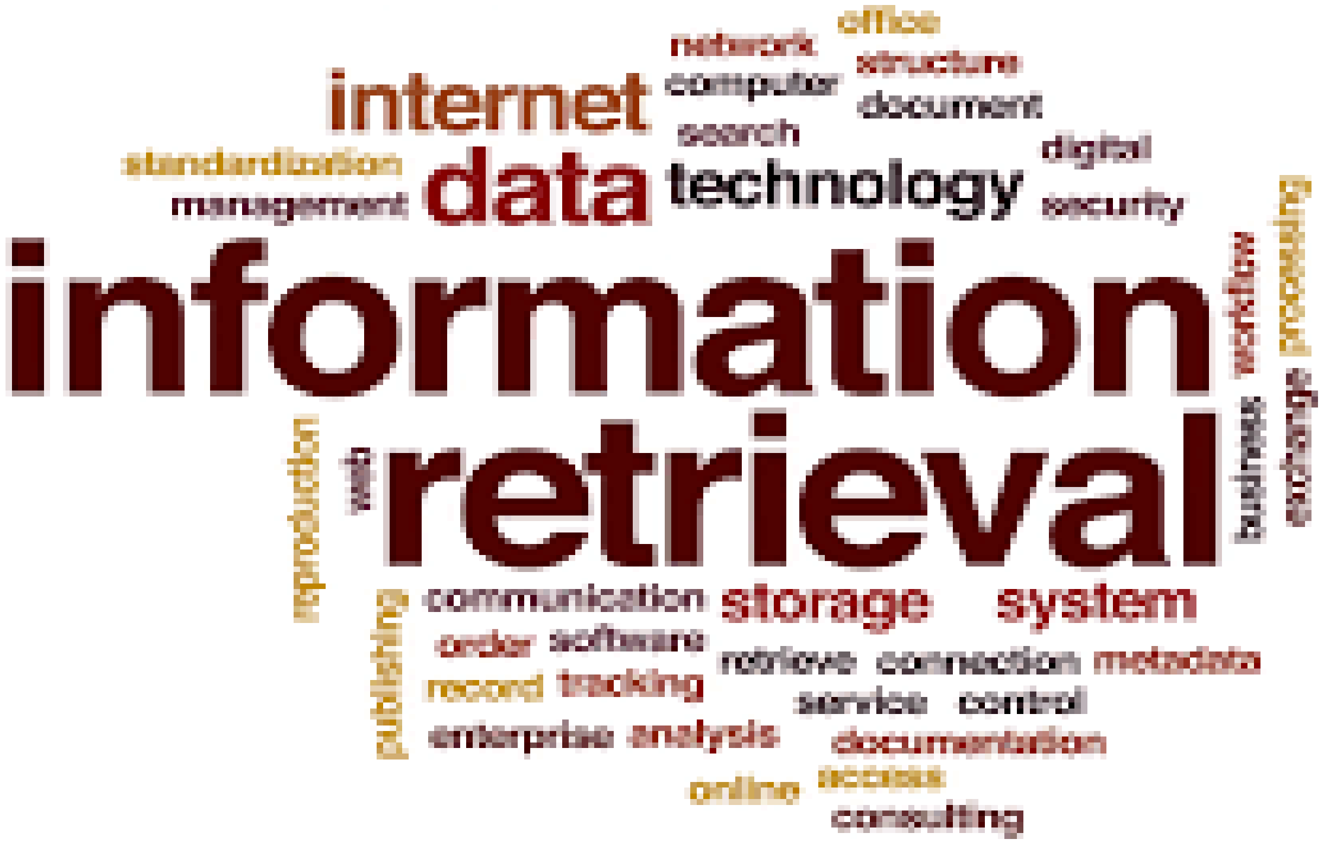


Information Retrieval

Instructor : Dr. Hanaa Bayomi Ali
Mail : h.mobarz @ fci-cu.edu.eg

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).

Information Retrieval

- What do we understand by **documents**? How do we decide what is a document and whatnot?
- What is an **information need**? What types of information needs can we satisfy automatically?
- What is a **large collection**? Which environments are suitable for IR

Basic assumptions of Information Retrieval

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

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

Why IR is hard?

IR is mostly about *relevance*

- Relevance is the core concept in IR, but nobody has a good definition
- Relevance = useful
- Relevance = topically related
- Relevance = new
- Relevance = interesting
- Relevance = ???
- However we still want *relevant information*

Why IR is hard?

- Information needs must be expressed as a query
  - But users don't often know what they want
- Problems
  - Verbalizing information needs
  - Understanding query syntax
  - Understanding search engines

Why this is hard?

- Documents/images/ video/speech/etc are complex. We need some representation
- Semantics
  - What do words mean?
- Natural language
  - How do we say things?
- ☹ Computers cannot deal with these easily

Semantics

Bank Note

Blood bank

River Bank

Bank

Information Retrieval Techniques

- Index Terms (Attribute) Selection:
  - Stop list
  - Word stem
  - Index terms weighting methods
- Terms $\times$ Documents Frequency Matrices
- Information Retrieval Models:
  - Boolean Model (incident matrix)
  - Vector Model
  - Probabilistic Model

Boolean Model

- Consider that index terms are either present or absent in a document
- As a result, the index term weights are assumed to be all binaries
- A query is composed of index terms linked by three connectives: not, and, and or
  - e.g.: car *and* repair, plane *or* airplane
- The Boolean model predicts that each document is either relevant or non-relevant based on the match of a document to the query

Example

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

Example

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**

	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

Boolean Models – Problems

- Very rigid: AND means all; OR means any.
- Difficult to express complex user requests.
- Difficult to control the number of documents retrieved.
  - *All* matched documents will be returned.
- Difficult to rank output.
  - *All* matched documents logically satisfy the query.
- Difficult to perform relevance feedback.
  - If a document is identified by the user as relevant or irrelevant, how should the query be modified?

Indexing Techniques

- **Inverted index**
  - Maintains two hash- or B+-tree indexed tables:
    - document_table: a set of document records <doc_id, postings_list>
    - term_table: a set of term records, <term, postings_list>
  - Answer query: Find all docs associated with one or a set of terms
  - + easy to implement
  - – do not handle well synonymy and polysemy, and posting lists could be too long (storage could be very large)

Example

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?

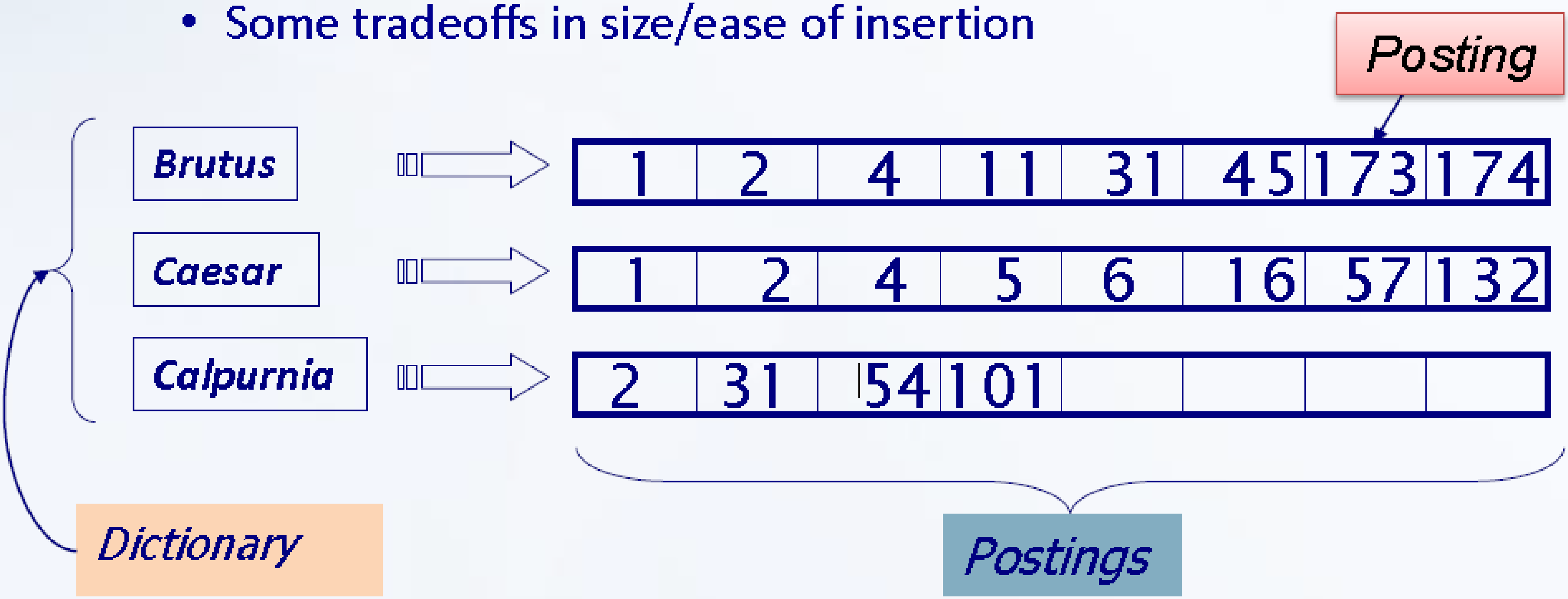
Brutus →	1	2	4	11	31	45	173	174
Caesar →	1	2	4	5	6	16	57	132
Calpurnia →	2	31	54	101				

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

Example

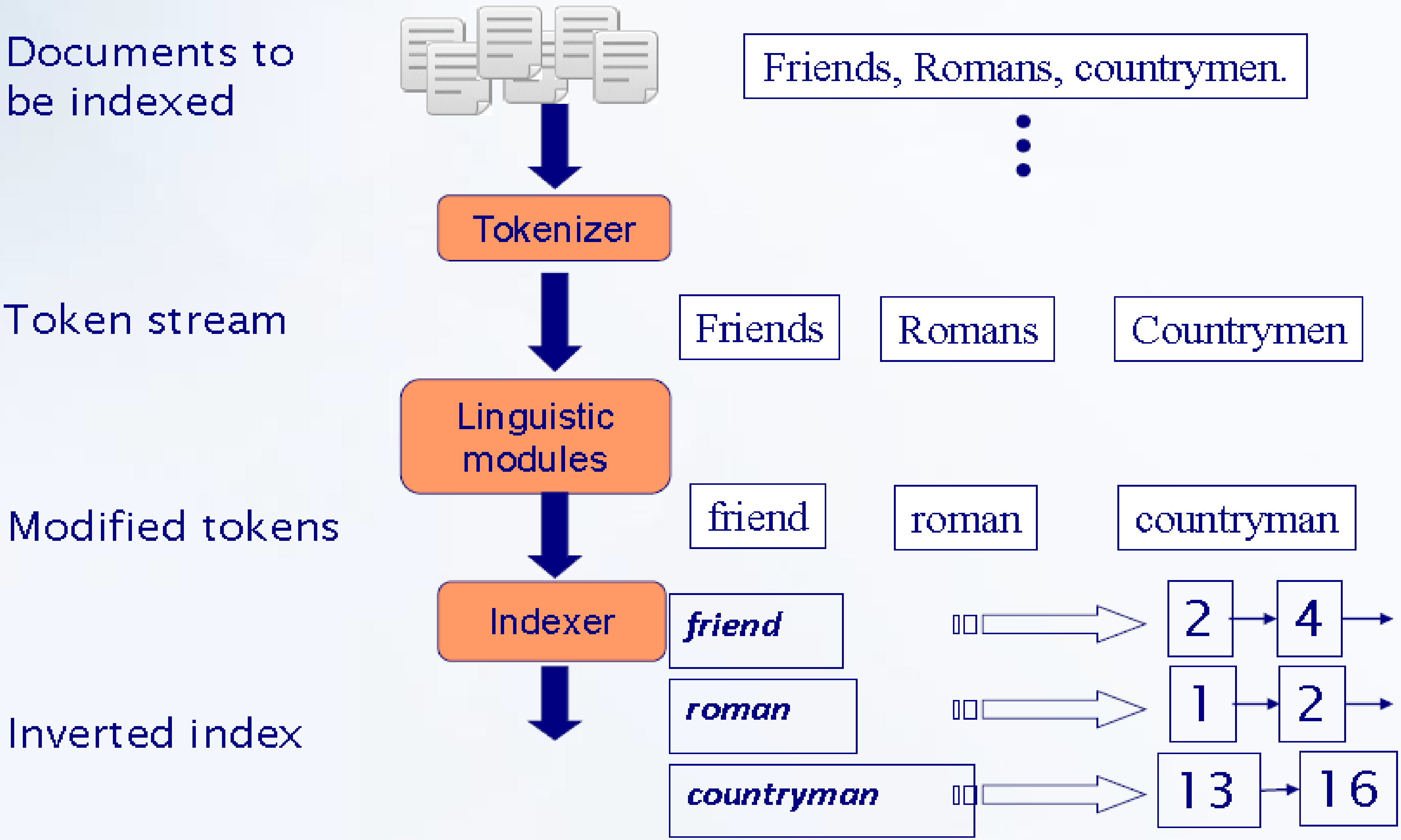
We need variable-size postings lists

- 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



Sorted by docID (more later on why).

Inverted index construction



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

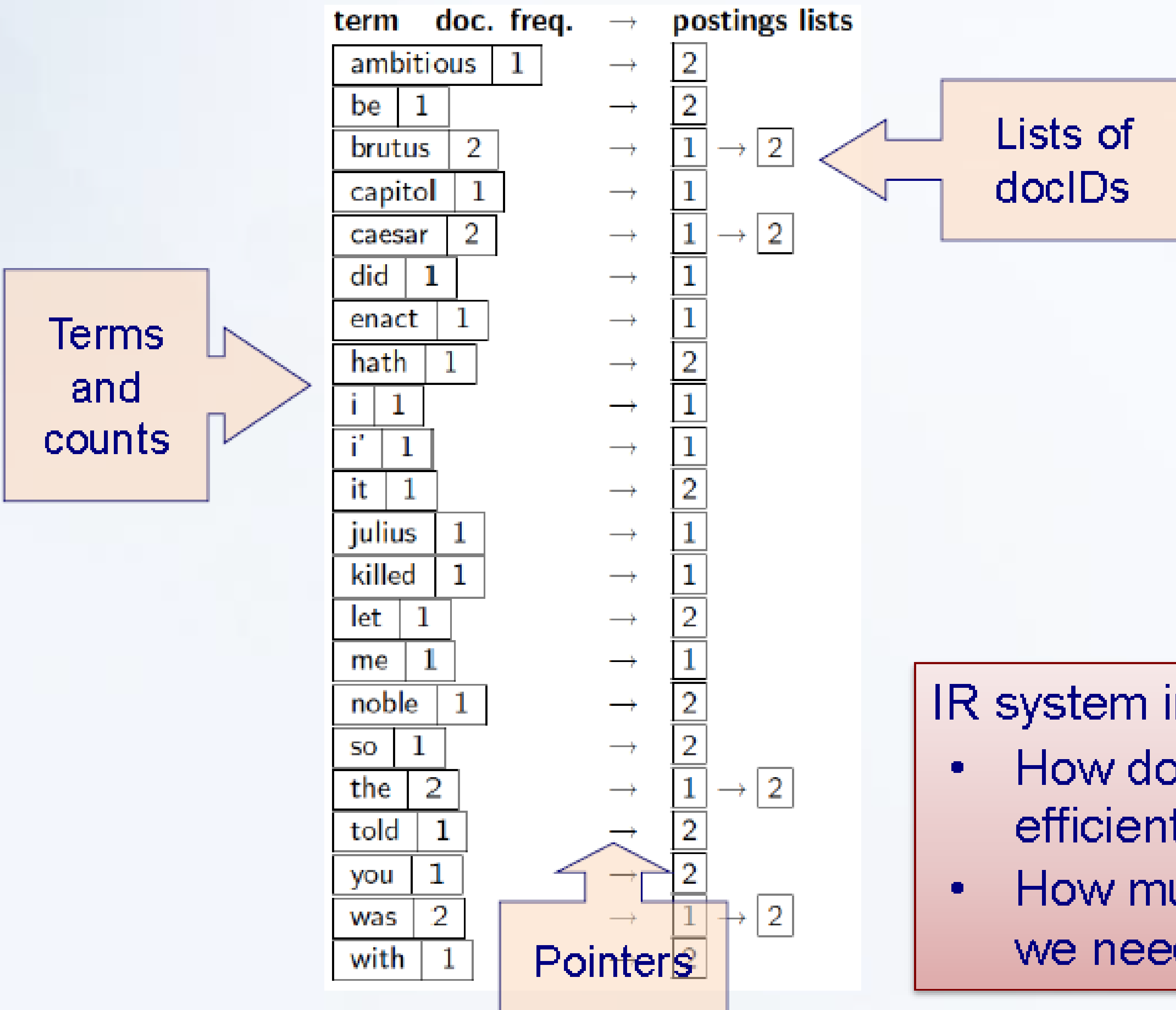
Doc. frequency information is added.

Why frequency?

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?



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

IR system implementation

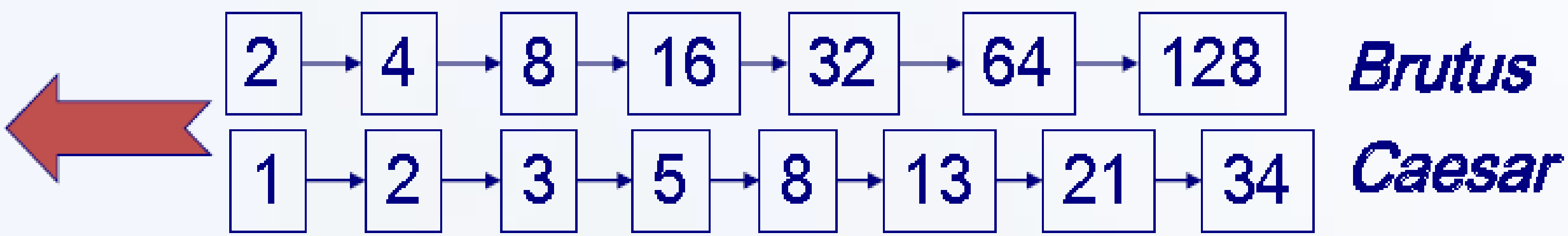
- How do we index efficiently?
- How much storage do we need?

Query processing: AND

Consider processing the query:

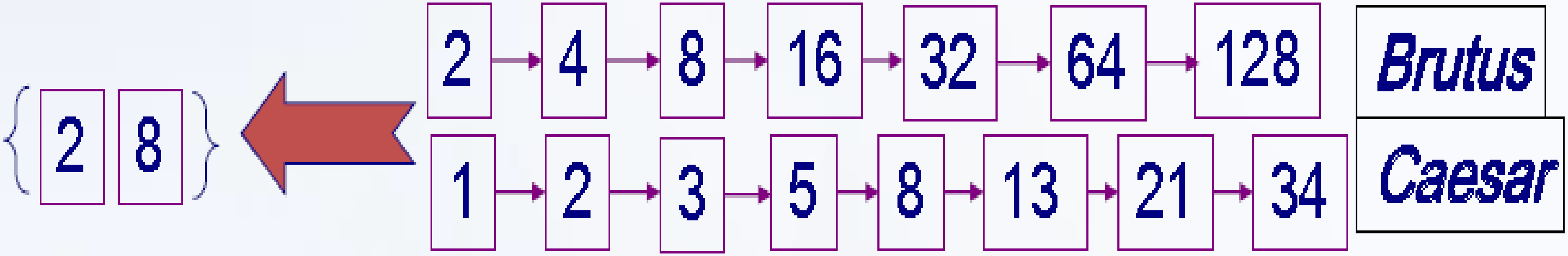
Brutus *AND* ***Caesar***

- Locate ***Brutus*** in the Dictionary;
  - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
  - Retrieve its postings.
- "Merge" the two postings (intersect the document sets):



The Merge

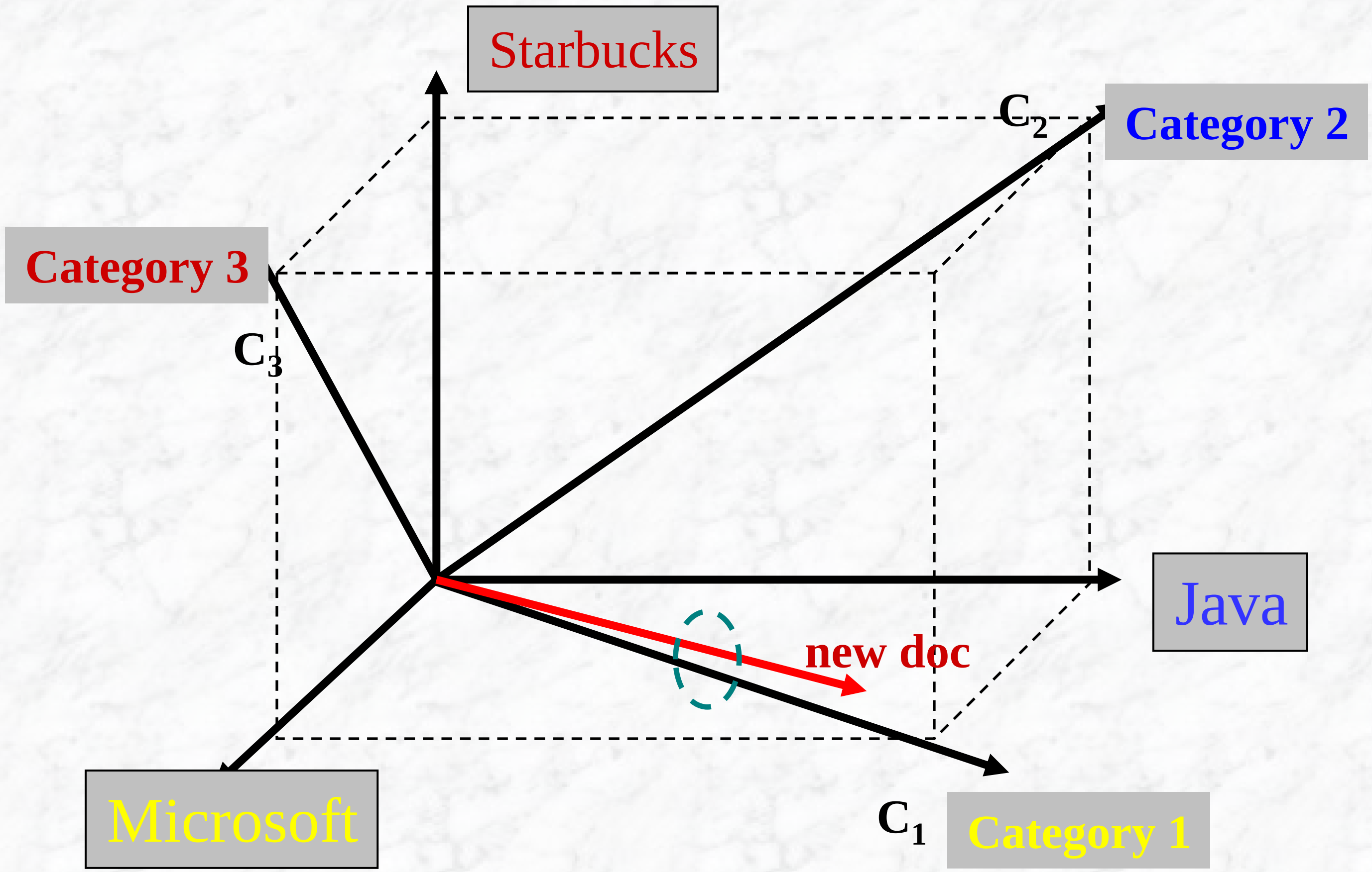
Walk through the two postings simultaneously, in time linear in the total number of postings entries



Vector Space Model

- Represent a doc by a term vector
  - Term: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define a N-dimensional space
  - Element of vector corresponds to term weight
  - E.g., d = (x_1,…,x_N), x_i is "importance" of term i
- New document is assigned to the most likely category based on vector similarity.

VS Model: Illustration



What VS Model Does Not Specify

- **How to select terms to capture “basic concepts”**
  - Word stopping
    - e.g. “a”, “the”, “always”, “along”
  - Word stemming
    - e.g. “computer”, “computing”, “computerize” => “compute”
- **How to assign weights**
  - Not all words are equally important: Some are more indicative than others
    - e.g. “algebra” vs. “science”
- How to measure the similarity

How to Assign Weights

- Two-fold heuristics based on frequency
  - TF (Term frequency)
    - More frequent ***within*** a document → more relevant to semantics
    - e.g., “query” vs. “commercial”
  - IDF (Inverse document frequency)
    - Less frequent ***among*** documents → more discriminative
    - e.g. “algebra” vs. “science”

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.

  f_{ij} = frequency of term *i* in document *j*

- May want to normalize *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of overall topic.

  df_i = document frequency of term i

  = number of documents containing term i

  idf_i = inverse document frequency of term i,

  $= \log_{10} (N / df_i)$

  (N: total number of documents)
- An indication of a term's *discrimination* power.
- Log used to dampen the effect relative to *tf*.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	0

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

There is one idf value for each term t in a collection.

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_{10} (N/ df_i)$$

- A **term occurring frequently in the document** but rarely in the rest of the collection is given **high weight**.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in R^{|V|}$

How to Measure Similarity?

- Given two document

$$D_i = (w_{i1}, w_{i2}, \cdots, w_{iN}) \qquad D_j = (w_{j1}, w_{j2}, \cdots, w_{jN})$$

- Similarity definition
  - dot product

$$Sim(D_i, D_j) = \sum_{t=i}^{N} w_{it} * w_{jt}$$

  - normalized dot product (or cosine)

$$Sim(D_i, D_j) = \frac{\sum_{t=i}^{N} w_{it} * w_{jt}}{\sqrt{\sum_{t=1}^{N} (w_{it})^2 * \sum_{t=1}^{N} (w_{jt})^2}}$$

Illustrative Example

doc1

text
mining
search
engine
text

doc2

travel
text

map
travel

doc3

government
president
congress

……

Sim(newdoc,doc1)=4.8*2.4+4.5*4.5

Sim(newdoc,doc2)=2.4*2.4

Sim(newdoc,doc3)=0

To whom is newdoc more similar?

	text	mining	travel	map	search	engine	govern	president	congress
IDF(faked)	2.4	4.5	2.8	3.3	2.1	5.4	2.2	3.2	4.3
doc1	2(4.8)	1(4.5)			1(2.1)	1(5.4)			
doc2	1(2.4)		2 (5.6)	1(3.3)					
doc3							1 (2.2)	1(3.2)	1(4.3)
newdoc	1(2.4)	1(4.5)							