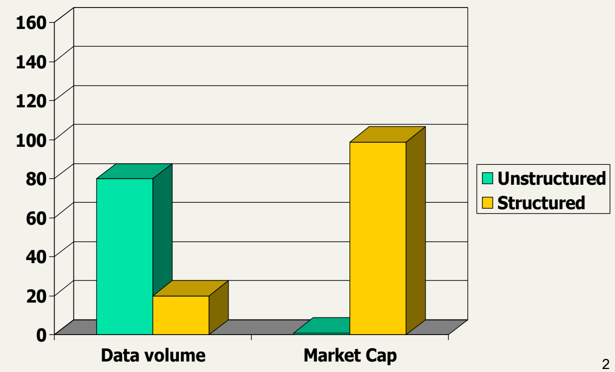
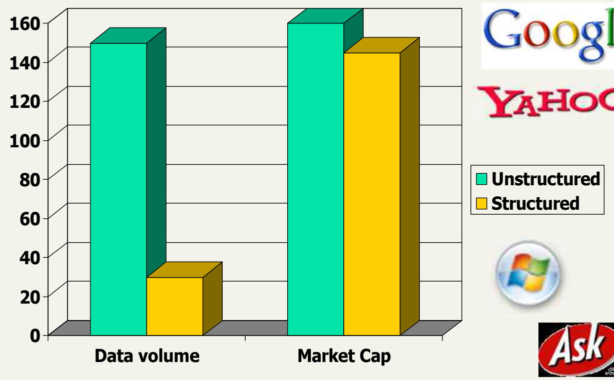
Information Retrieval

Lecture 1: Boolean retrieval

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



Unstructured (text) vs. structured (database) data in 2006



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*?
  - 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 lectures

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

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.

6

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.

7

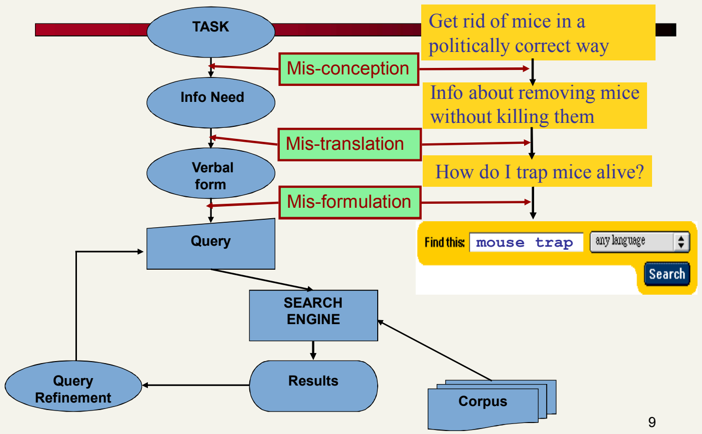
Basic assumptions of Information Retrieval

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

SIGIR 2005

8

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* = 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.

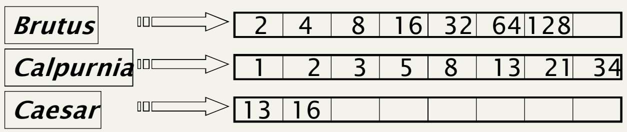
Can't build the matrix

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

Why?

Inverted index

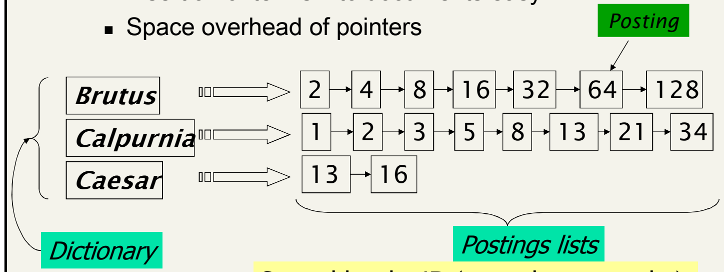
- For each term *T*, we must store a list of all documents that contain *T*.
- Do we use an array or a list for this?



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

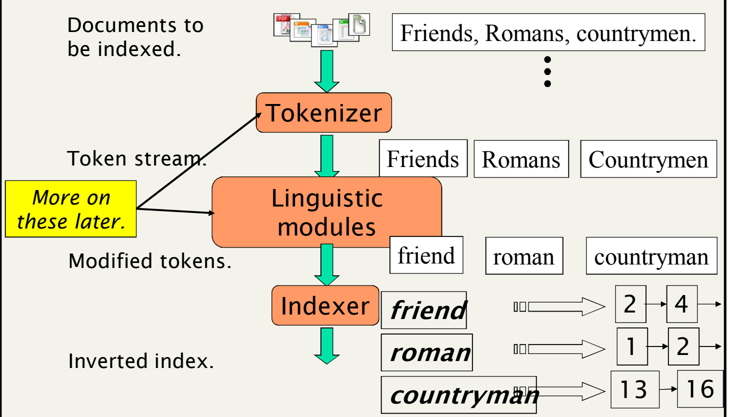
Inverted index

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



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 #	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

- 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	Coll 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	Coll 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	[illegible]
1	[illegible]
2	[illegible]
1	[illegible]
1	[illegible]
2	[illegible]
1	[illegible]
1	[illegible]
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

Pointers

Will quantify the storage, later.

20

The index we just built

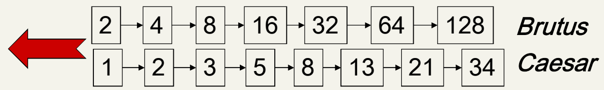
- How do we process a query?
  - Later - what kinds of queries can we process?

Today's focus

21

Query processing: AND

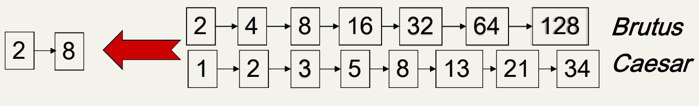
- Consider processing the query: *Brutus AND Caesar*
  - Locate *Brutus* in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



22

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.

23

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.
- Primary commercial retrieval tool for 3 decades.
- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting.

24

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

25

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
- Professional searchers often like Boolean search:
  - Precision, transparency and control
- But that doesn't mean they actually work better….

Boolean queries: 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($x+y$)? What can we achieve?

27

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?

28

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* them together.

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

Query: ***Brutus*** AND ***Calpurnia*** AND ***Caesar***

29

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

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

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

30

More general optimization

- e.g., *(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.

31

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

32

Query processing exercises

- If the query is *friends AND romans AND (NOT countrymen)*, how could we use the freq of *countrymen*?

33

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. More later.
- Zones in documents: Find documents with (*author = Ullman*) *AND* (text contains *automata*).

34

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

35

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Often we want to rank/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.

36

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.

37

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

38

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 <u>data</u> AND *Bullets* contain <u>search</u>

… to say nothing of linguistic structure

39

More sophisticated semi-structured search

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

40

More sophisticated *information* retrieval

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

41