Information Retrieval

Faculty: Prof. U.S. Tiwary

Study Material Available on:
https://silp.iiita.ac.in/

Google Classroom Code:

Information Retrieval

Lecture 1: Introduction

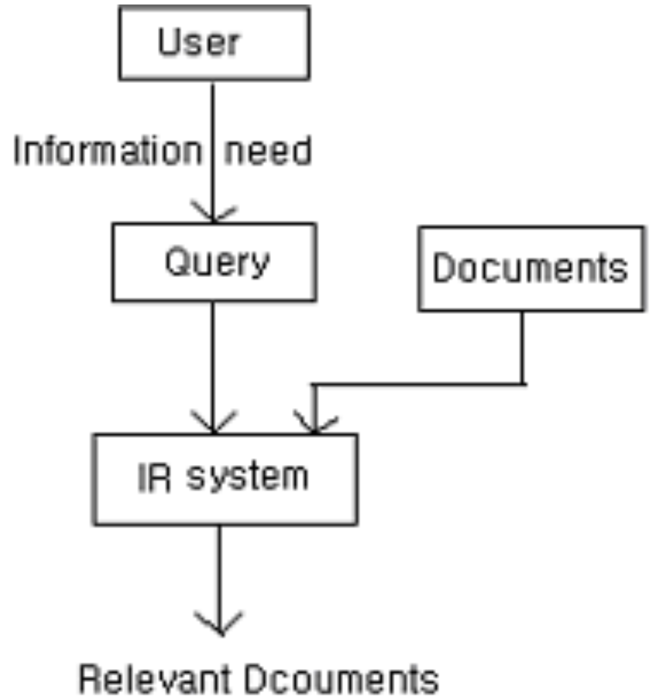
Information Retrieval : Intro

- Information retrieval (IR) deals with the organization, storage, retrieval and evaluation of information relevant to user's need (query).
- Query written in a natural language.
- The retrieval system responds by retrieving document that seems relevant to the query

Information Retrieval

- Traditionally it has been accepted that information retrieval system does not return the actual information but the documents containing that information in a large corpus.
- ‘An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of her inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to her request.’

Information Retrieval Process



Basic Information Retrieval Process

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

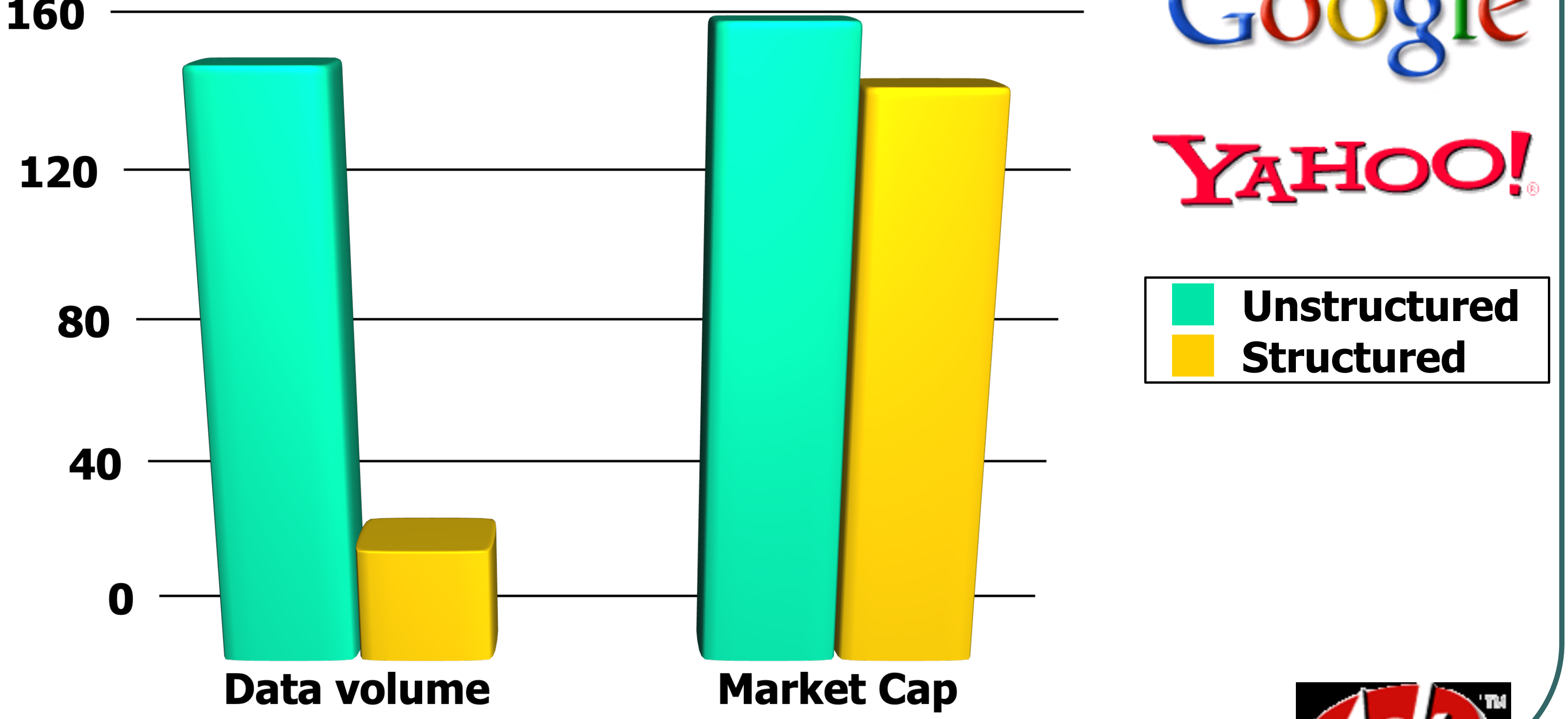
- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
  - *Title* contains <u>data</u> AND *Bullets* contain <u>search</u>

… 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
- The focus of XML search.

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



IR: An Example

Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?

- Simplest approach is to grep all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
  - 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)

How to avoid linear scanning ?

➔ Index the documents in advance

Indexing

- The process of transforming document text to some representation of it is known as *indexing*.
- Different index structures might be used. One commonly used data structure by IR system is ***inverted index***.

Information Retrieval Model

An IR model is a pattern that defines several aspects of retrieval procedure, for example,

- how the documents and user's queries are represented
- how system retrieves relevant documents according to users' queries &
- how retrieved documents are ranked.

IR Model

- An IR model consists of

  - a model for documents

  - a model for queries and

  - a matching function which compares queries to documents.

  - a ranking function

Classical IR Model

IR models can be classified as:

- Classical models of IR
- Non-Classical models of IR
- Alternative models of IR

Classical IR Model

- based on mathematical knowledge that was easily recognized and well understood
- simple, efficient and easy to implement
- The three classical information retrieval models are:

    -Boolean

    -Vector and

    -Probabilistic models

Non-Classical models of IR

Non-classical information retrieval models are based on principles other than similarity, probability, Boolean operations etc. on which classical retrieval models are based on.

information logic model, *situation theory model* and *interaction model*.

Alternative IR models

- Alternative models are enhancements of classical models making use of specific techniques from other fields.

Example:

Cluster model, fuzzy model and latent semantic indexing (LSI) models.

Information Retrieval Model

- The actual text of the document and query is not used in the retrieval process. Instead, some representation of it.
- Document representation is matched with query representation to perform retrieval
- One frequently used method is to represent document as a set of index terms or keywords

Boolean model

- the oldest of the three classical models.
- is based on Boolean logic and classical set theory.
- represents documents as a set of keywords, usually stored in an inverted file.

Boolean model

- Users are required to express their queries as a boolean expression consisting of keywords connected with boolean logical operators (AND, OR, NOT).
- Retrieval is performed based on whether or not document contains the query terms.

Boolean model

Given a finite set

$T = \{t1, t2, ...,ti,...,tm\}$

of index terms, a finite set

$D = \{d1, d2, ...,dj,...,dn\}$

of documents and a boolean expression in a normal form - representing a query Q as follows:

$$Q = \wedge(\vee\, \theta_i), \theta i \in \{t_i, \neg\, t_i\}$$

Boolean model

1. The set R_i of documents are obtained that contain or not term t_i:

$$R_i = \{ d_j \mid \theta i \in d_j \}, \quad \theta i \in \{t_i, \neg t_i\},$$

where $\neg t_i \in d_j$ means $t_i \notin d_j$

2. Set operations are used to retrieve documents in response to Q:

$$\cap R_i$$

Basics of Boolean IR model

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- Document collection: A collection of Shakespeare's work

Binary Term-document matrix

	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

- 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.

Answers to query

- Antony and Cleopatra, Act III, Scene ii

  ....

  ....

- Hamlet, Act III, Scene ii

  ....

  ....

- Boolean retrieval model answers any query which is in the form of Boolean expression of terms.

Bigger corpora

- 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 <u>*distinct*</u> terms among these.

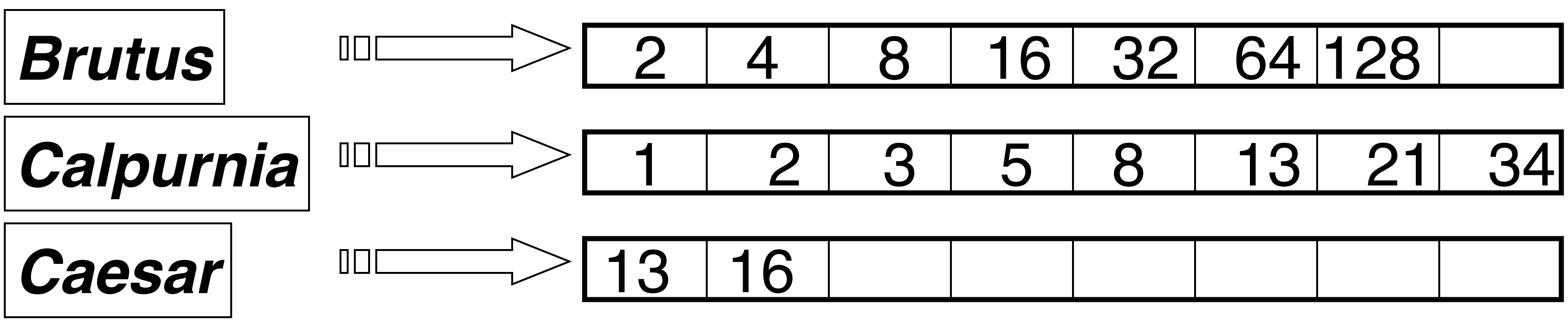
Can't build the matrix

- 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*.

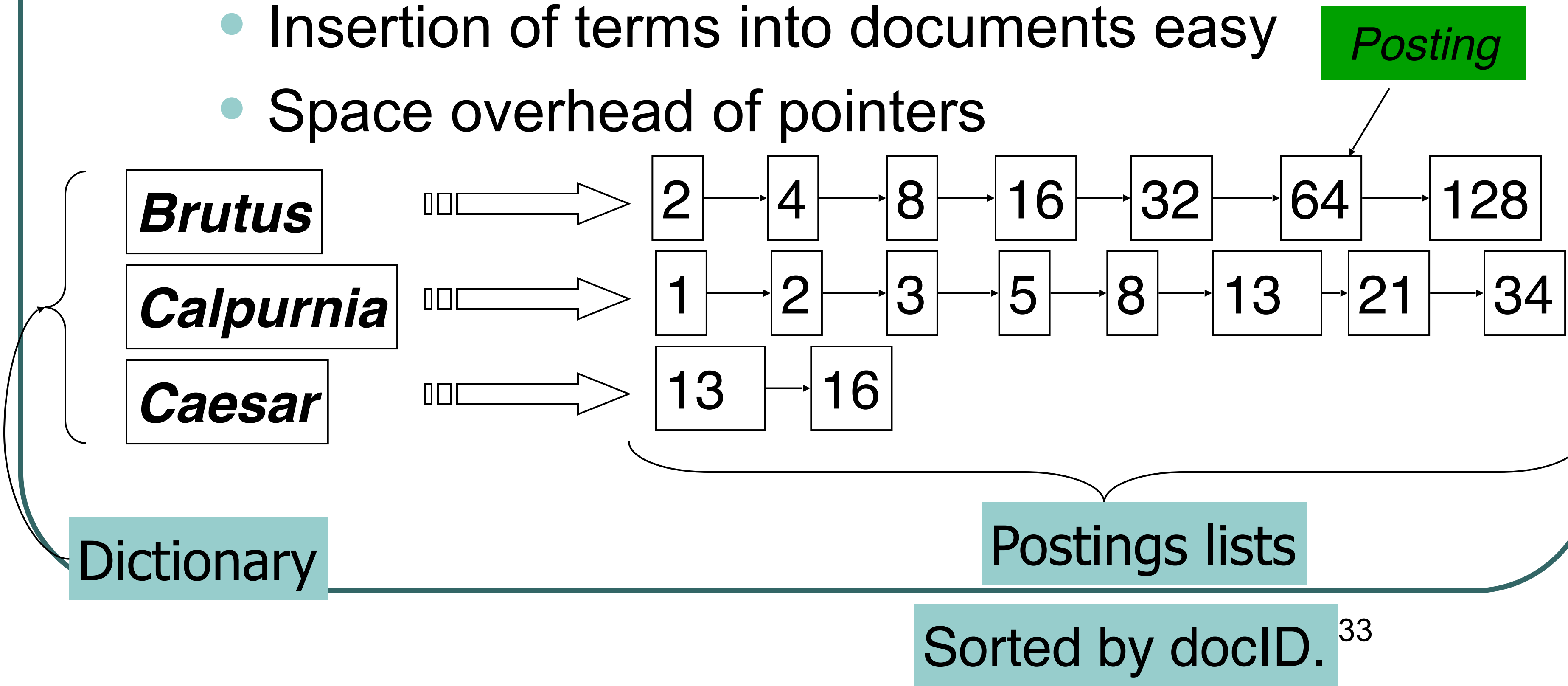


  - we can use an array or a list.

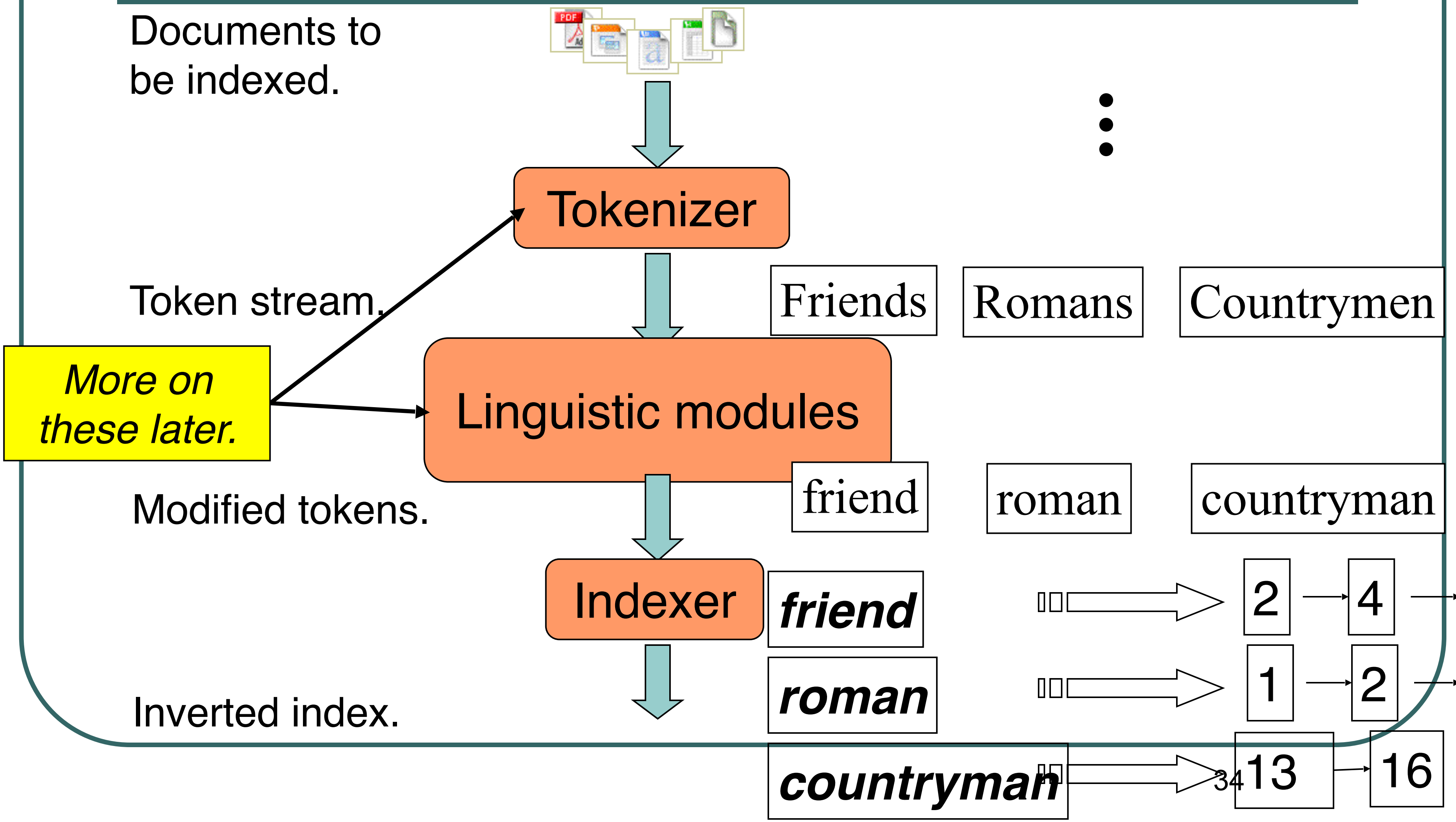
What happens if the word ***Caesar*** is added to document 14?

Inverted index

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



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(**<u>Core indexing step.)</u>**.

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.

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

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

Pointers

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

The index we just built

- How do we process a query?

Query processing: AND

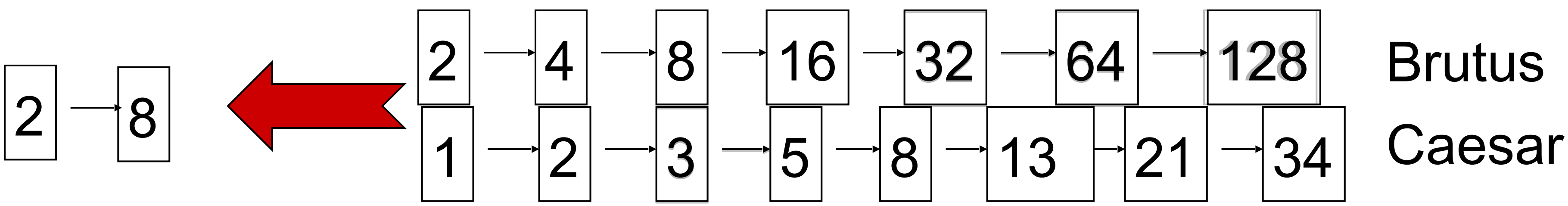
- Consider processing the query:
  Brutus *AND* ***Caesar***
  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:

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.

Merging Algorithm

Merge(p,q)

1 Start

2. Ans ← ()

3. While p<> nil and q <> nil do

    if p→docID = q→ docID

    then ADD(answer, p→docID) // add to result and advance pointers

    else if p→docID < q→docID

        then p← p→next

        else q← q→next

4. end {of algo}

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.

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

Merging: More general merges

Consider an arbitrary Boolean formula:

(Brutus *OR* ***Caesar)*** *AND NOT*

(Antony *OR* ***Cleopatra)***

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.

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***

Query Optimization

- How to organize the work of getting results for a query so that the amount of work is reduced.

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

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.

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.

Beyond term search

- Phrases?
  - ***Indian Institute of Information Technology***
- Proximity: Find ***Murty*** *NEAR* ***Infosys***.
  - Need index to capture position information in docs.
- Find documents with (*author* = ***Zufrasky***) *AND* (text contains ***Retrieval***).

What else to consider ?

- 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 give inclusion or exclusion of docs.
- Requires precise language for building query expressions (instead of free text)
- Often we want to rank/group results

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 ...
- 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
- …