Introduction to
Information Retrieval

Dr. Mustafa Jarrar

Sina Institute, University of Birzeit

mjarrar@birzeit.edu
www.jarrar.info

Mustafa Jarrar: Online Courses – Watch and Download Slides

Reference:
Mustafa Jarrar: Lecture Notes on Conceptual Analyses
University of Birzeit, Palestine, 2015

Online Course

Watch Do ad Slides

Mustafa Jarrar

Birzeit University, Palestine

mjarrar@birzeit.edu

Watch this lecture and download the slides from

http://jarrar-courses.blogspot.com/2011/11/artificial-intelligence-fall-2011.html

Outline

- Part 1: Information Retrieval Basics
- Part 2: Term-document incidence matrices
- Part 3: Inverted Index
- Part 4: Query processing with inverted index
- Part 5: Phrase queries and positional indexes

Keywords: Natural Language Processing, Information Retrieval, Precision, Recall, Inverted Index , Positional Indexes, Query processing, Merge Algorithm, B iwords, P hrase queries

اللسانيات احلاسوبية , استرجاع املعلومات ,الدقة , استعالم , تطبيقات لغوية , املعاجلة اآللية للغات الطبيعية

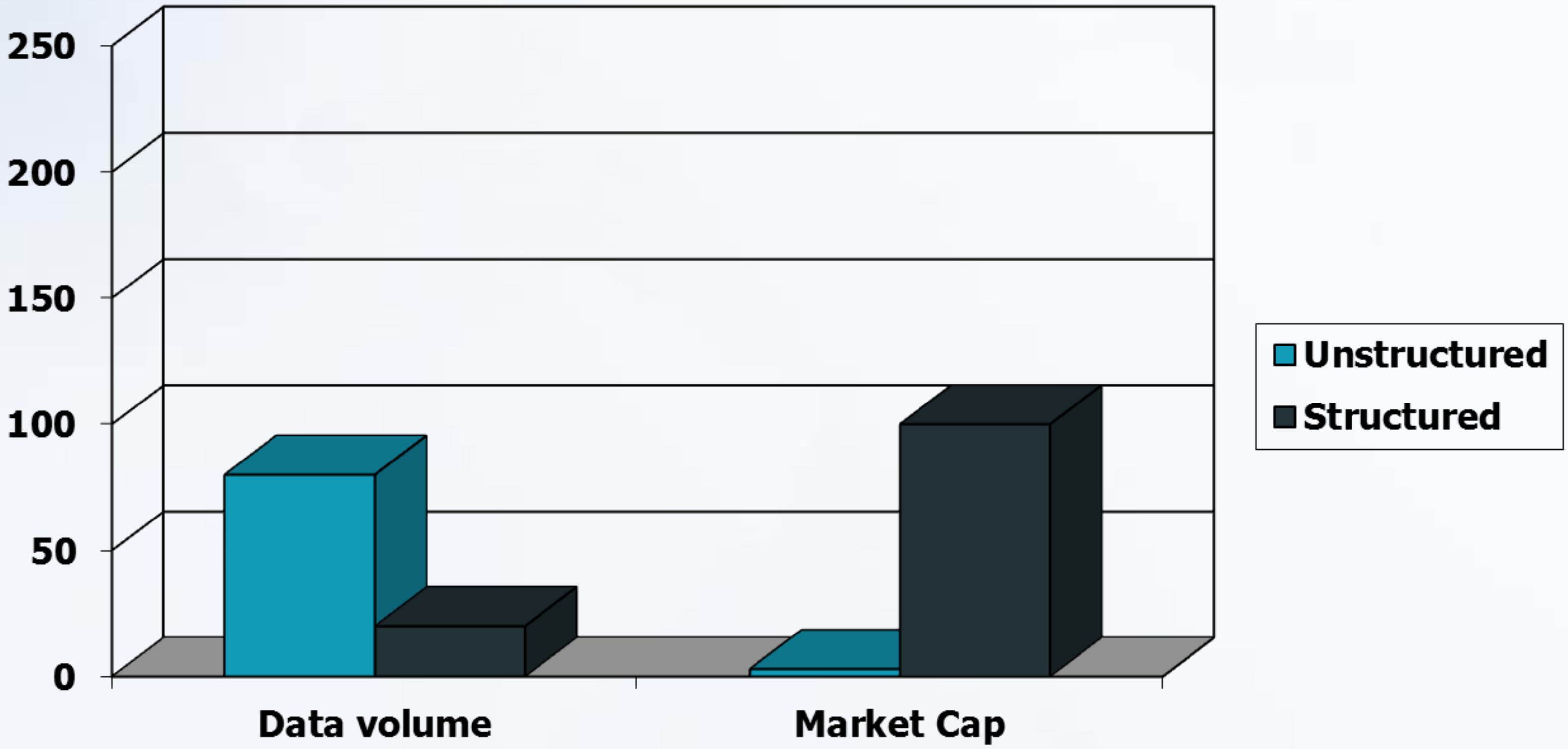
Acknowledgement: This lecture is largely based on Chris Manning online course on NLP, which can be accessed at http://www.youtube.com/watch?v=s3kKlUBa3b0, [1]

Information Retrieval

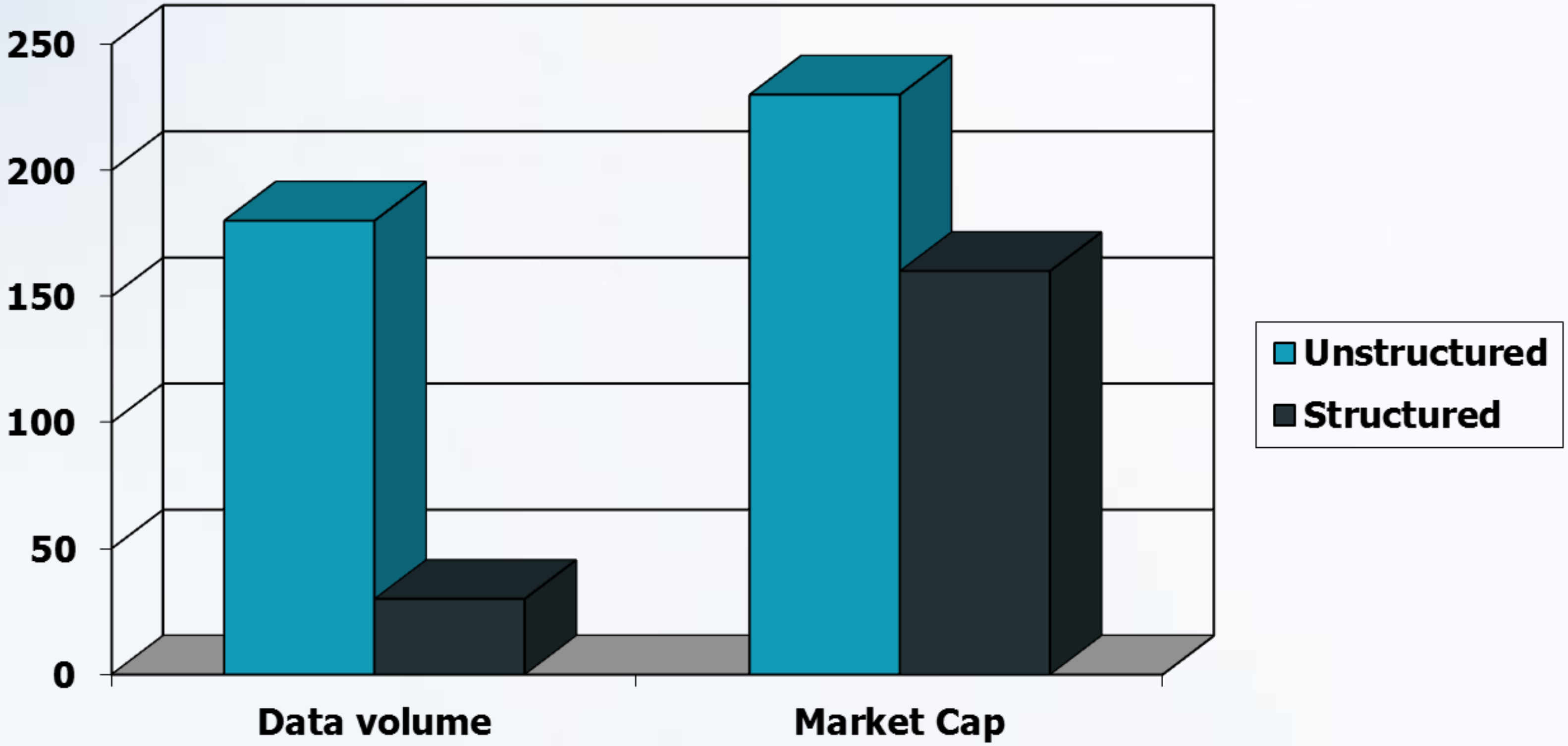
Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

- 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 (database) data in the mid-nineties



Unstructured (text) vs. structured (database) data later



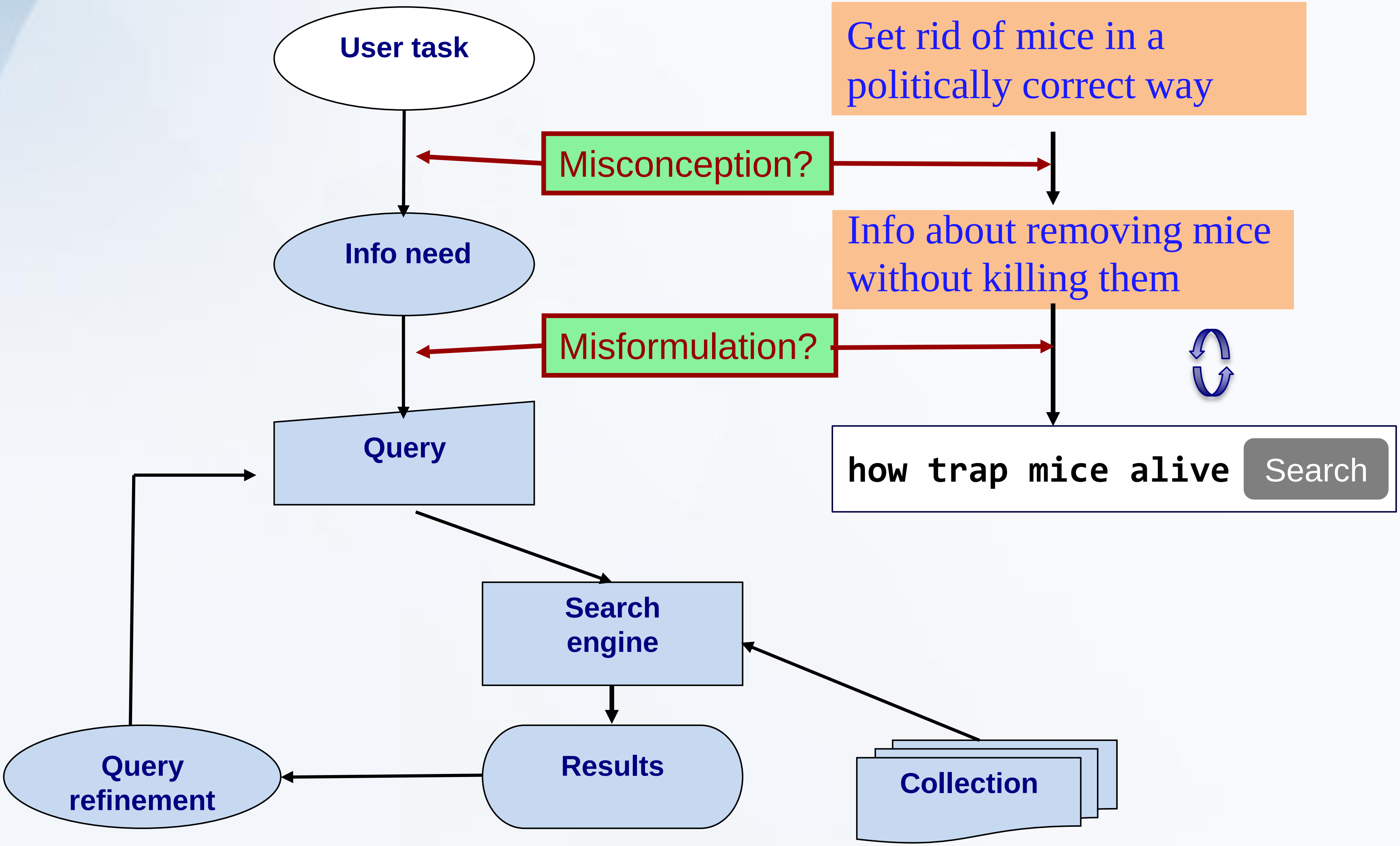
Basic Assumptions of Information Retrieval

Collection: A set of documents

- Assume it is a static collection (but in other scenarios we may need to add and delete documents).

Goal: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task.

Classic Search Model



User task

Misconception?

Info need

Misformulation?

Query

Search engine

Results

Query refinement

Collection

Get rid of mice in a politically correct way

Info about removing mice without killing them

how trap mice alive

How good are the retrieved docs?

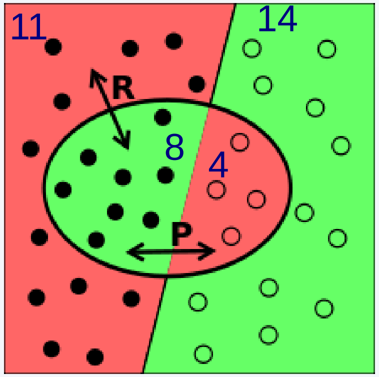
- ***Precision*** : Fraction of retrieved docs that are relevant to the user's information need (نسبة الصحيح من بين ما تم استرجاعة).

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

- ***Recall*** : Fraction of relevant docs in collection that are retrieved

نسبة الصحيح المسترجع من بين جميع الصحيح

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$



In this figure the relevant items are to the left of the straight line while the retrieved items are within the oval. The red regions represent errors. On the left these are the relevant items not retrieved (false negatives), while on the right they are the retrieved items that are not relevant (false positives).

P= 8/12 = 66%,   R=8/14 = 57%

Are these measures are always useful as in case of huge collections?
Ranking becomes more important sometimes.

Introduction to Information Retrieval

- Part 1: Information Retrieval Basics
- ⇨ Part 2: Term-document incidence matrices
- Part 3: Inverted Index
- Part 4: Query processing with inverted index
- Part 5: Phrase queries and positional indexes

Unstructured Data in 1620

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)

– *NOT* ***Calpurnia*** is non-trivial

– Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible

– Ranked retrieval (best documents to return)

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

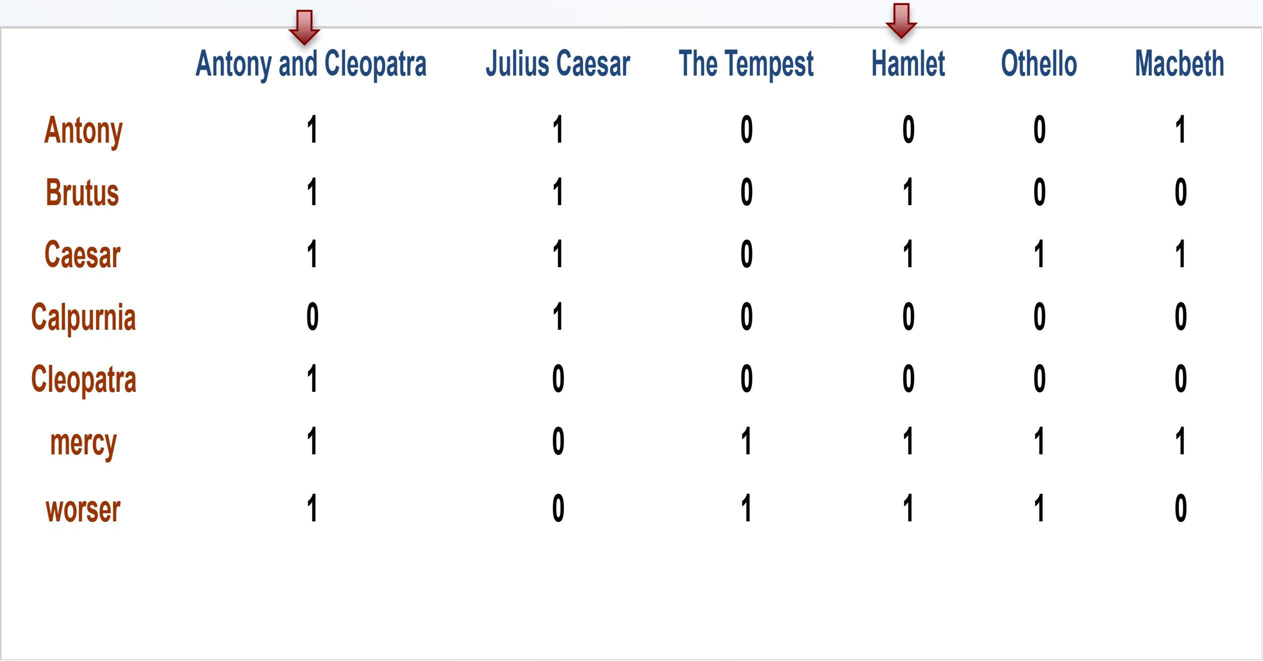
1 if play contains word, 0 otherwise

Incidence Vectors

So we have a 0/1 vector for each term.

<u>To answer query</u>: 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.

With 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.

We Can’t Build this Matrix!

500K x 1M matrix has half-a-trillion 0’s and 1’s.

But it has no more than one billion 1’s. (=1000 words x 1 M doc.)

➔ matrix is extremely sparse.

What’s a better representation?

- We only record the 1 positions.

Introduction to Information Retrieval

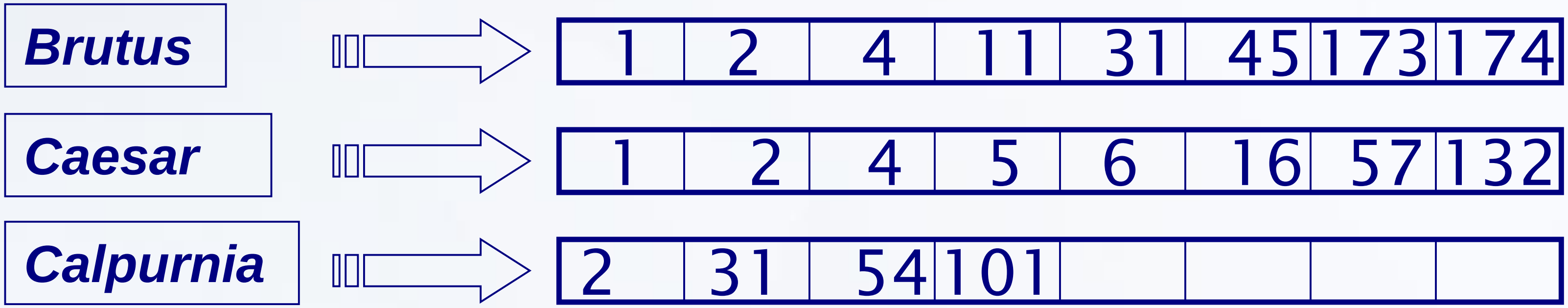
- Part 1: Information Retrieval Basics
- Part 2: Term-document incidence matrices
- ➡ Part 3: Inverted Index
- Part 4: Query processing with inverted index
- Part 5: Phrase queries and positional indexes

What is an Inverted Index?

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 used fixed-size arrays for this?



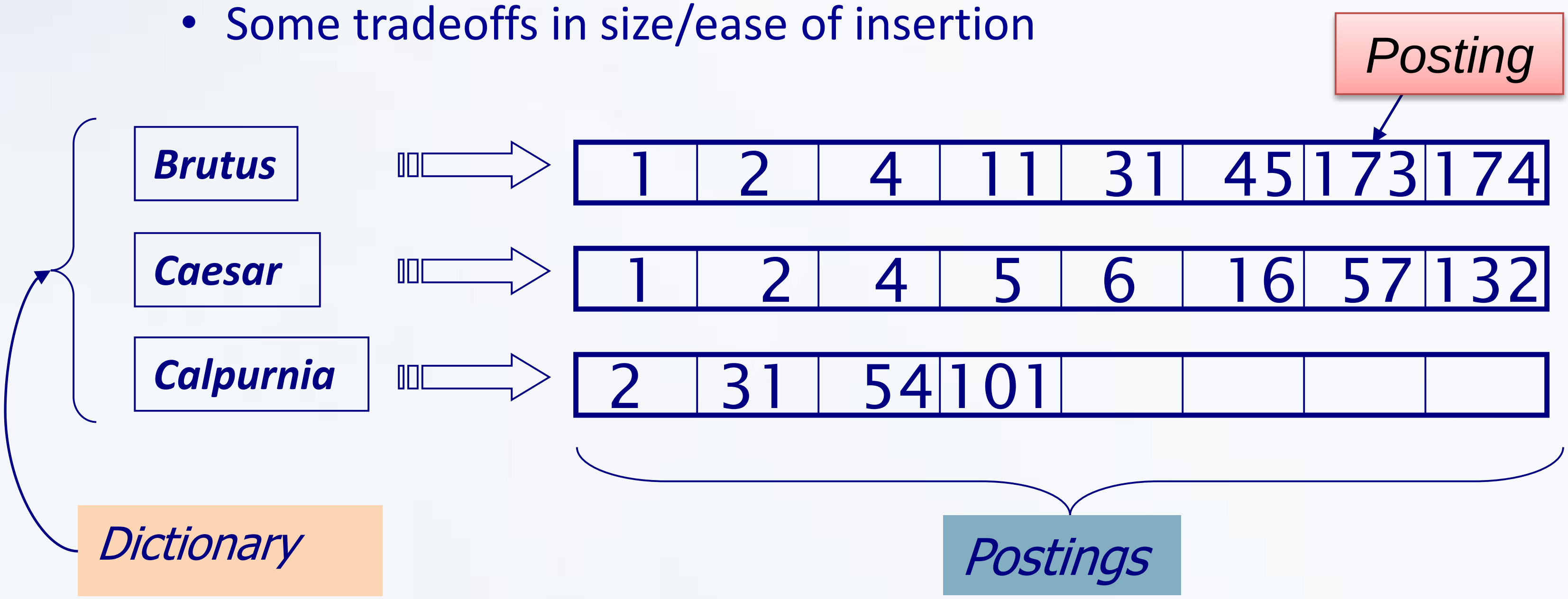
Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

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

Inverted index

We need variable-size postings lists

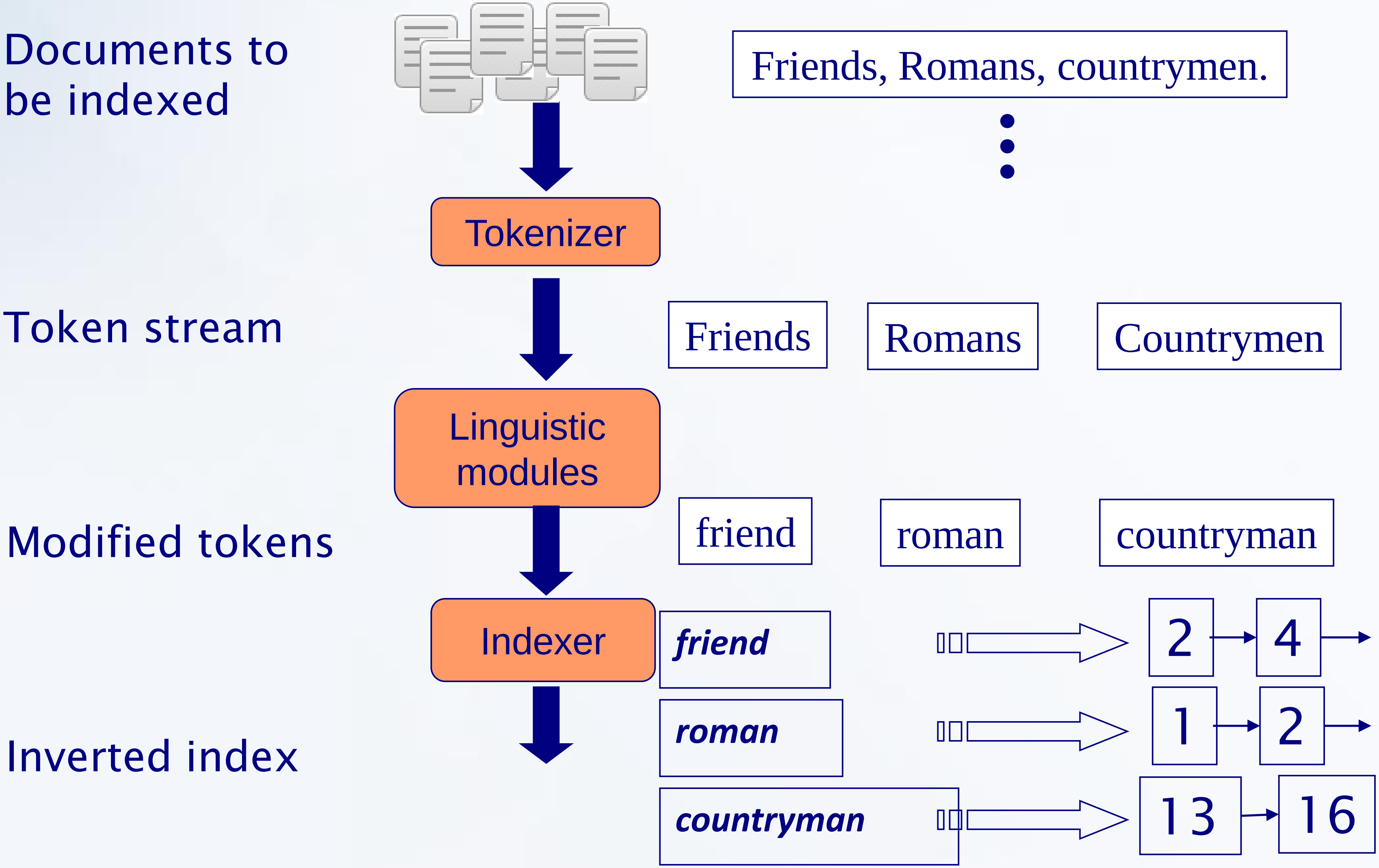
- On disk, a continuous run of postings is normal and best
- In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion



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

Sorted by docID (more later on why).

Inverted Index Construction



Documents to be indexed

Friends, Romans, countrymen.

Tokenizer

Token stream

Friends | Romans | Countrymen

Linguistic modules

Modified tokens

friend | roman | countryman

Indexer

Inverted index

friend → 2 → 4

roman → 1 → 2

countryman → 13 → 16

Initial Stages of Text Processing

Tokenization

- Cut character sequence into word tokens (=what is a word!)
  - 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 have different forms of a root to match
  - ***authorize***, ***authorization***

Stop words

- We may omit very common words (or not)
  - ***the, a, to, of, over, between, his, him***

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

Indexer Steps: Sort

Sort by terms

- And then docID

Core indexing step

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

→

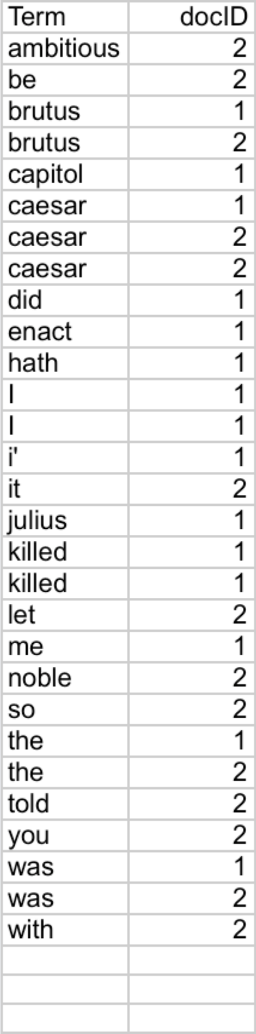
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer Steps: Dictionary & Postings

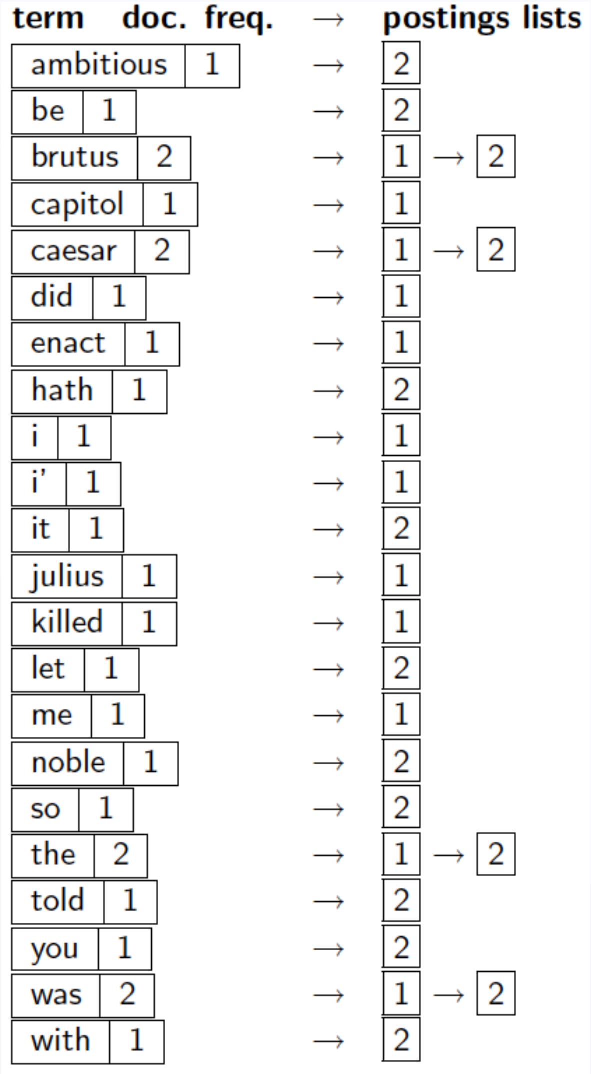
Multiple term entries in a single document are merged.

Split into Dictionary and Postings

Doc. frequency information is added.



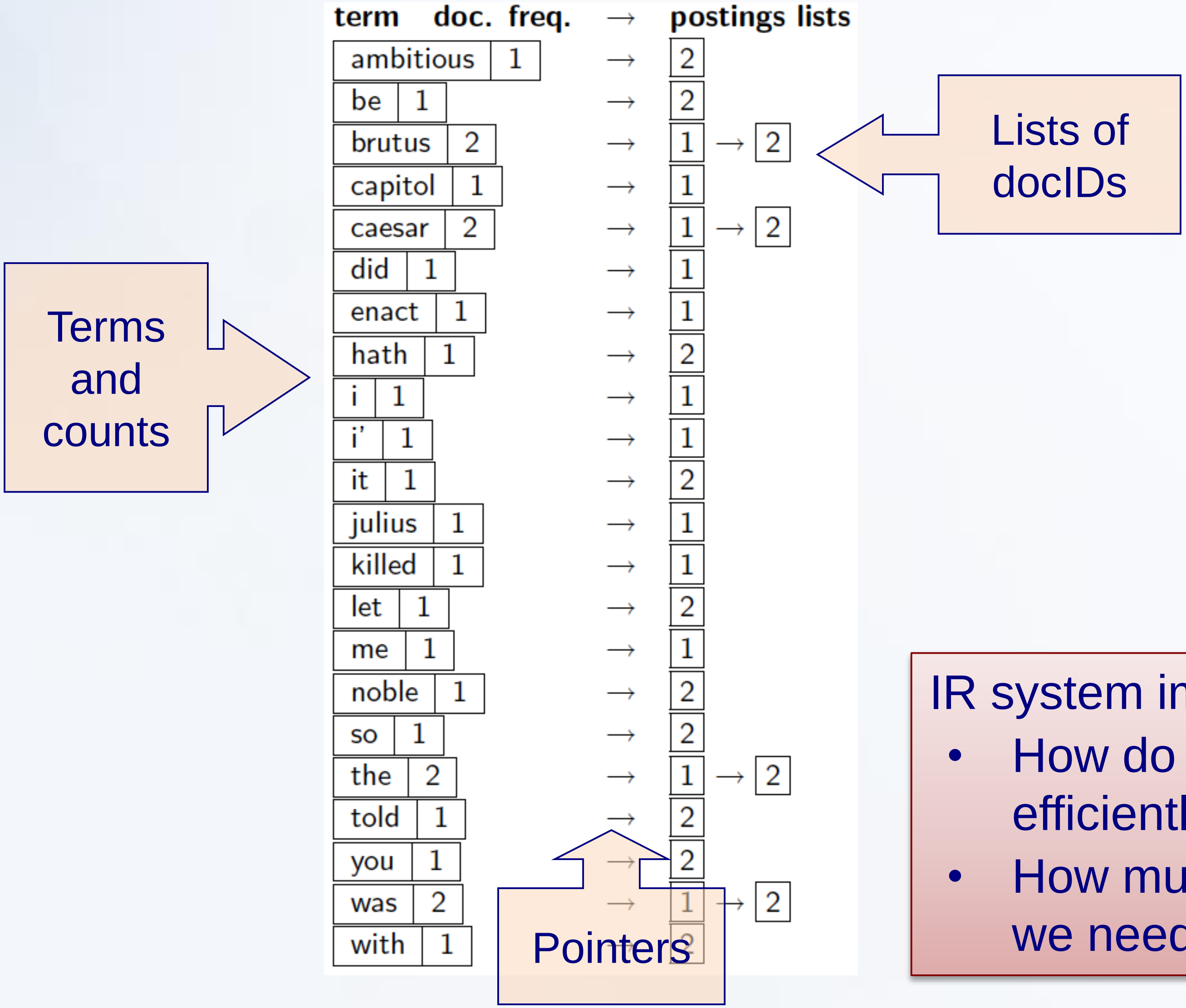
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



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

Why frequency?

Where do we pay in storage?



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

IR system implementation

- How do we index efficiently?
- How much storage do we need?

Introduction to Information Retrieval

- Part 1: Information Retrieval Basics
- Part 2: Term-document incidence matrices
- Part 3: Inverted Index

➡ Part 4: Query processing with inverted index

- Part 5: Phrase queries and positional indexes

Query processing: AND

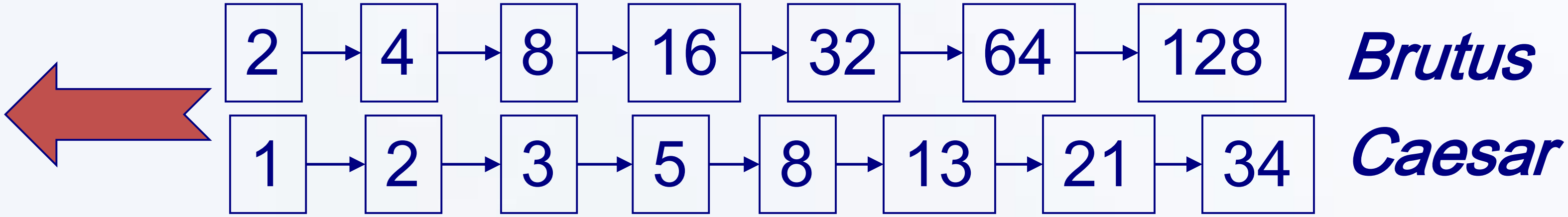
Consider processing the query:

Brutus *AND* ***Caesar***

– Locate ***Brutus*** in the Dictionary;
  - Retrieve its postings.

– Locate ***Caesar*** in the Dictionary;
  - Retrieve its postings.

– "Merge" the two postings (intersect the document sets):

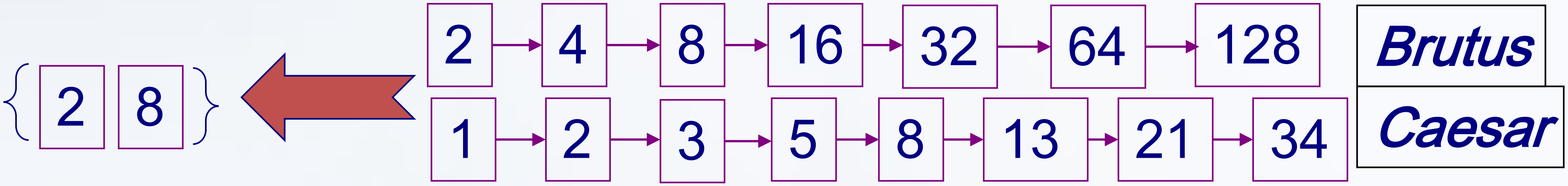


2 → 4 → 8 → 16 → 32 → 64 → 128 ***Brutus***

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 ***Caesar***

The Merge

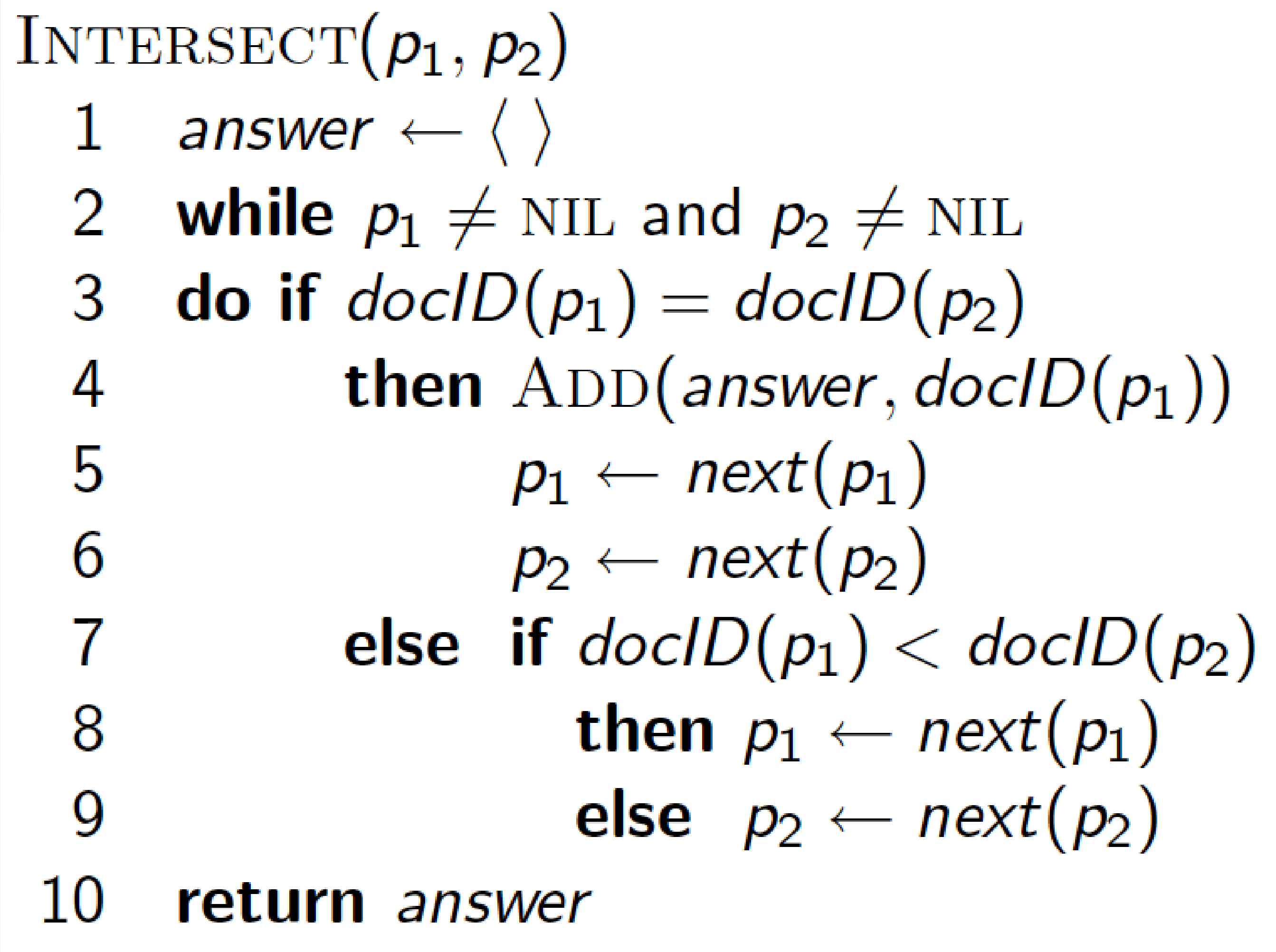
Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

Crucial: postings sorted by docID.

Intersecting two postings lists (a “merge” algorithm)



```
INTERSECT($p_1$, $p_2$)
answer $\leftarrow \langle\ \rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then ADD($answer$, $docID(p_1)$)
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
             then $p_1 \leftarrow next(p_1)$
             else $p_2 \leftarrow next(p_2)$
return answer
```

Introduction to Information Retrieval

- Part 1: Information Retrieval Basics
- Part 2: Term-document incidence matrices
- Part 3: Inverted Index
- Part 4: Query processing with inverted index
- ⇨ Part 5: Phrase queries and positional indexes

Phrase Queries

We want to be able to answer queries such as ***"stanford university"*** as a phrase

Thus the sentence *"I went to university at Stanford"* is not a match.

- The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works

For this, it no longer suffices to store only

<*term* : *docs*> entries

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-word phrase query-processing is now immediate.

Longer Phrase Queries

- Longer phrases can be processed by breaking them down

- ***stanford university palo alto*** can be broken into the Boolean query on biwords:
  - ***stanford university*** *AND* ***university palo*** *AND* ***palo alto***

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

→ This is not practical/standard solution!

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.>

Example: Positional index

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

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

For phrase queries, we use a merge algorithm recursively at the document level.

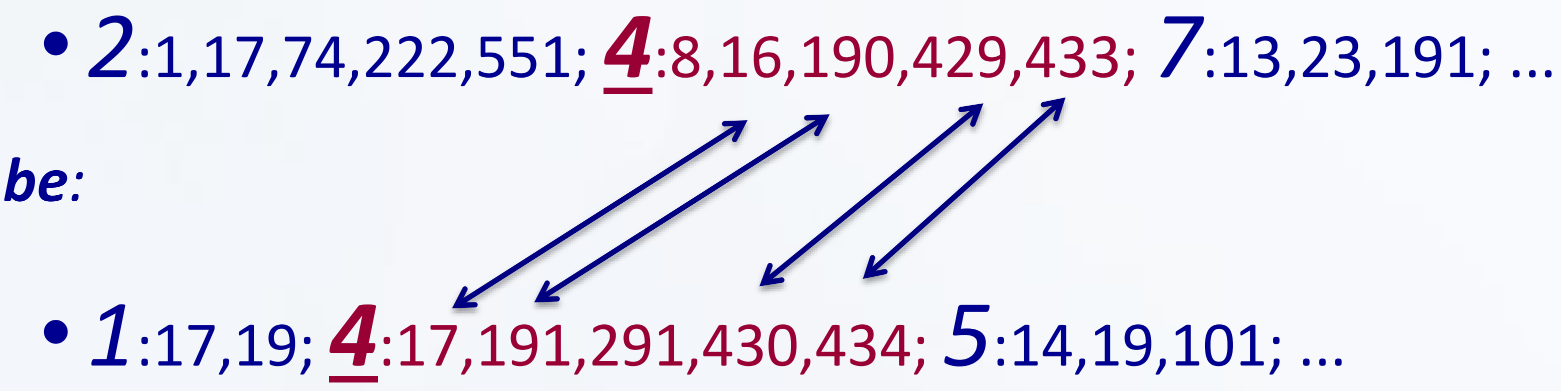
But we now need to deal with more than just equality.

Example: if we look for "Birzeit University" then if Birzeit in 87 then University should be in 88.

Processing Phrase Queries

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***".



- ***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; ...

Same general method for proximity searches

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.

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 (both Solutions)

These two approaches can be profitably combined

- For particular phrases (***"Michael Jackson", "Britney Spears"***) it is inefficient to keep on merging positional postings lists
  - Even more so for phrases like ***"The Who"***

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

Project (Search Engine)

Building an Arabic Search Engine based on a positional inverted index. A corpus of at least 10 documents containing at least 10000 words should be used to test this engine.

The engine should support single, multiple word, as phrase queries.

The inverted index should take into account Arabic stemming and stop words.

The results page will list the titles of all relevant documents in a clickable manner.

This project is a continuation of the previous project, thus students are expected to also allow users to "auto complete" their search queries.

Deadline: 9/10/2014

References

[1] Dan Jurafsky: From Languages to Information notes
http://web.stanford.edu/class/cs124