

CS3245

Information Retrieval

Lecture 2: Boolean retrieval

2

Blanks on slides, you may want to fill in

Last Time: Ngram Language Models

- Unigram LM: Bag of words
- Ngram LM: use $n-1$ tokens of context to predict n^{th} token
- Larger n -gram models more accurate but each increase in order requires exponentially more space

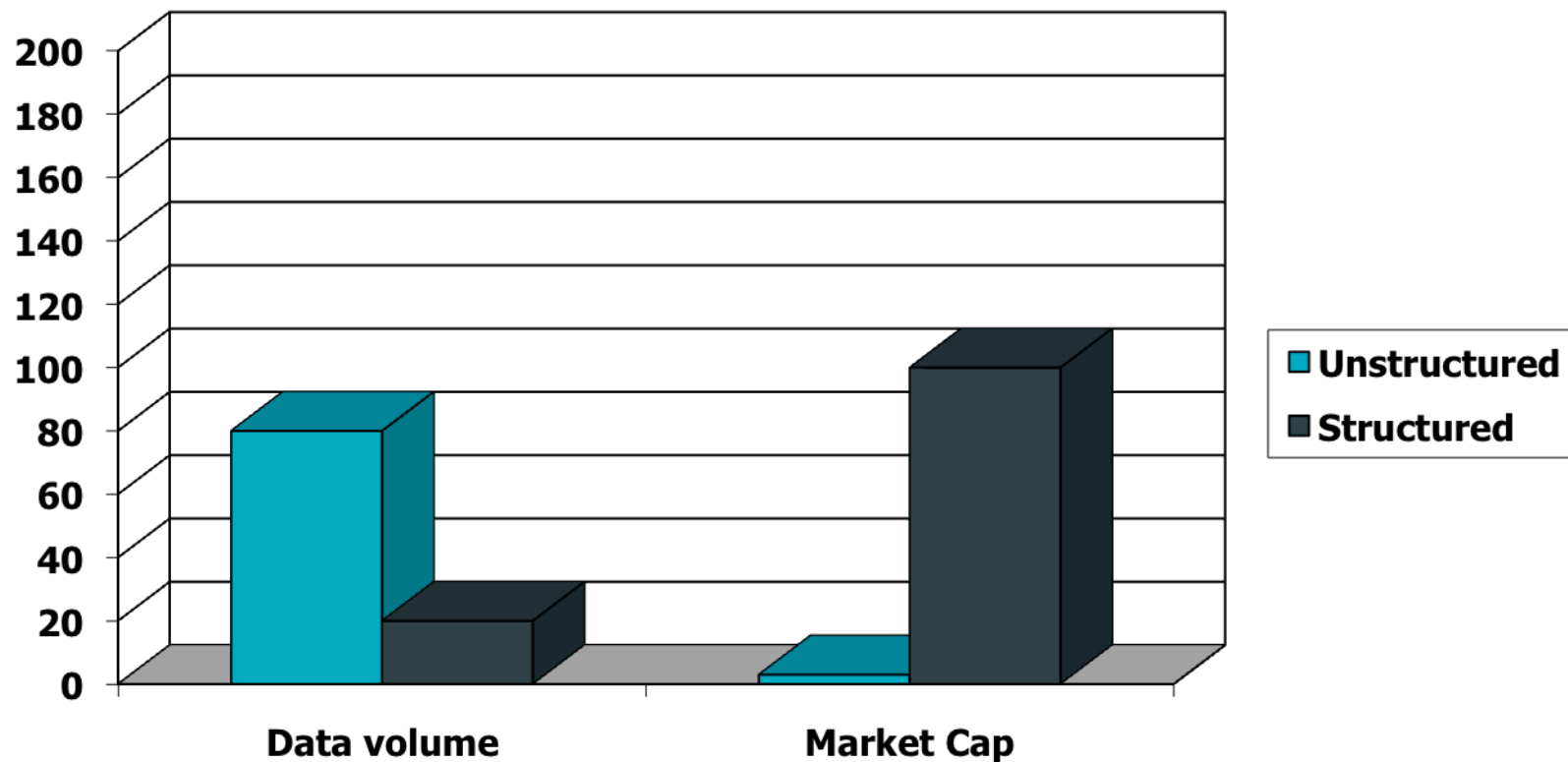
Your turn: what do you think? Can we use a LM to do information retrieval?

Information Retrieval

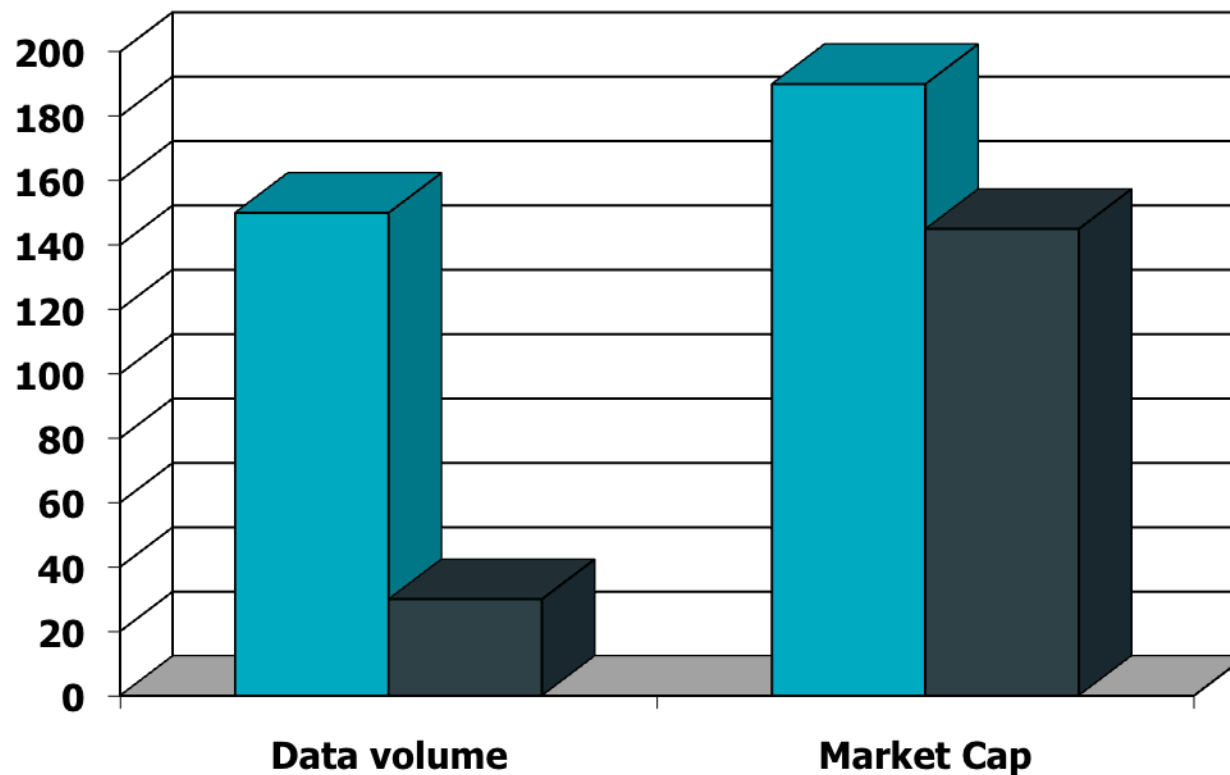


- Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured** nature (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).

Unstructured (text) vs. structured (database) data in 1996



Unstructured (text) vs. structured (database) data in 2009



Google™

YAHOO!®

■ Unstructured
■ Structured

bing

Ask™
.com

Unstructured data in 1680



- Which plays of Shakespeare contain the words ***Brutus*** ***AND Caesar*** but ***NOT Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?
- It's one answer, but why isn't it the only answer?
 - 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)
 - Later in W7 (after the midterm): Scoring and VSM

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

Brutus AND Caesar BUT NOT Calpurnia

1 if **play** contains **word**, 0 otherwise

Blanks on slides, you may want to fill in



Incidence vectors

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

$$110100 \text{ AND } 110111 \text{ AND } 101111 = 100100.$$

What are the answers to that 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.

N.B: In this example, the “documents” being indexed are single lines in each play.
Check your understanding: What would happen if we were to treat plays as documents?

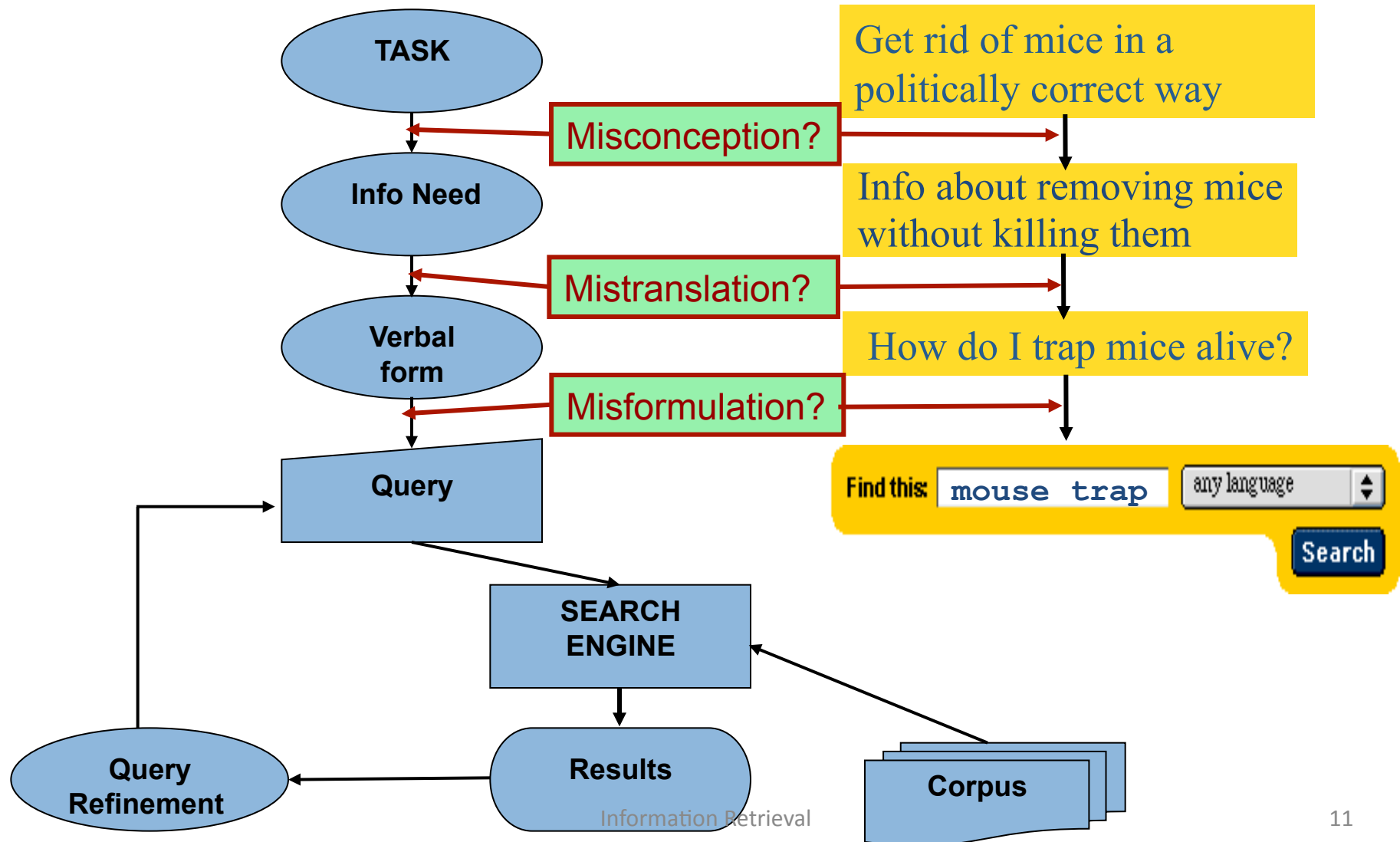




Basic assumptions of Information Retrieval

- **Collection**: Fixed set of documents
- **Goal**: Retrieve documents with information that is relevant to the user's **information need** and helps the user complete a **task**

The classic search model





How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved
- More precise definitions and measurements to follow in later lectures

Bigger collections



- Consider $N = 1$ million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
 - 6GB of data in the documents.
- Say there are $M = 500K$ *distinct* terms among these.

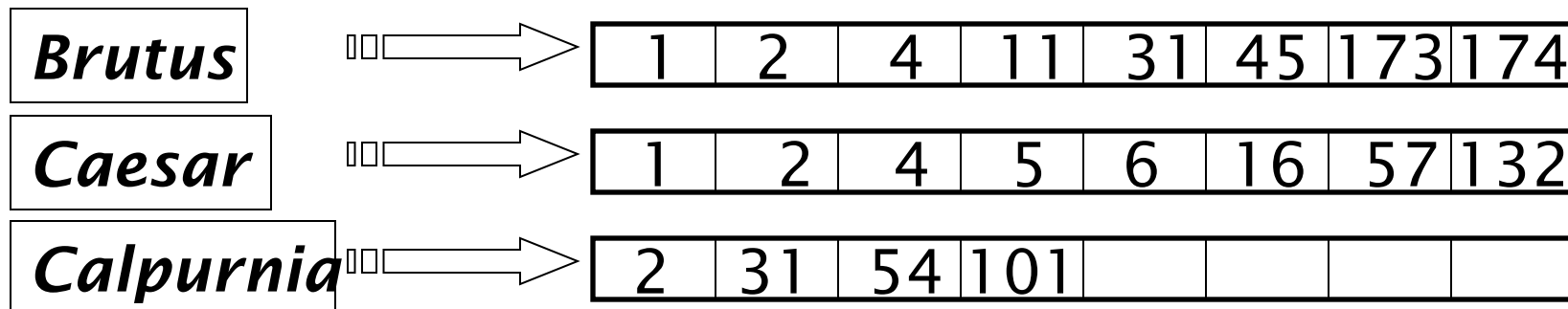


Can't build the matrix

- $500K \times 1M$ matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's.
 - Matrix is extremely sparse.
 - Check your understanding: why?
 - How many 1's are there on average in a row?
 - What do you call a document that has all 1's in its entry?
- What's a better representation?
 - We only record the positions of the 1's.

Inverted index

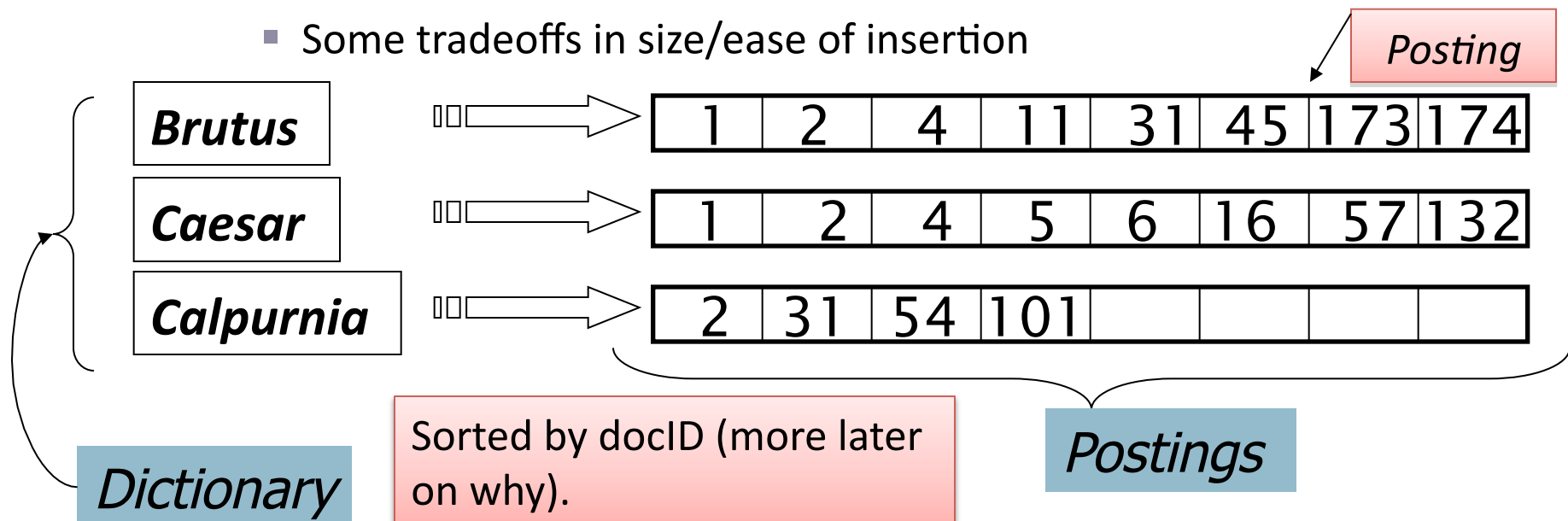
- For each term t , we must store a list of all documents that contain t .
 - Identify each by a **docID**, a document serial number
- Can we use fixed-size arrays for this?



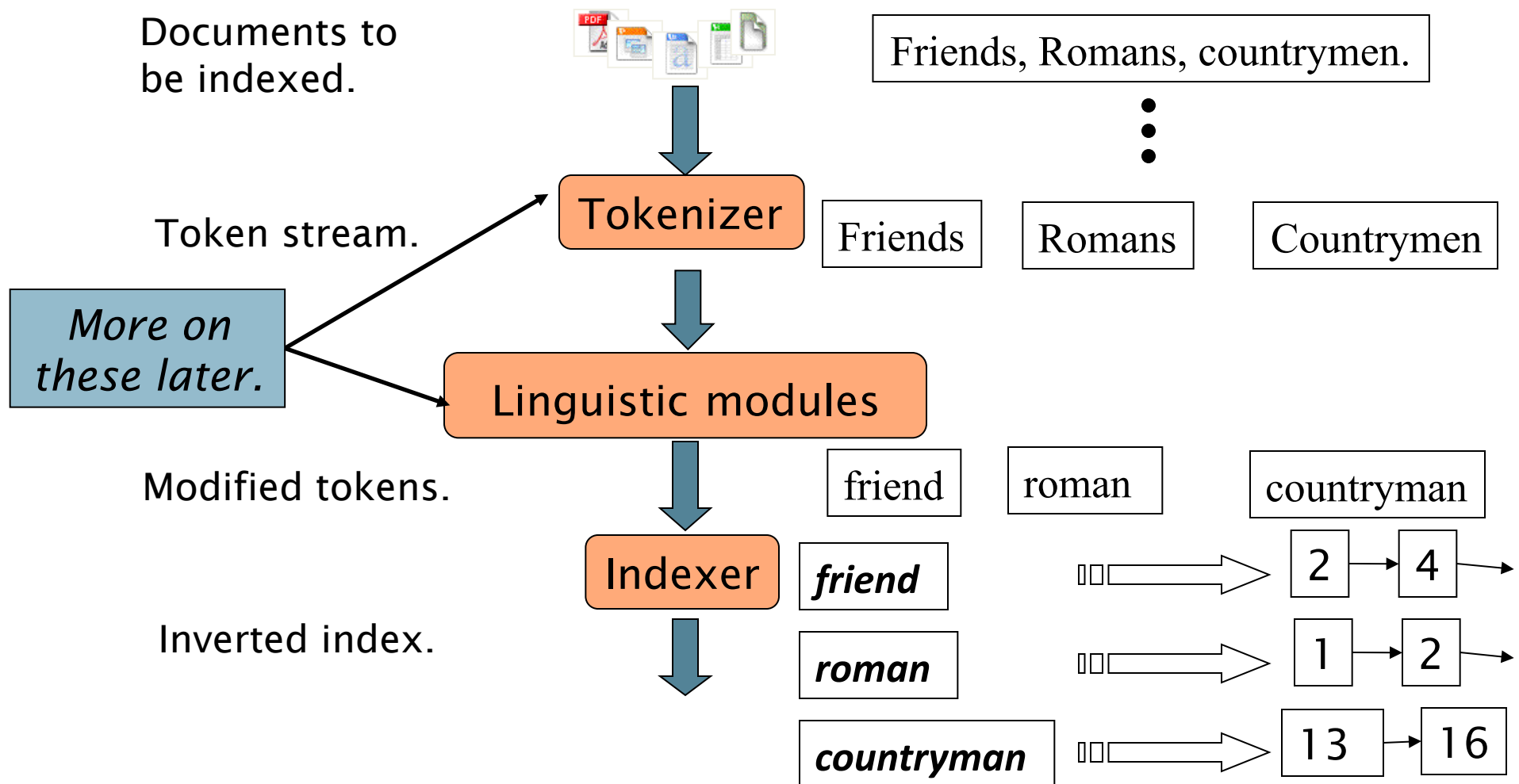
What happens if the word **Caesar** is added to document 14?

Inverted index

- We need variable-size postings lists
 - On disk, a continuous run of postings is normal and best
 - In memory, can use linked lists or variable length arrays
 - Some tradeoffs in size/ease of insertion



Inverted index construction



Indexer steps: Token sequence

- 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	docID
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
	18

Indexer steps: Sort

- Sort by terms
 - And then docID

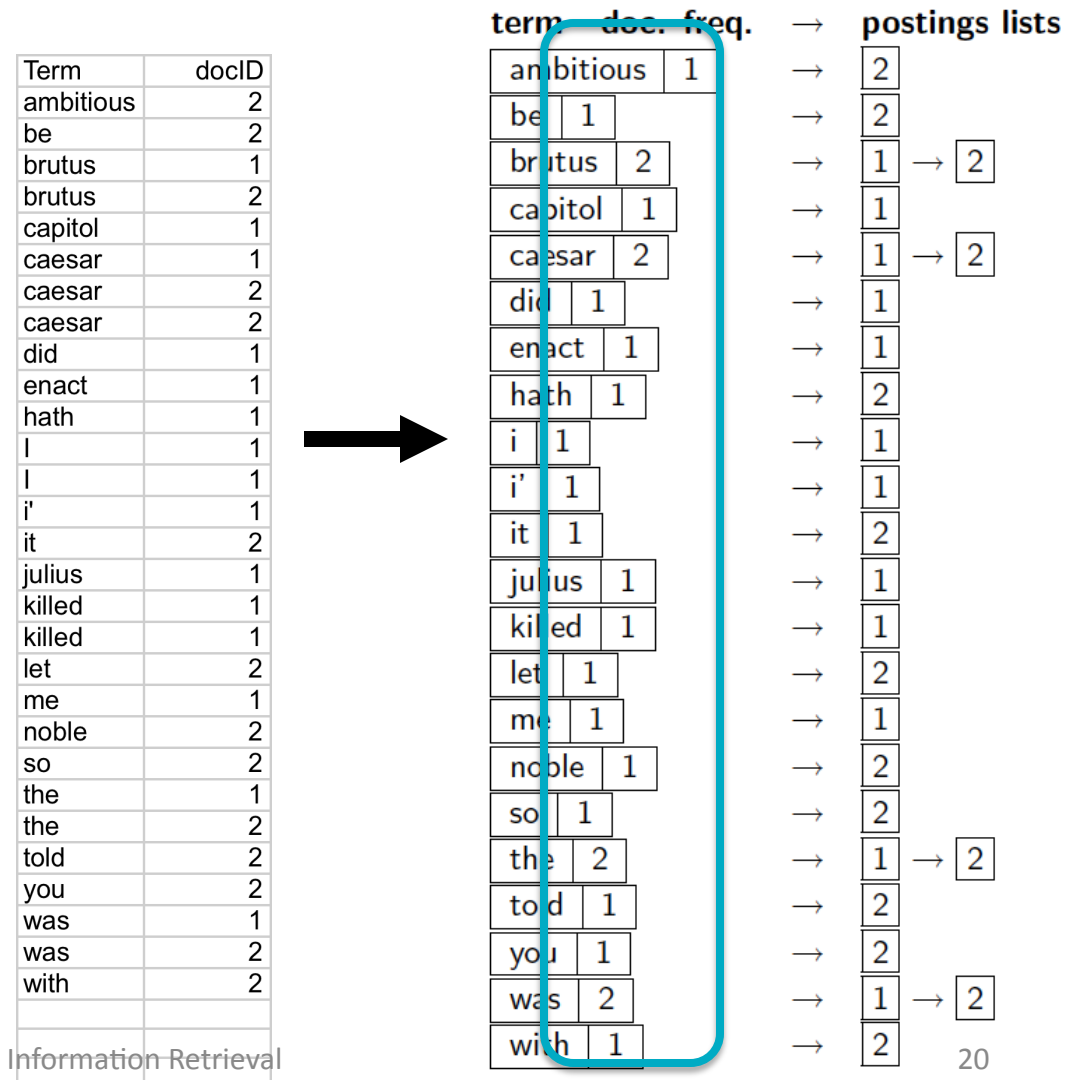
Core indexing step

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

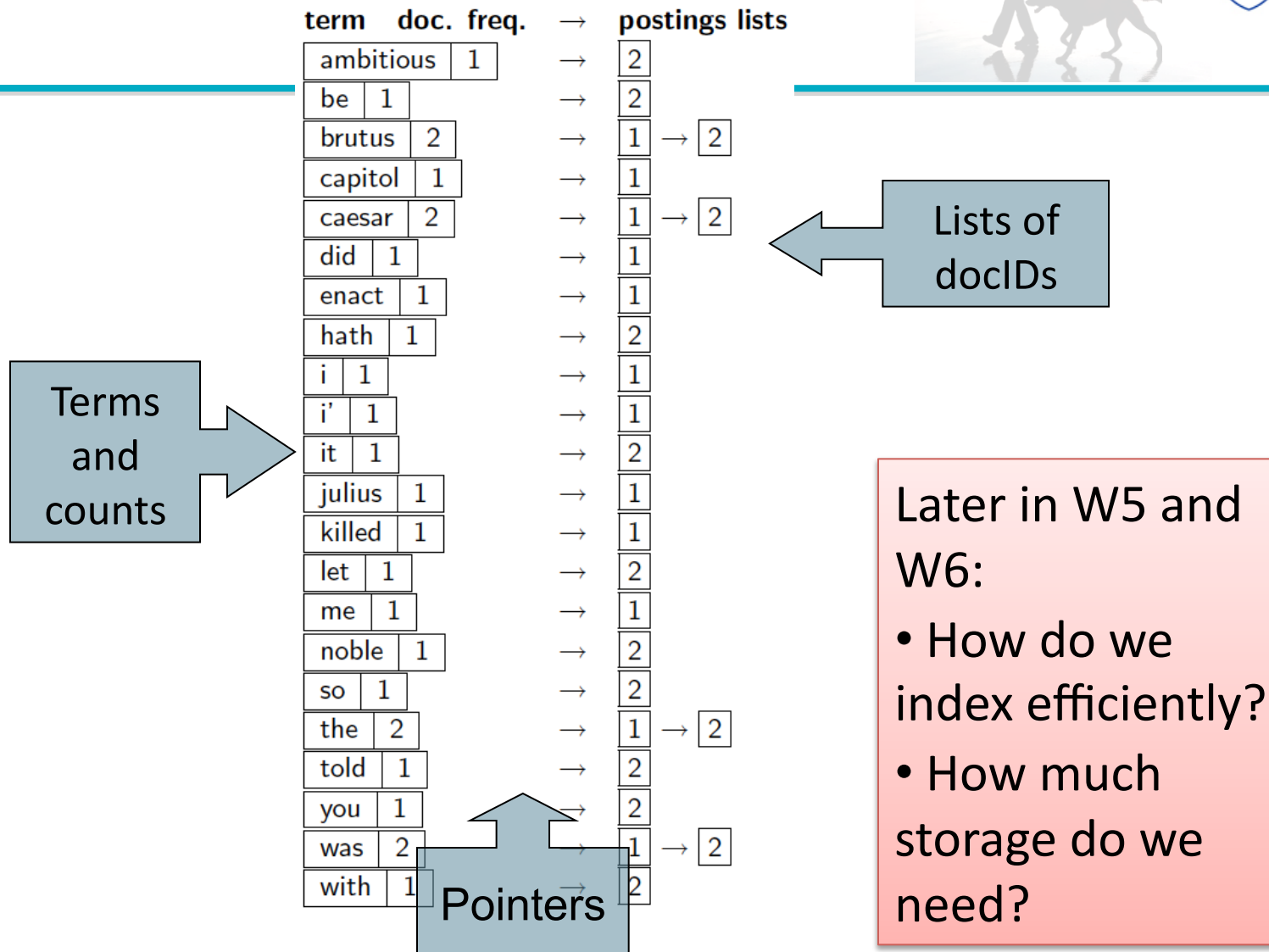
Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency** information is also stored.

Why frequency?
Will discuss later.



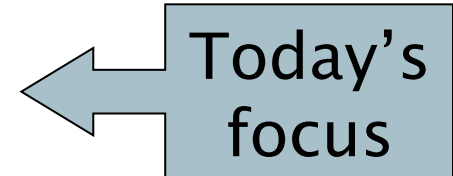
What do we pay in storage?



The index we just built



- How do we process a query?
 - Later in W3 and W4 –
what kinds of queries can we process?



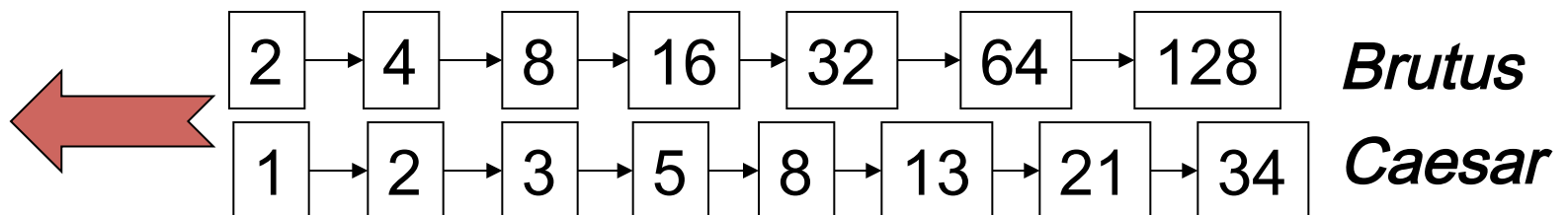


Query processing: AND

- Consider processing the query:

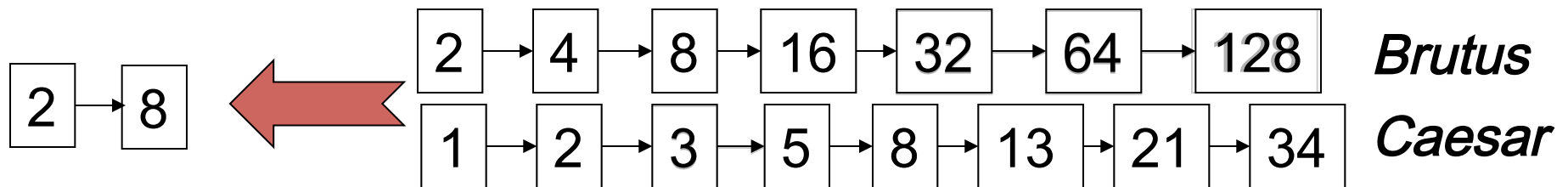
Brutus AND Caesar

- 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 must be sorted by docID.

Intersecting two postings lists (a “merge” algorithm)



```
INTERSECT( $p_1, p_2$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $docID(p_1) = docID(p_2)$ 
4      then  $\text{ADD}(answer, docID(p_1))$ 
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7      else if  $docID(p_1) < docID(p_2)$ 
8          then  $p_1 \leftarrow next(p_1)$ 
9          else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 
```



Boolean queries: Exact match

- 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.
 - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
 - Email, library catalog, Mac OS X Spotlight



Example: WestLaw

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

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Tens of 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
 - /3 = within 3 words, /S = in same sentence

Example: WestLaw

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



- Another example query:
 - Requirements for disabled people to be able to access a workplace
 - `disabl! /p access! /s work-site work-place (employment /3 place`
- Note that SPACE is disjunction, not conjunction!
- Long, precise queries; proximity operators; incrementally developed; not like web search
- Many professional searchers still like Boolean search
 - You know exactly what you are getting
- But that doesn't mean it actually works better....

Boolean queries: More general merges



- Exercise: Adapt the merge for the queries:

Brutus AND NOT Caesar

Brutus OR NOT Caesar

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

What can we achieve?



Merging

What about an arbitrary Boolean formula?

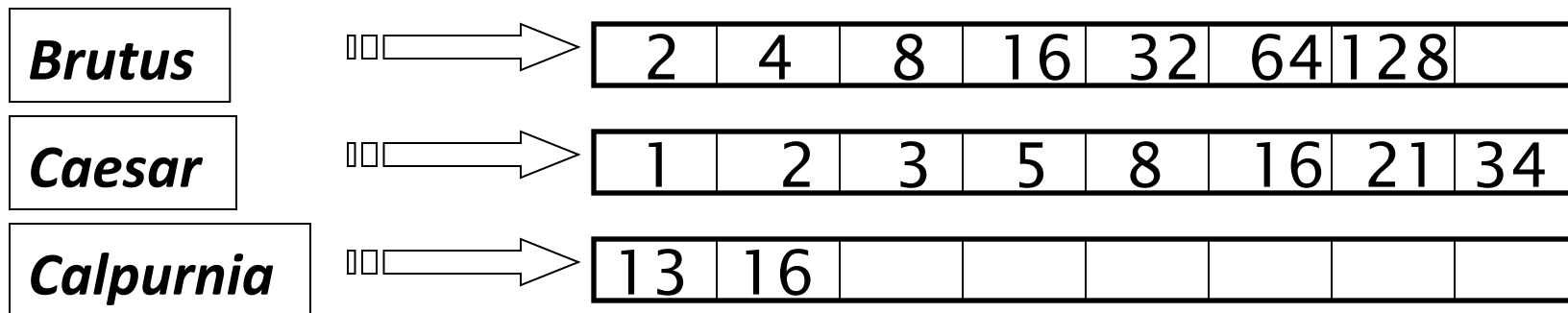
(Brutus OR Caesar) AND NOT

(Antony OR Cleopatra)

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

Query optimization

- What is the best order for query processing?
- Consider a query that is an *AND* of n terms.
- For each of the n terms, get its postings, then *AND* them 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
document frequency in the dictionary!

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

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

Blanks on slides, you may want to fill in



More general optimization

- e.g., (*madding OR crowd*) AND (*ignoble OR strife*)
- Get document frequencies (**dfs**) for all terms.
- Estimate the size of each *OR* by the sum of its df's (conservative).
- Process in increasing order of *OR* sizes.

Check your memory: What other course has something similar?

Check your understanding

- 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

- **Exercise:** 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: in this, each query term appears only once in the query.

What's ahead in IR?

Beyond term search



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

To think about: which (if any) of these does
ngram LMs could potentially address?



Evidence accumulation

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



Ranking search results

- Boolean queries just give setwise semantics: inclusion or exclusion of docs.
- Often we want to rank or group results
 - Need to measure proximity from query to each doc.
 - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

IR vs. databases:

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

- 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

... to say nothing of linguistic structure

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 (*IIR* chapter 10)

Clustering, classification and ranking

- **Clustering:** Given a set of docs, group them into clusters based on their contents.
- **Classification:** Given a set of topics, plus a new doc D , decide which topic(s) D belongs to.
- **Ranking:** Can we learn how to best order a set of documents, e.g., a set of search results

We won't cover the first two in this class; take machine learning and advanced IR for these topics.



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 ...
- How do search engines work? And how can we make them better?

More sophisticated *information* retrieval

- Cross-language information retrieval
- Question answering
- Summarization
- Text mining
- ...

✓ Many advanced IR topics require an understanding of language; this is why Natural Language Processing and IR are intertwined courses.



Summary

Covered the whole of information retrieval from 1000 feet up

- Indexing to store information efficiently for both space and query time.
- Run time builds relevant document list. Must be *fast*.

Resources for today's lecture

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