

Information Retrieval using the Boolean Model

Adapted by Alberto Simões
ambs@di.uminho.pt

Original slides by Hinrich Schütze, Universität Stuttgart

April 19, 2008

- Queries are Boolean expressions, e.g., **Caesar AND Brutus**
- The search engine returns all documents that satisfy the Boolean expression
- Does Google use the Boolean model?

- Which plays of Shakespeare contain the words Brutus **AND** Caesar **BUT NOT** Calpurnia?
- One could grep all of Shakespeare's play for Brutus and Caesar, then strip out lines containing Calpurnia?
 - slow (for large corpora);
 - **NOT** Calpurnia is non-trivial;
 - Other operations (e.g., find the word Romans near Countrymen) not feasible;
 - Ranked retrieval (best documents to return).

Example

Brutus AND Caesar but NOT Calpurnia

	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

1 if play contains word, 0 otherwise



- So we have a 0/1 vector for each term.
- To answer query: take the vector for Brutus, Caesar and Calpurnia (complemented) and perform a bitwise AND.
- $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$.



- Antony and Cleopatra, Act III, Scene ii

*Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.*

- Hamlet, Act III, Scene ii

*Lord Polonius: I did enact Julius Caesar I was killed i'the
Capitol; Brutus killed me.*



- Consider $N = 1M$ documents, each with about $1K$ terms.
- Average $6\text{bytes}/\text{term}$ including spaces and punctuation:
 - $6GB$ of data in the documents.
- Say there are $m = 500K$ **distinct** terms among these.



- $500K \times 1M$ matrix has half-a-trillion 0's and 1's;
- But it has no more than one billion 1's.
 - matrix is extremely sparse. (but why?)
- What's a better representation?
 - We only record the 1 positions.



- For each term T , we must store a list of all documents that contain T .
- Do we use an array or a list for this?

Brutus \Rightarrow

2	4	8	16	32	64	128	
---	---	---	----	----	----	-----	--

Calpurnia \Rightarrow

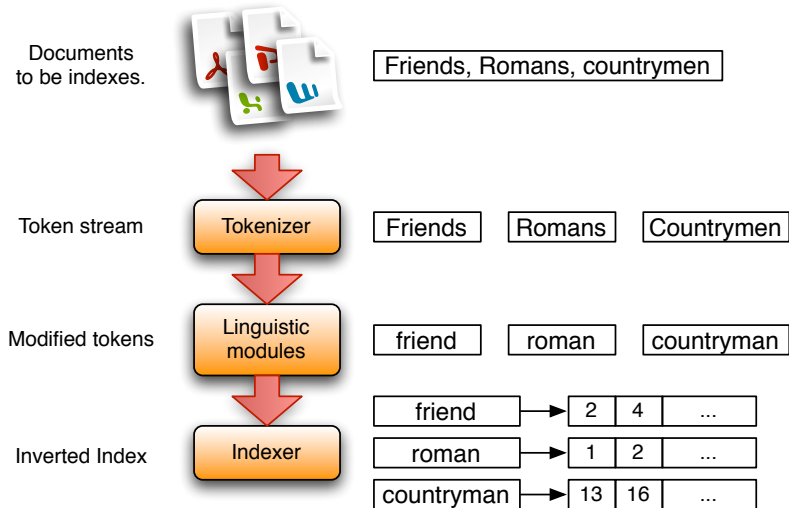
1	2	3	5	8	13	21	34
---	---	---	---	---	----	----	----

Caesar \Rightarrow

13	16						
----	----	--	--	--	--	--	--

Updating...

What happens if the word **Caesar** is added to document 14?





- Sequence of (modified token, document ID) pairs.

Doc1: I did enact
Julius Caesar. I was
Killed I' the Capitol;
Brutus killed me.

+

Doc2: So let it be
with Caesar. The
noble Brutus hath
told you Caesar was
ambitious.



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
I'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



- Sort by terms (Core indexing step)

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
I'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	2
I	1
I	1
I'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



- Multiple term entries in a single document are merged.
- Frequency information is added (why frequency?)

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	2
I	1
I	1
I'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2



Term	Doc #	Term Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
I'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

term	docID	freq	term	coll. freq.	→	postings lists
ambitious	2	1	ambitious	1	→	2
be	2	1	be	1	→	2
brutus	1	1	brutus	2	→	1 → 2
brutus	2	1	brutus	2	→	1 → 2
capitol	1	1	capitol	1	→	1
caesar	1	1	caesar	3	→	1 → 2
caesar	2	2	did	1	→	1
did	1	1	enact	1	→	1
enact	1	1	hath	1	→	2
hath	2	1	I	2	→	1
I	1	2	i'	1	→	1
i'	1	1	it	1	→	2
it	2	1	julius	1	→	1
julius	1	1	killed	2	→	1
killed	1	2	let	1	→	2
let	2	1	me	1	→	1
me	1	1	noble	1	→	2
noble	2	1	so	1	→	2
so	2	1	the	2	→	1 → 2
the	1	1	told	1	→	2
the	2	1	you	1	→	2
told	2	1	was	2	→	1 → 2
you	2	1	with	1	→	2
was	1	1				
was	2	1				
with	2	1				



- The result is split into a **Dictionary** file and a **Postings** file.

Term	Doc #	TermF
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
I'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1
...



Term	#Docs	Col.F
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
I'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	1	1
the	1	1
told	1	1
you	1	1
was	2	2
with	1	1
...

Doc #	TermF
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
...	...

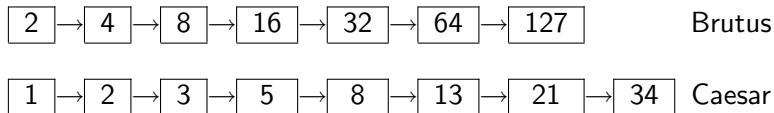
Where do we pay in storage?

Store terms just once, using pointers between tables.

Term	#Docs	Col.F	→	Doc #	TermF
ambitious	1	1	→	2	1
be	1	1	→	2	1
brutus	2	2	→	1	1
capitol	1	1	→	2	1
caesar	2	3	→	1	1
did	1	1	→	1	1
enact	1	1	→	2	2
hath	1	1	→	1	1
I	1	2	→	1	1
I'	1	1	→	2	1
it	1	1	→	1	2
julius	1	1	→	1	1
killed	1	2	→	2	1
let	1	1	→	1	1
me	1	1	→	1	2
noble	1	1	→	2	1
so	1	1	→	1	1
the	1	1	→	2	1
the	1	1	→	1	1
told	1	1	→	2	1
you	1	1	→	2	1
was	2	2	→	2	1
with	1	1	→	1	1
...	→	2	1
			→	2	1
			→

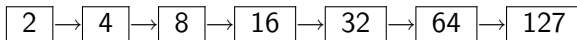
- How do we process a query?

- Consider processing the query:
Brutus AND Caesar
 - Locate **Brutus** in the Dictionary;
 - Retrieve its postings.
 - Locate **Caesar** in the Dictionary;
 - Retrieve its postings.
 - “Merge” the two postings:

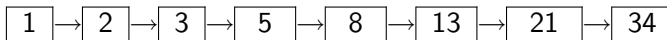




- Walt through the two postings simultaneously, in time linear in the total number of posting entries

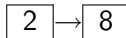


Brutus



Caesar

Results in



Notes:

- If the list lengths are x and y , the merge takes $O(x + y)$.
- Crucial:** postings sorted by document ID.



Intersecting (“merging”) two postings lists

Merge(p, q)

```
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p \neq \text{nil}$  and  $q \neq \text{nil}$ 
3  do if  $\text{docID}[p] = \text{docID}[q]$ 
4      then Add(answer,  $\text{docID}[p]$ )
5      else if  $\text{docID}[p] < \text{docID}[q]$ 
6          then  $p \leftarrow \text{next}[p]$ 
7          else  $q \leftarrow \text{next}[q]$ 
8  return answer
```

- The Boolean Retrieval model is being able to ask a query that is a Boolean expression:
 - Boolean Queries are queries using AND, OR and NOT to join query terms
 - Views each document as a **set** of words
 - Is precise: document matches condition or not
- Primary commercial retrieval tool for 3 decades.
- Professional searches (e.g. lawyers) still like Boolean queries:
 - You know exactly what you're getting.



- What is the best order for query processing?
- Consider a query that is an AND of t terms.
- For each of the t terms, get its postings, then AND them together.

Brutus	\Rightarrow	2	4	8	16	32	64	128	
--------	---------------	---	---	---	----	----	----	-----	--

Calpurnia	\Rightarrow	1	2	3	5	8	13	21	34
-----------	---------------	---	---	---	---	---	----	----	----

Caesar	\Rightarrow	13	16						
--------	---------------	----	----	--	--	--	--	--	--

Query:

Brutus **AND** Calpurnia **AND** Caesar

- Process in order of increasing freq:
 - start with smallest set, then keep cutting further.
This is why we kept frequency in dictionary!!

Brutus	⇒	2	4	8	16	32	64	128	
--------	---	---	---	---	----	----	----	-----	--

Calpurnia	⇒	1	2	3	5	8	13	21	34
-----------	---	---	---	---	---	---	----	----	----

Caesar	⇒	13	16						
--------	---	----	----	--	--	--	--	--	--

Query:

Execute the query as:

(Caesar **AND** Brutus) **AND** Calpurnia



Merge($\langle t_i \rangle$)

```
1  terms  $\leftarrow$  SortByFreq( $\langle t_i \rangle$ )
2  result  $\leftarrow$  postings[first[terms]]
3  terms  $\leftarrow$  rest[terms]
4  while terms  $\neq$  nil and result  $\neq$  nil
5  do list  $\leftarrow$  postings[first[terms]]
6     terms  $\leftarrow$  rest[terms]
7     MergeInPlace(result, list)
8  return result
```


- (madding **OR** crowd) **AND** (ignoble OR strife)
- Get freq's for all terms.
- Estimate the size of each OR by the sum of its freq's (conservative).
- Process in increasing order of OR sizes.



- What about phrases?
 - Stanford University
- Proximity: Find Gates **NEAR** Microsoft
 - Need index to capture position information in documents.
- Zones in documents: Find document with
(author = Ullman) **AND** (text contains automata)



- Boolean queries give inclusion or exclusion of documents;
- Often we want to rank/group results:
 - Need to measure proximity from query to each document;
 - Need to decide whether documents presented to user are singletons, or a group of documents covering various aspects of the query.

- Introduction to Information Retrieval, chapter 1;
- Managing Gigabytes, Chapter 3.2;
- Modern Information Retrieval, Chapter 8.2;