The Boolean Model ~1955

The *boolean model* is the first, most criticized, and (until a few years ago) commercially more widespread, model of IR. Its functionalities can often be found in the *Advanced Search* windows of many search engines.

- A document is represented by means of an *and* of index terms;
  d_1=(and jazz saxophone Coltrane)
- A query is represented by a combination, obtained through {and,or,not} of index terms belonging to a controlled vocabulary;
  q_1=(and jazz (or saxophone clarinet) (or Parker Coltrane))
- The matching function is $RSV(d_1,q_1)=1$ if q_1 is a *logical consequence* of d_1 according to Boolean logic.

Boolean queries: Exact match

- Boolean Queries are queries using *AND, OR* and *NOT* together with 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 searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting.

Closed Word Assumption

There is an implicit *Closed World Assumption* in the interpretation of the document representations:

The absence of term t in the representation of d is equivalent to the presence of (not t) in the same representation.

Given document

d_1 = (and jazz saxophone Coltrane)

and query

q_2 = (and jazz (not Parker))

then $RSV(d_1,q_2)=1$ even if ($d_1 \not\models q_2$)

This means that the true definition of $RSV(d_i,q_j) \in \{0,1\}$ is

$RSV(d_i,t) = 1$, if $t \in IREP(d_i)$, 0 otherwise

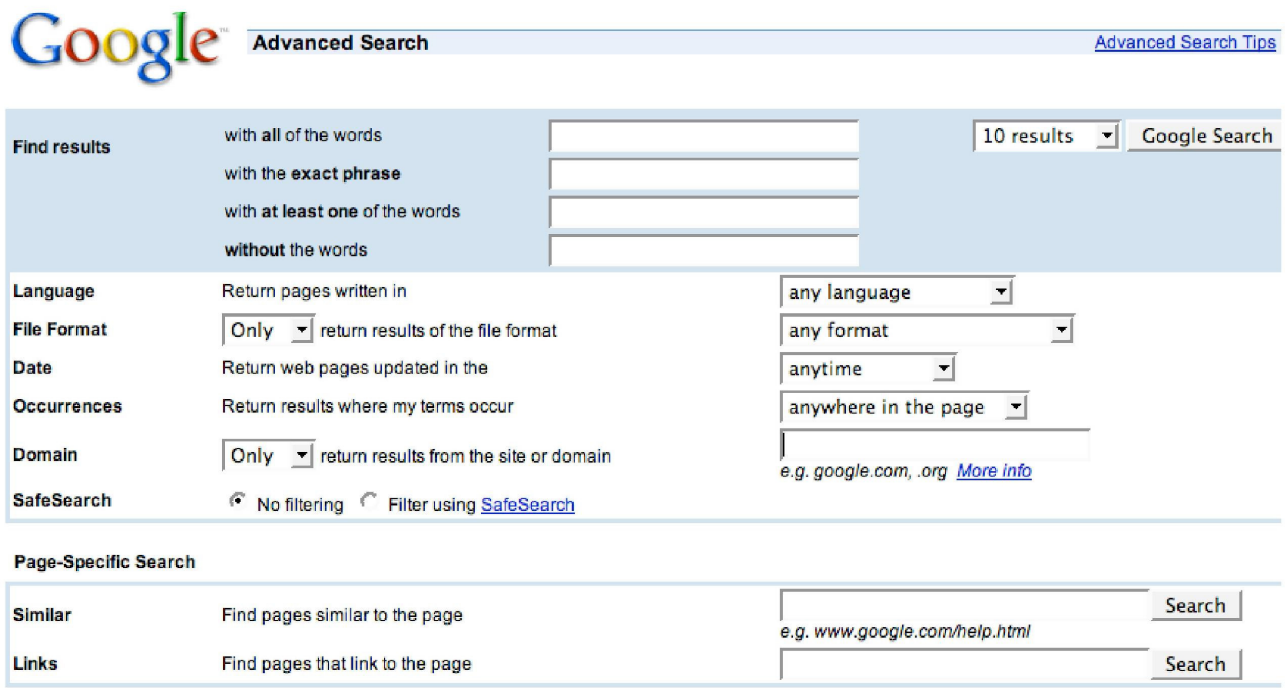
$RSV(d_i,(\textbf{not } q_j)) = 1 - RSV(d_i,q_j)$

$RSV(d_i,(\textbf{and } q_1 \ldots q_n)) = \min\{RSV(d_i,q_1),\ldots,RSV(d_i,q_n)\}$

$RSV(d_i,(\textbf{or } q_1 \ldots q_n)) = \max\{RSV(d_i,q_1),\ldots,RSV(d_i,q_n)\}$

- In queries, expert users tend to use only faceted queries, i.e. disjunctions of quasi-synonyms (*facets*) conjoined through **and**:
  (**and** (**or** jazz classical) (**or** sax* clarinet flute) (**or** Parker Coltrane))

- Extensions:
  - Truncation of the query term; e.g.
    q_3 = (**and** (**or** sax* clarinet) (**or** Parker Coltrane))
  - Information on adjacency, distance and word order invertibility; e.g.
    q_4 = (**and** jazz electric **prox[0,F]** guitar)
  - Possibility to restrict the search to given subfields such as title, author, abstract, publication date, etc. (fielded search).



Query Example

- Which plays of Shakespeare contain the words Brutus *AND* Caesar but *NOT* Calpurnia?

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

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for Brutus, Caesar and Calpurnia (complemented) ➔ bitwise *AND*

  110100 *AND* 110111 *AND* 101111 = 100100

Bigger corpora

- Consider n = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
- Say there are m = 500K distinct terms among these.

Can't build the matrix

- 500K x 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. Why?
- What's a better representation?
  - We only record the 1 positions.

Inverted index

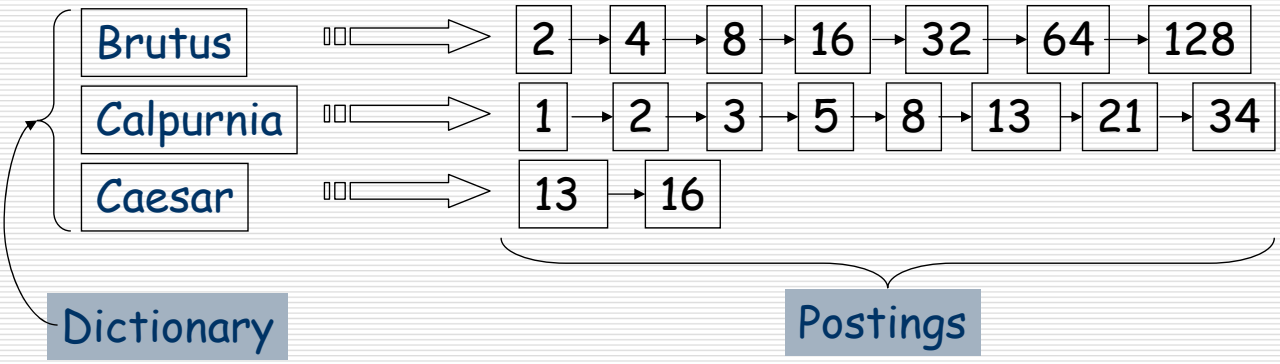
- For each term *t*, we must store a list of all documents that contain t.
- Do we use an array or a list for this?

Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

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

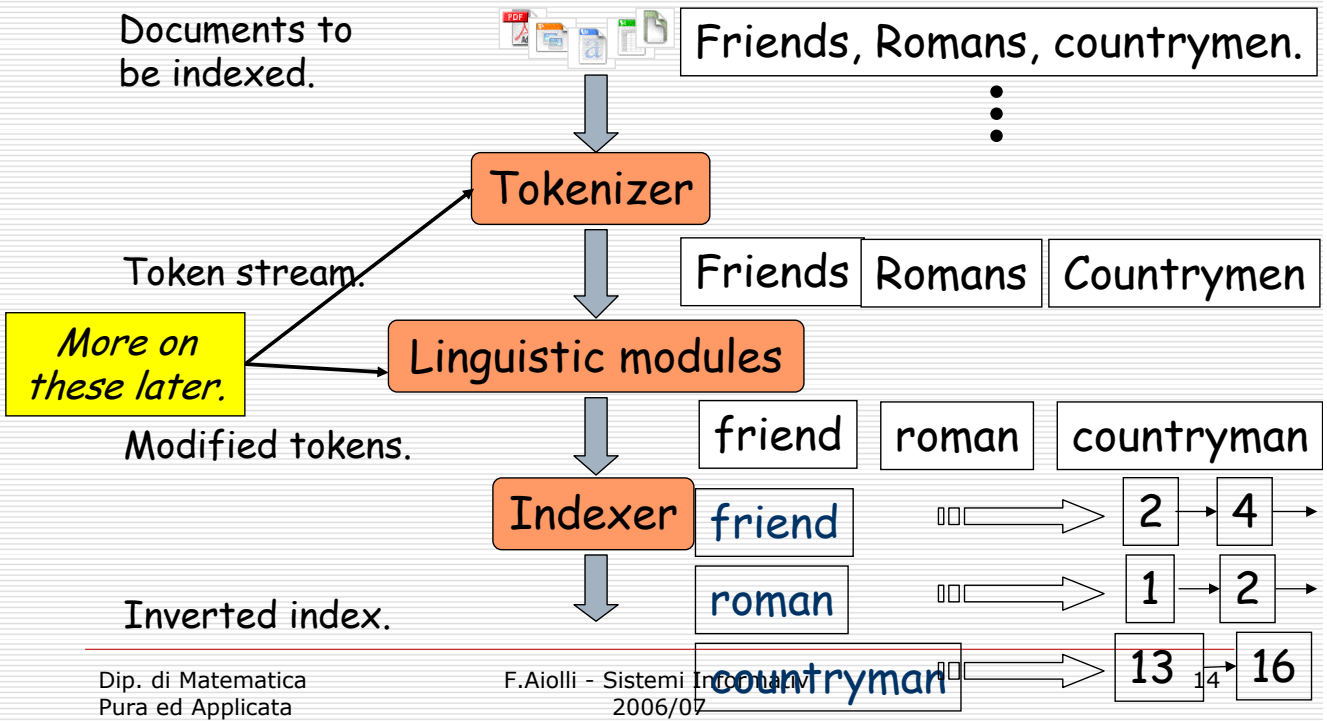
Inverted index

- Linked lists generally preferred to arrays
  - Pro: Dynamic space allocation
  - Pro: Insertion of terms into documents easy
  - Cons: Space overhead of pointers



Sorted by docID (more later on why).

Inverted index construction



Indexer steps

Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

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	1
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?
Will discuss later

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
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

Term	Doc #	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

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

Term	Doc #	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	N docs	Tot Freq
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	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
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
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Where do we pay in storage?

Terms

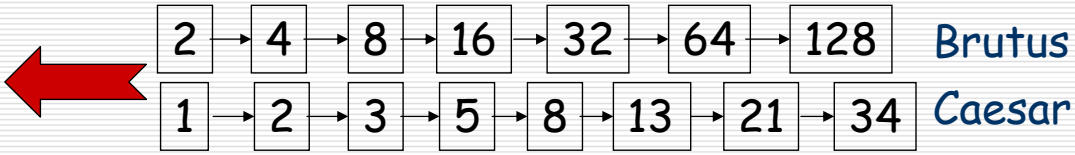
Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Pointers

Doc #	Freq
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
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

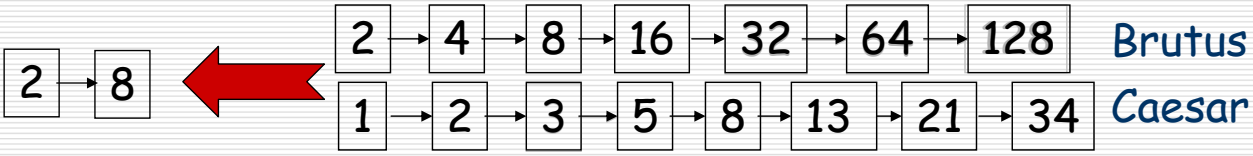
Query processing

- 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:



The merge

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



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

Crucial: postings sorted by docID.

More general merges

Exercise: Adapt the merge for the queries:

Brutus *AND NOT* ***Caesar***

Brutus *OR NOT* ***Caesar***

Can we still run through the merge in time O(*x*+*y*)?

Merging

What about an arbitrary Boolean formula?
***(Brutus** OR **Caesar)** AND NOT*
(Antony** OR **Cleopatra)

- Can we always merge in "linear" time?
  - Linear in what?
- Can we do better?

Beyond term search

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

Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
- Need term frequency information in docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Need to measure proximity from query to each doc.

Advantages of the BM

- Possibility to formulate structured queries. E.g. to distinguish 'synonymy' (**or** t_1 t_2) from noun phrases (**and** t_1 **prox[0,F]** t_2)

- In the case of expert users, intuitivity. To those users familiar with Boolean Logic, it is immediately clear why a doc has been or not been retrieved following a given query. This allows query refinement and query reformulation on the part of the user.

- Efficiency obtained through the use of inverted files (Ifs) the data structures in secondary storage in which document representations are physically stored.

Disadvantages of the BM

1. Intimidating (in general, there is not automatic query acquisition). The proponent of the BM where mathematicians; but "lay users" find the language of the BM unnatural.

2. Lack of output magnitude control after a given query, unless the user knows well the distribution of the topics in the collection. This is a consequence of the fact that RSV takes binary values.

3. Output obtained as a result of a given query is not ranked with respect to the estimated degree (probability) of relevance.

4. No importance factors can be attached to the index terms in the IREPs of documents and query.

5. "Flattened", hence unintuitive, results (lack of output discrimination)

Boolean systems have thus mainly been used by expert intermediaries and are widely considered unfit the needs of 'casual' (non professional) information seekers.