UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Natural Language Processing

Information retrieval

LESSON 11

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

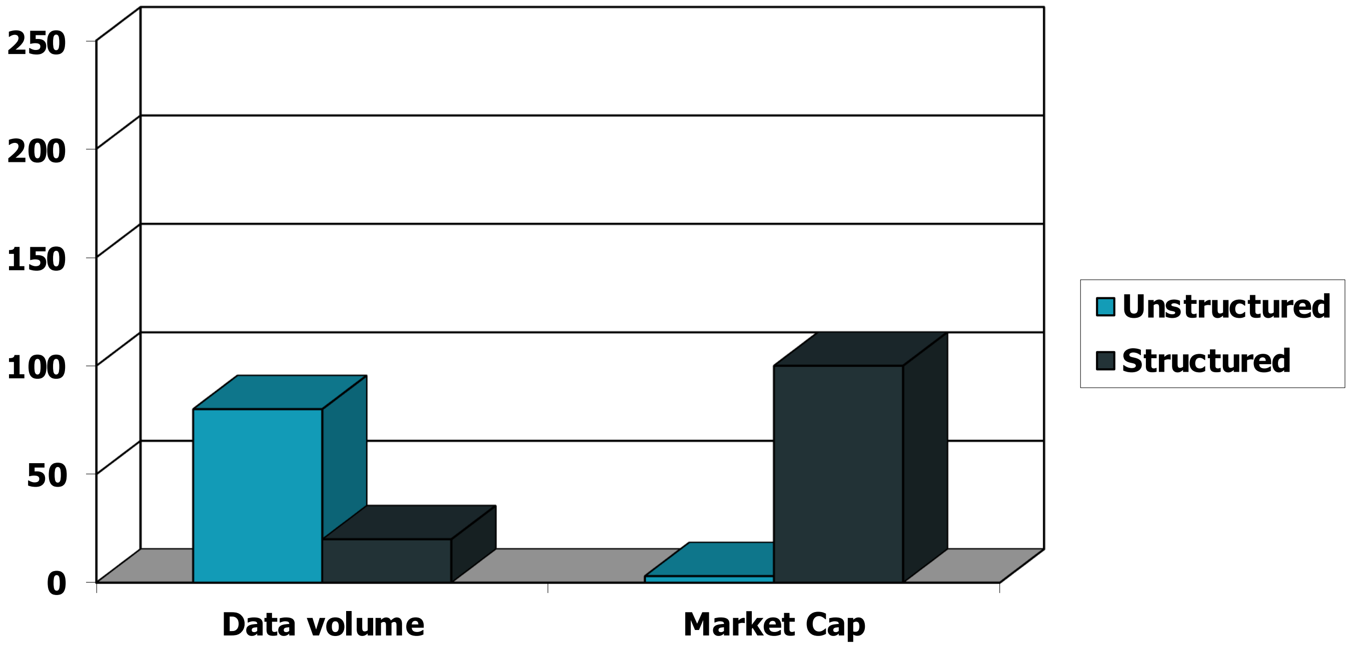
The slides are taken from Chris Manning NLP course

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)
  - These days we frequently think first of web search, but there are many other cases
    - E-mail search
    - Searching your laptop
    - Corporate knowledge bases
    - Legal information retrieval

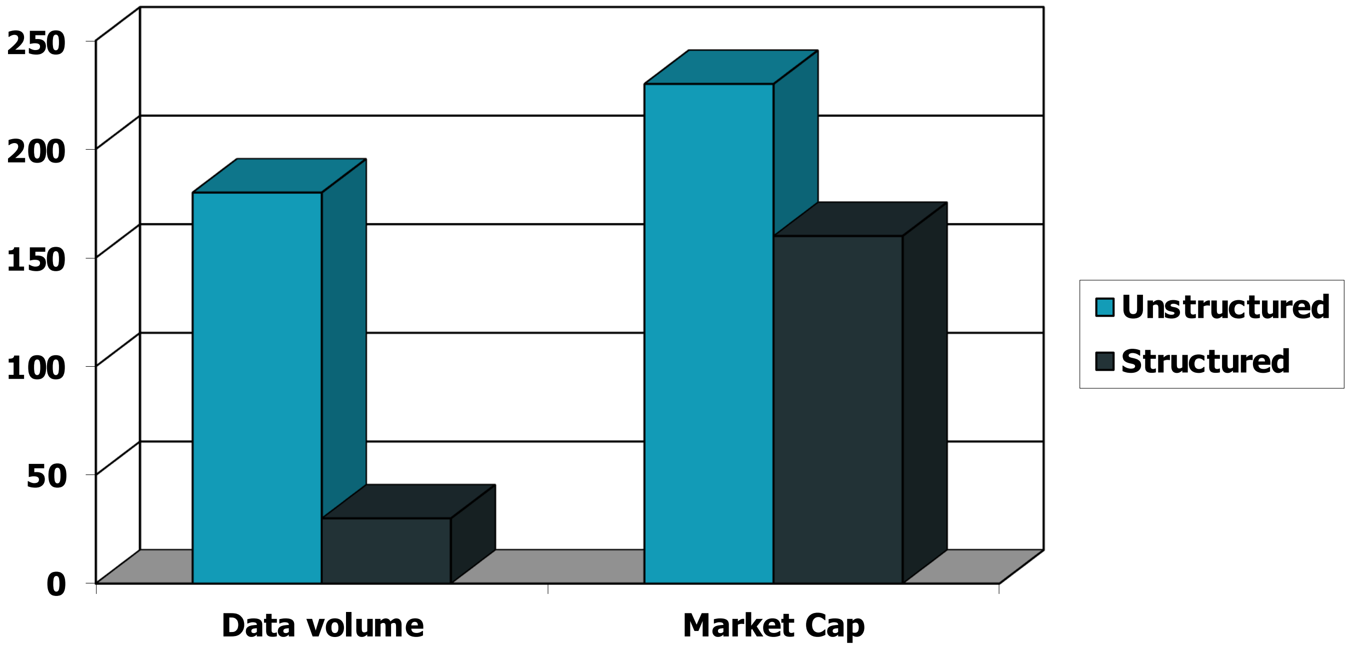
Unstructured (text) vs structured (DB) data

- Mid '90s



Unstructured (text) vs. structured (DB) data

- Today

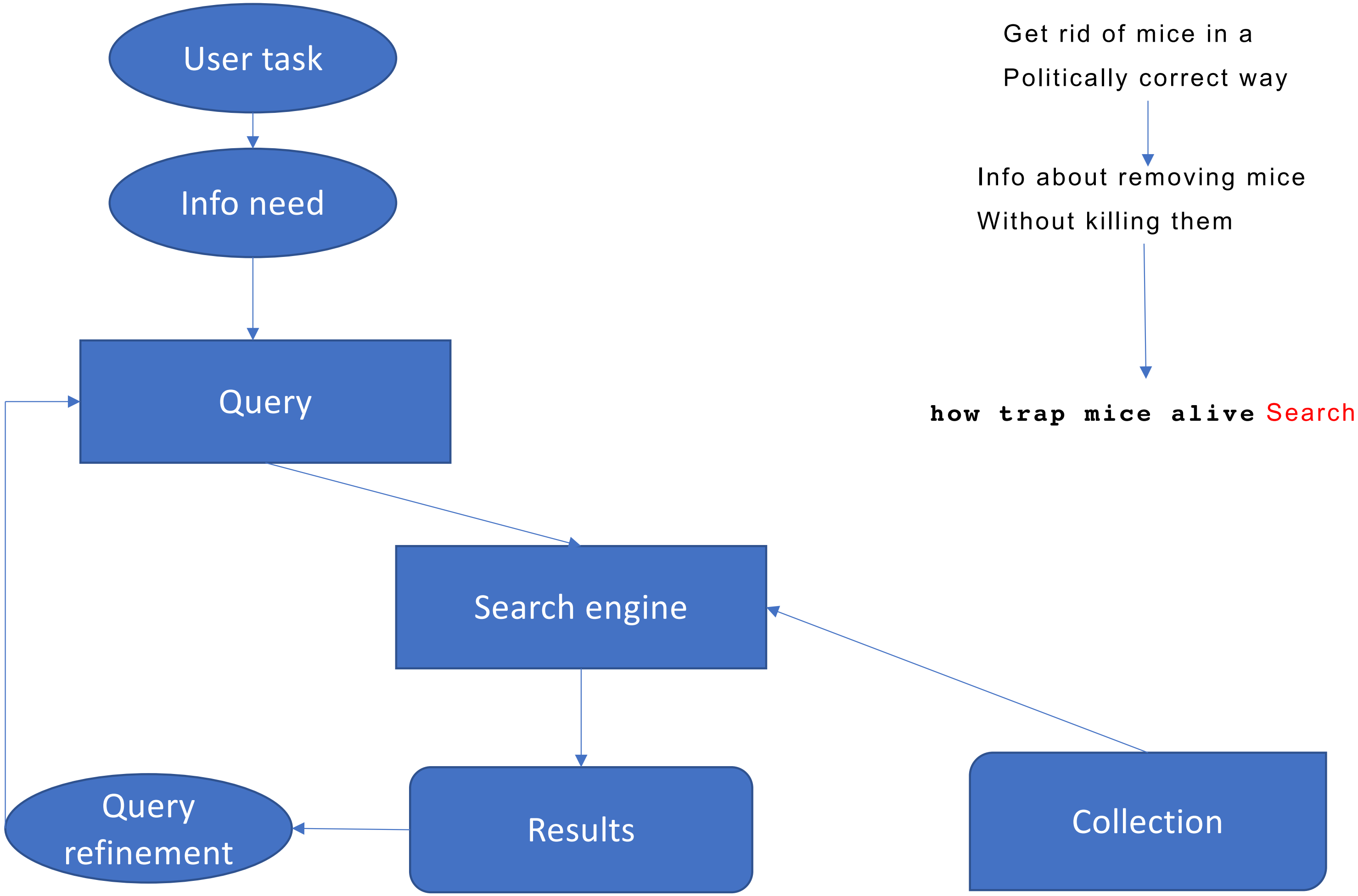


Basic assumptions of Information Retrieval

- Collection: A set of documents
  - Assume it is a static collection for the moment

- 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 the user's information need
- Recall
  - Fraction of relevant docs in the collection that are retrieved

Boolean retrieval model

- 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***?
- Why is that not the answer?
  - Slow (for large corpora)
  - <u>*NOT*</u> ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)
    - Next lecture

Term-document incidence matrices

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

Incidence vectors

- 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

	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

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 collections (corpora)

- 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 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 1s' positions

Why?

Inverted index

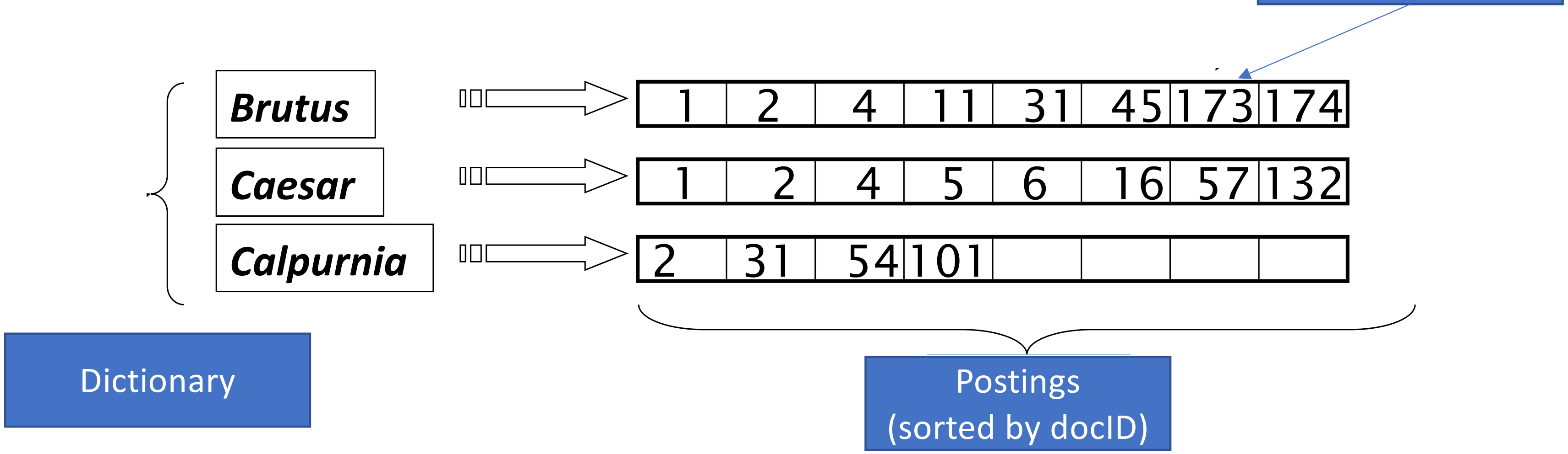
- The key data structure underlying modern IR
- For each term **t**, we must store a list of all documents that contain **t**
  - Identify each doc by a **docID**, a document serial number
- Can we use fixed-size arrays for this?

Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

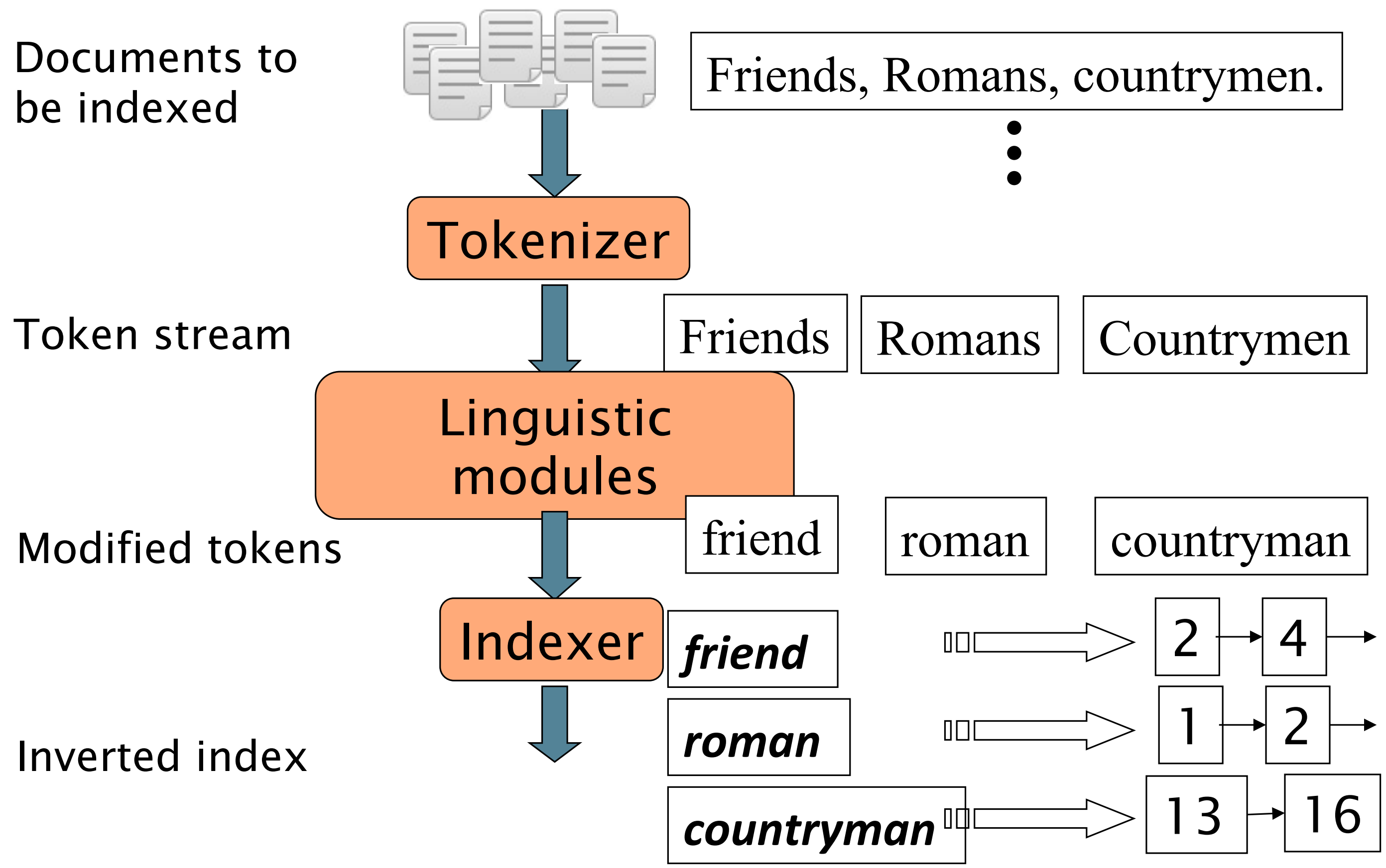
What happens if the word ***Caesar*** is added to document 14?

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



Recall: Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
    - Deal with *"John's"*, *a state-of-the-art solution*
- Normalization
  - Map text and query term to same form
    - You want *U.S.A.* and *USA* to match
- Stemming
  - We may wish different forms of a root to match
    - *authorize*, *authorization*
- Stop words
  - We may omit very common words (or not)
    - *the, a, to, of*

Indexer steps: Token sequence

- A 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

Indexer steps: Sort

- Sort by terms
  - And then docID

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

Term	docID
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

Indexer steps: Dictionary & Postings

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

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

Where do we pay in storage?

- IR system implementation
  - How do we index efficiently?
  - How much storage do we need?

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

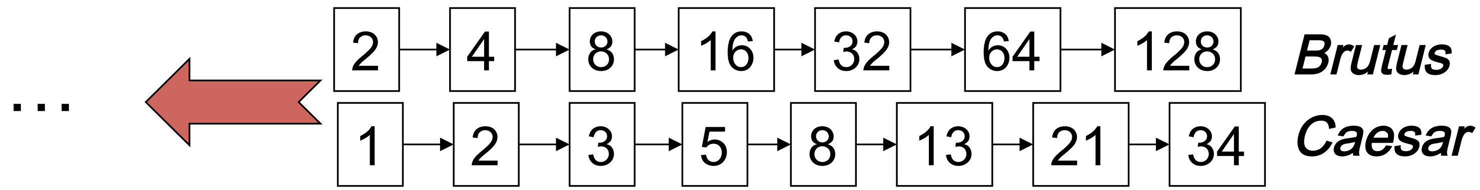
Terms and counts

Lists of docIDs

Pointers

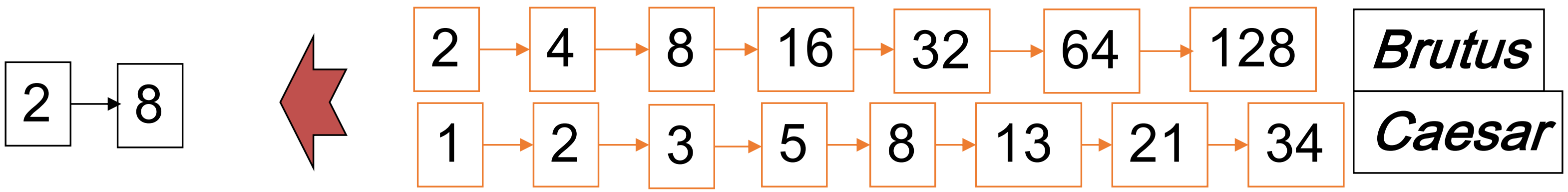
Query processing: AND

- Consider processing the query:
  - *Brutus* AND *Caesar*
    1. Locate *Brutus* in the dictionary
    2. Retrieve its postings
    3. Locate *Caesar* in the dictionary
    4. Retrieve its postings
    5. "Merge" the two postings (intersect the document sets):



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
    - It's crucial posting sorted by docID



Intersecting two postings lists

```
INTERSECT(p1, p2)
  answer ← ⟨ ⟩
  while p1 ≠ NIL and p2 ≠ NIL
  do if docID(p1) = docID(p2)
        then ADD(answer, docID(p1))
             p1 ← next(p1)
             p2 ← next(p2)
        else if docID(p1) < docID(p2)
                then p1 ← next(p1)
                else p2 ← next(p2)
  return answer
```

Boolean queries: Exact match

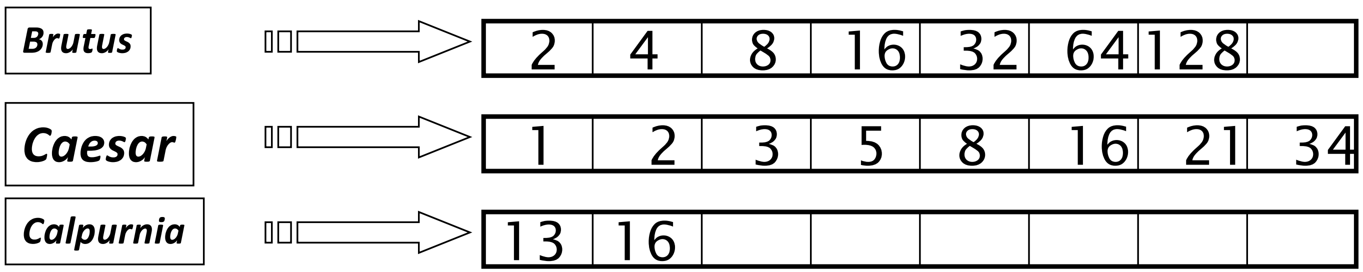
- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using AND, OR and NOT to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for three decades
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

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?

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

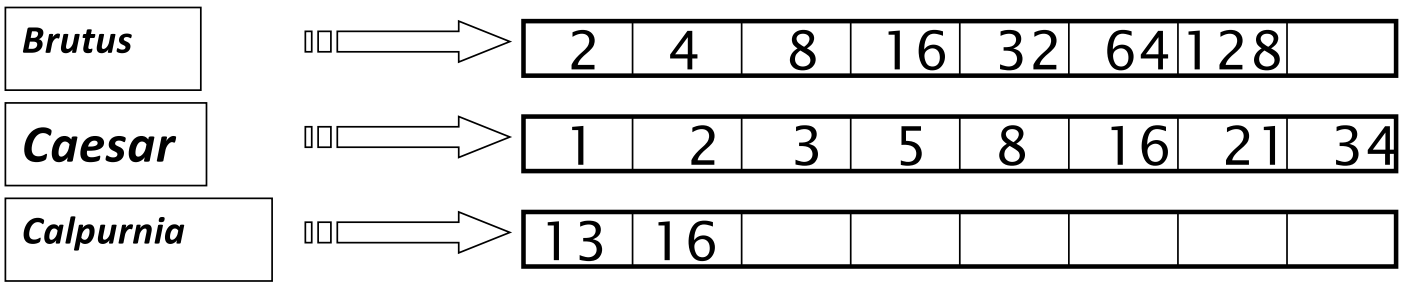


Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

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

This is why we kept document freq. in dictionary



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

More general optimization

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

Phrase queries

- We want to be able to answer queries such as “parthenope university” as a phrase
- Thus, the sentence “*I went to university at Parthenope*” is not a match
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that work
  - Many more queries are implicit phrase queries
- For this, it no longer suffices to store only *<term : docs> entries*

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example, the text "*Friends*, *Romans*, *Countrymen*" would generate the biwords
  - *friends romans*
  - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-words phrase query-processing is now immediate

Longer phrase queries

- Longer phrases can be processed by breaking them down
  - *parthenope university centro direzionale* can be broken into the Boolean query on biwords:

    parthenope university AND university centro AND centro direzionale

- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the four-words phrase

Can have false positives!

Issues for Biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them

- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Solution 2: Positional indexes

- In the postings, store, for each term the position(s) in which tokens of it appear:

<term, number of docs containing term;
doc1: position1, position2 … ;
doc2: position1, position2 … ;
etc.>

Positional index example

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

<***to***: 993427;
1,6: 7, 18, 33, 72, 86, 231;
2,2: 3, 149;
4,5: 17, 191, 291, 430, 434;
5,2: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

Processing a phrase query

- Extract inverted index entries for each distinct term: *to*, *be*, *or*, *not*
- Merge their doc:position lists to enumerate all positions with "*to be or not to be*"
- We look for
  1. Documents that contain both terms
  2. Occurrence of *be* with a token index higher than a position of *to*
  3. Another occurrence for each word with token index 4 higher than the first occurrence
  - to: 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - be: 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...

Positional index size

- A positional index expands postings storage substantially
  - Even though indices can be compressed
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries
  - … whether used explicitly or implicitly in a ranking retrieval system

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - Books, even some epic poems … easily 100,000 terms
- Consider a term with frequency 0.001%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text
    - Caveat: all of this holds for “English-like” languages

Combination schemes

- These two approaches can be profitably combined
  - For common queries ("*Michael Jackson*", "*Britney Spears*") it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like "*The Who*"
      - Individual words common but the desired phrase is comparatively rare
        - Candidates for a phrase index
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

IR vs. databases: Structured vs unstructured data

- Structured data tends to refer to information in “tables”
- Typically allows numerical range and exact match (for text) queries, e.g.,
  - *Salary < 60000 AND Manager = Smith*

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

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*
    - … to say nothing of linguistic structure
- Facilitates "semi-structured" search such as
    - *Title* contains data AND *Bullets* contain search
- Or even
    - *Title* is about <u>Object Oriented Programming</u> AND *Author* something like <u>stro*rup</u>
    - where * is the wild-card operator