

Information Retrieval

Lecture 1

Query

- Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurnia*?
- Could grep all of Shakespeare's plays 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 phrase *Romans and countrymen*) not feasible

Term-document incidence

	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

Incidence vectors

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

Answers to query

■ 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.

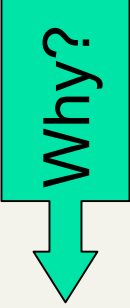
■ Hamlet, Act III, Scene ii

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

Bigger corpora

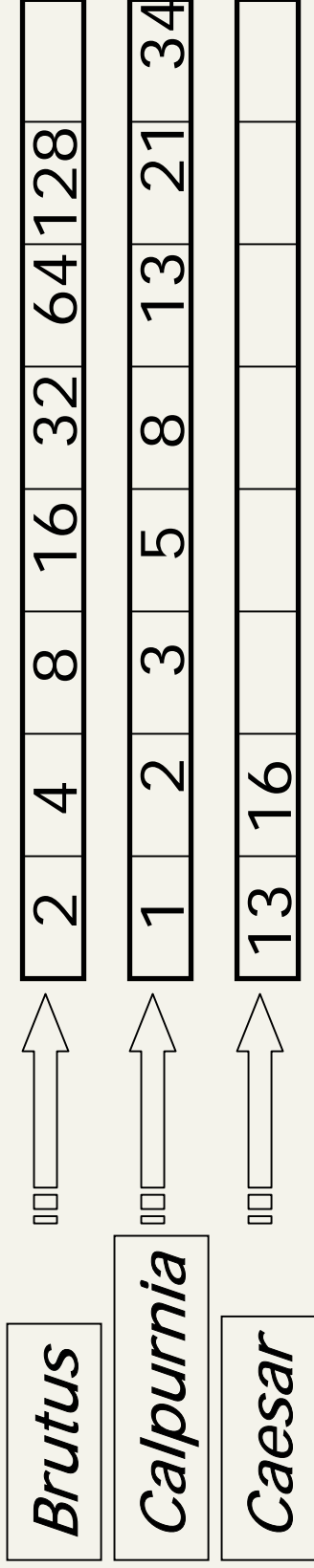
- Consider $n = 1\text{M}$ documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
 - 6GB of data in the documents.
- Say there are $m = 500\text{K}$ distinct terms among these.

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.
- What's a better representation?
 - We only record the 1 positions.

Inverted index

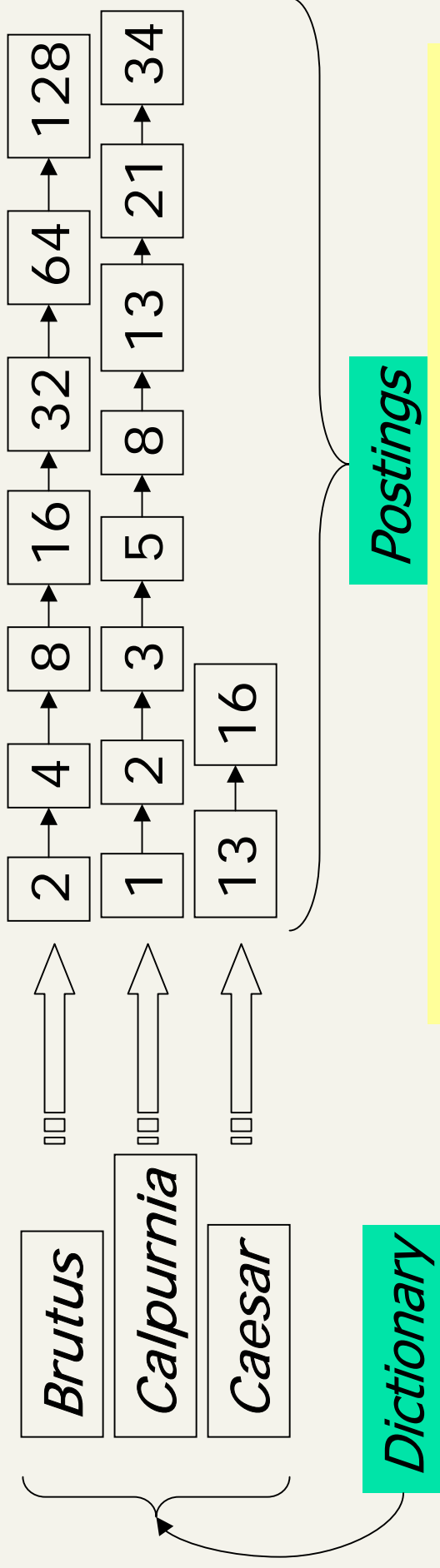
- For each term *T*, must store a list of all documents that contain *T*.
- Do we use an array or a list for this?



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

Inverted index

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



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

The index we just built

- How do we process a query?
 - What kinds of queries can we process?

Today's
focus



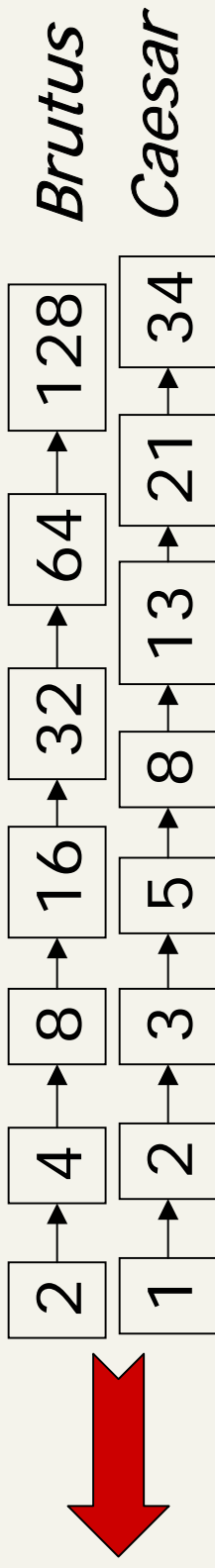
- Which terms in a doc do we index?
 - All words or only “important” ones?
- Stopword list: terms that are so common that they're ignored for indexing.
 - *e.g., the, a, an, of, to ...*
 - language-specific.

Query processing

- Consider processing the query:

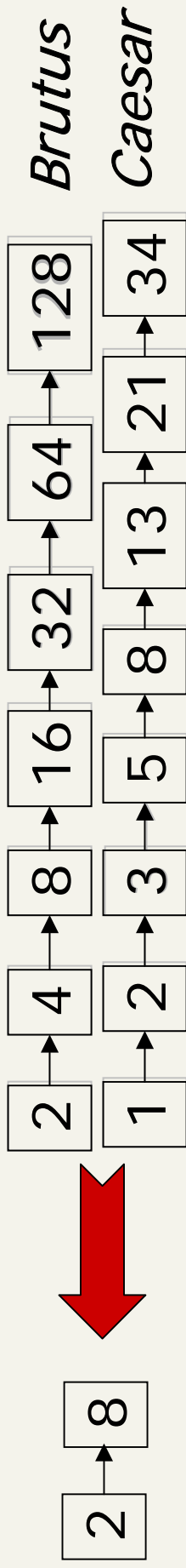
Brutus AND Caesar

- 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 m and n , the merge takes $O(m+n)$ operations.

Crucial: postings sorted by docID.

Boolean queries: Exact match

- 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.

Example: WestLaw

<http://www.westlaw.com/>

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- About 7 terabytes of data; 700,000 users
- Majority of users *still* use boolean queries
- Example query:
 - What is the statute of limitations in cases involving the federal tort claims act?
 - **LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM**
- Long, precise queries; proximity operators; incrementally developed; not like web search

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 $O(m+n)$?

Merging

What about an arbitrary Boolean formula?

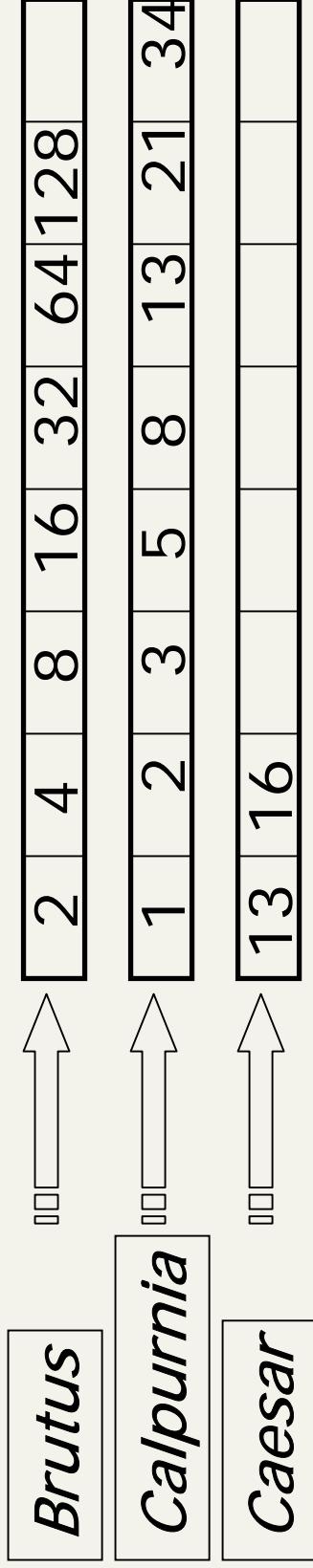
(Brutus OR Caesar) AND NOT

(Antony OR Cleopatra)

- Can we always merge in “linear” time?
- Can we do better?

Query optimization

- 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* together.



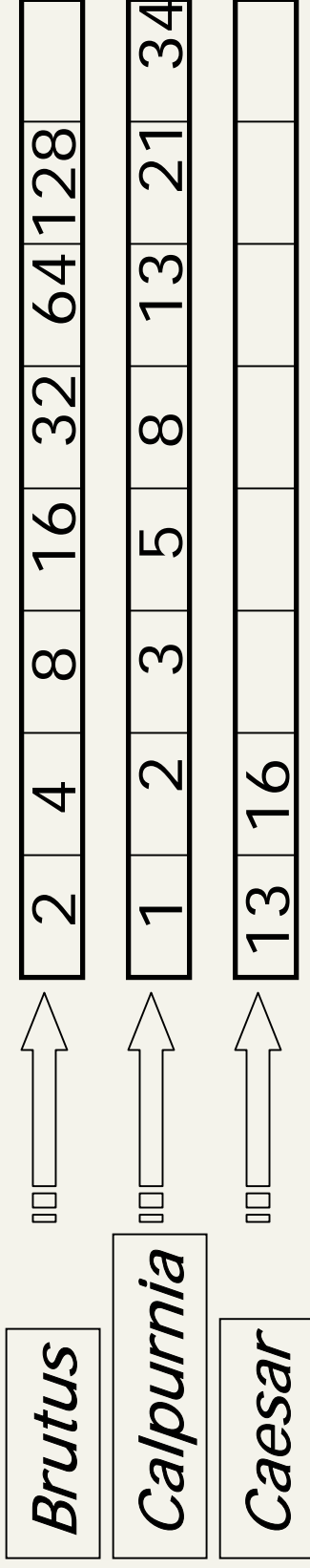
Query: *Brutus AND Calpurnia AND Caesar*

Query optimization example

- Process in order of increasing freq:
 - *start with smallest set, then keep cutting further.*



This is why we kept
freq in dictionary



Execute the query as (*Caesar AND Brutus*) AND *Calpurnia*.

More general optimization

- e.g., (*maddening 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.

Exercise

- Recommend a query processing order for

*(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)*

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Query processing exercises

- If the query is *friends AND romans AND (NOT countrymen)*, how could we use the freq of *countrymen*?
- **Exercise:** Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?
- **Hint:** Begin with the case of a Boolean *formula* query: the each query term appears only once in the query.

Digression: food for thought

- What if a doc consisted of *components*
 - Each component has its own *access control list*.
- Your search should get a doc only if your query meets one of its components that you have access to.
- More generally: doc assembled from *computations* on components
 - e.g., in Lotus databases or in content management systems
- Welcome to the real world ... more later.

Beyond term search

- What about phrases?
- Proximity: Find *Gates NEAR Microsoft*.
 - Need index to capture position information in docs. More later.
- 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.
- Whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Structured vs unstructured data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows numerical range and exact match (for text) queries, e.g.,

Salary < 60000 AND Manager = Smith.

Unstructured data

- Typically refers to free text
- Allows
 - Keyword queries including operators
 - More sophisticated “concept” queries e.g.,
 - find all web pages dealing with *drug abuse*
- Classic model for searching text documents

Semi-structured data

- But in fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
 - *Title* contains data AND *Bullets* contain search

More sophisticated semi-structured search

- *Title* is about Object Oriented Programming
AND *Author* something like stro*rup
- where * is the wild-card operator
- Issues:
 - how do you process “about”?
 - how do you rank results?
- The focus of XML search.

Clustering and classification

- Given a set of docs, group them into clusters based on their contents.
- Given a set of topics, plus a new doc D , decide which topic(s) D belongs to.

The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
- Beyond terms, exploit ideas from social networks
 - link analysis, clickstreams ...

Exercise

- Try the search feature at <http://www.rhymezone.com/shakespeare/>
- Write down five search features you think it could do better

Course administrivia

- 2 lectures each morning
- On Thursday 26th afternoon, special session for projects
 - Available resources
 - Target projects – right scope
- Thursday 2nd and Friday 3rd afternoons, proposals of projects – student presentations

Resources for today's lecture

- *Managing Gigabytes*, Chapter 3.2
- *Modern Information Retrieval*, Chapter 8.2
- Shakespeare:
<http://www.rhymezone.com/shakespeare/>
- Try the neat browse by keyword sequence feature!