparisons, more frequent skips, higher memory cost
- less skip pointers: less comparisons, less frequent skips, longer jumps, lower memory cost
- rule of thumb: $\sqrt{n}$ skip pointers for $n =$ length of posting list

INTERMEZZO: Boolean query optimization

Query $= term_1$ AND $term_2$ AND $term_3$

Options:

- merge p_1 with p_2, and merge the result with p_3
- two alternatives by permutation
- do a three-way-merge of p_1, p_2 and p_3

Question:
which approach takes most advantage of skip pointers?

Phrase queries

Make a distinction between:

Q1 = "fight" AND "club"

Q2 = "fight club"

How do we support juxtaposition of terms?

Phrase queries

How do we support juxtaposition of terms?

Solution 1: biword index

Disadvantages:

- index size quadratic
- how do we support juxtaposition of three or more terms?

Phrase queries

How do we support juxtaposition of terms?

Solution 1: biword index

Disadvantages:

- index size quadratic
- how do we support juxtaposition of three or more terms?

Solution 2: positional index

Positional index

For each term, we also register the position(s) of the term in each document, where a document is regarded to be an array of tokens.

So, for each term *myterm*, we have the following entry in the index:

$<$ *myterm*: nr of docs containing *myterm*;
    doc1: position1, position2, ... ;
    doc2: position1, position2, ... ;
    ...
$>$

Positional index

Example:

$<$*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367;
... $>$

Which of the docs could contain:
"to be or not to be"

Wild-card queries

Query: *w*rd*

matches *word*, *weird* and *wild-card*

Wild-card queries may put a heavy load on query processing

Wild-card query processing using B-tree

Case 1: prefix known

Query = *pre**

- find all terms between *pre* and *prf*
- B-tree supports range queries very well

Wild-card query processing using B-tree

Case 2: suffix known

Query = **post*

- ?

Wild-card query processing using B-tree

Case 2: suffix known

Query = **post*

- maintain a second B-tree with inverted terms
- find all terms between *tsop* and *tsoq*

Wild-card query processing

Case 3: general form

Query = *pre*post*

- Option 1: intersection of results from *pre** and **post*
- Option 2: permuterm index

Wild-card query processing: permuterm index

For a term *hello*, add $\$$ to the end of the term, and create entries for each rotation of the term. All these entries are connected to the posting list of the term *hello*.

- *hello$*
- *ello$h*
- *llo$he*
- *lo$hel*
- *o$hell*

For a query = *he*o*,
we add $\$$ and rotate the term until ...

Wild-card query processing: permuterm index

For a term *hello*, add *$* to the end of the term, and create entries for each rotation of the term. All these entries are connected to the posting list of the term *hello*.

- *hello$*
- *ello$h*
- *llo$he*
- *lo$hel*
- *o$hell*

For a query = *he*o*, we add *$* and rotate the term until the *** is at the end of the query string: query = *o$he**.
Finally, notice that *o$he** has a prefix match with *o$hell*.

Wild-card query processing: k-grams

- Note that k-grams can also be used to deal with the wild-card problem
- Example: entries in search tree (k=3) pointing to *viraal*
  - vir
  - ira
  - raa
  - aal
- Determination of k requires tuning
- We will deal extensively with k-grams within the context of biological sequence alignment

References

Manning:

- chapter 1
- chapter 2.3, 2.4; the chapters on language issues are recommended as background reading
- chapter 3 - 3.2

"-" means: up to and including