

# Introduction to Information Retrieval

<http://informationretrieval.org>

## IIR 1: Boolean Retrieval

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.04.22

# Overview

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries
- 4 Course overview

# Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries
- 4 Course overview

# Definition of *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).

# Definition of *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).

# Definition of *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).

# Definition of *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).

# Definition of *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).



# Definition of *information retrieval*

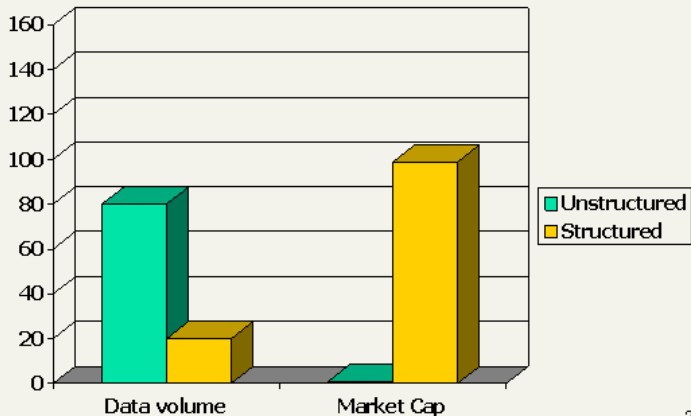
Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within **large collections** (usually stored on computers).

# Definition of *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).

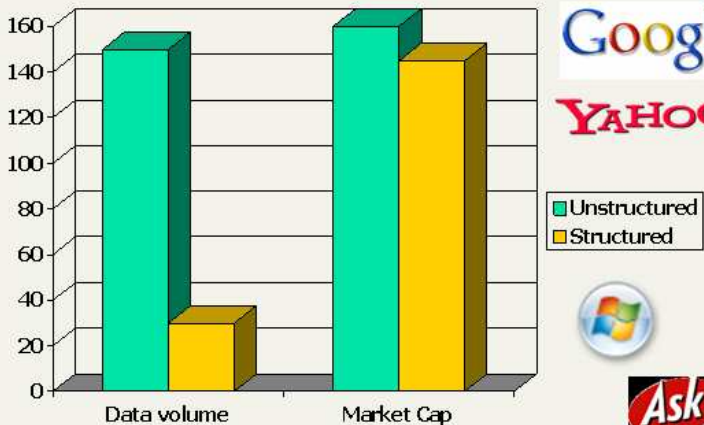
# Unstructured (text) vs. structured (database) data in 1996

---



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

---



Google™

YAHOO!®



# Boolean retrieval

- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The search engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?

# Outline

- 1 Introduction
- 2 Inverted index**
- 3 Processing Boolean queries
- 4 Course overview

# Unstructured data in 1650: Shakespeare



# Unstructured data in 1650

- 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 grep not the solution?



# Unstructured data in 1650

- 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 grep not the solution?
  - Slow (for large collections)
  - "NOT CALPURNIA" is non-trivial
  - Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible
  - Ranked retrieval (best documents to return) – focus of later lectures, but not this one

# Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
WORSE	1	0	1	1	1	0	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

# Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
WORSE	1	0	1	1	1	0	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

# Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
WORSE	1	0	1	1	1	0	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

# Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:

# Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
  - Take the vectors for BRUTUS, CAESAR, and CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) AND on the three vectors
  - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

## 0/1 vector for BRUTUS

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

## 0/1 vector for BRUTUS

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							



## 0/1 vector for BRUTUS

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							

10

A : [A] + D : [E] + [F]

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.

---

### Lead Release

Lord Polonius: I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

# Bigger collections

- Consider  $N = 10^6$  documents, each with about 1000 tokens

# Bigger collections

- Consider  $N = 10^6$  documents, each with about 1000 tokens
- On average 6 bytes per token, including spaces and punctuation  $\Rightarrow$  size of document collection is about 6 GB

# Bigger collections

- Consider  $N = 10^6$  documents, each with about 1000 tokens
- On average 6 bytes per token, including spaces and punctuation  $\Rightarrow$  size of document collection is about 6 GB
- Assume there are  $M = 500,000$  distinct terms in the collection

# Bigger collections

- Consider  $N = 10^6$  documents, each with about 1000 tokens
- On average 6 bytes per token, including spaces and punctuation  $\Rightarrow$  size of document collection is about 6 GB
- Assume there are  $M = 500,000$  distinct terms in the collection
- (Notice that we are making a term/token distinction.)

# Can't build the incidence matrix

- $M = 500,000 \times 10^6 =$  half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?
  - We only record the 1s.

# Inverted Index

For each term  $t$ , we store a list of all documents that contain  $t$ .

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----

⋮

**dictionary**

**postings**



# Inverted Index

For each term  $t$ , we store a list of all documents that contain  $t$ .

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----

⋮

**dictionary**

**postings**

# Inverted Index

For each term  $t$ , we store a list of all documents that contain  $t$ .

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----

⋮

**dictionary**

**postings**

# Inverted index construction

- 1 Collect the documents to be indexed:

Friends, Romans, countrymen. So let it be with Caesar ...

- 2 Tokenize the text, turning each document into a list of tokens:

Friends Romans countrymen So ...

- 3 Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend roman

countryman so ...

- 4 Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

# Tokenization and preprocessing

**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:



**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

# Generate postings

**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

# Sort postings

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

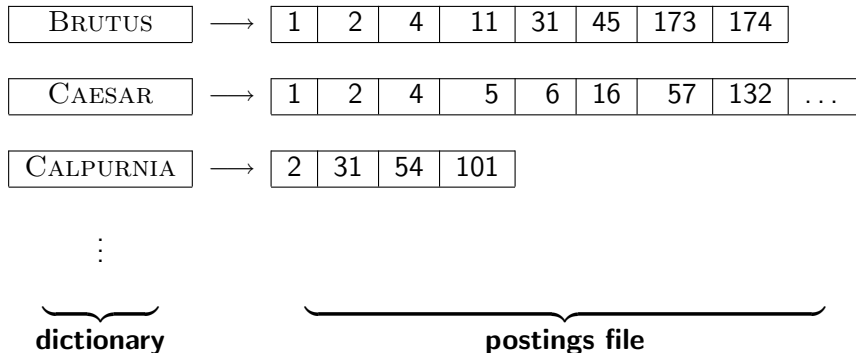
# Create postings lists, determine document frequency

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			
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
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
me	1	→	2
noble	2	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

# Split the result into dictionary and postings file





# Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?
- Index compression: how can we efficiently store and process indexes for large collections?
- Ranked retrieval: what does the inverted index look like when we want the “best” answer?

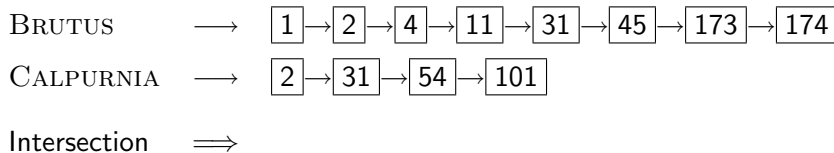
# Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries**
- 4 Course overview

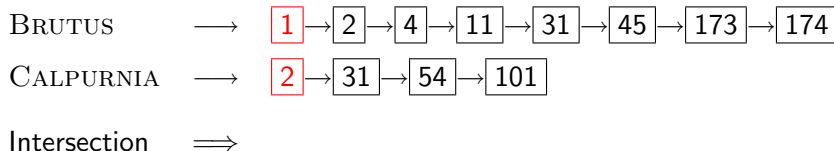
# Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  - 1 Locate BRUTUS in the dictionary
  - 2 Retrieve its postings list from the postings file
  - 3 Locate CALPURNIA in the dictionary
  - 4 Retrieve its postings list from the postings file
  - 5 Intersect the two postings lists
  - 6 Return intersection to user

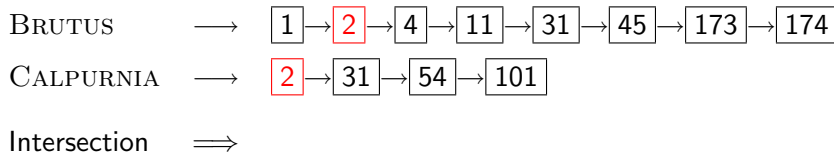
# Intersecting two postings lists



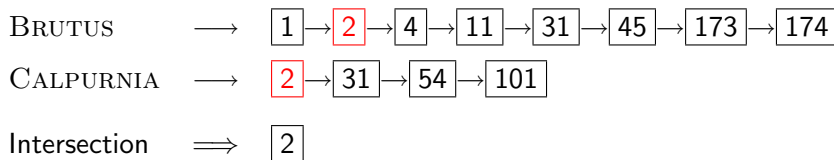
# Intersecting two postings lists



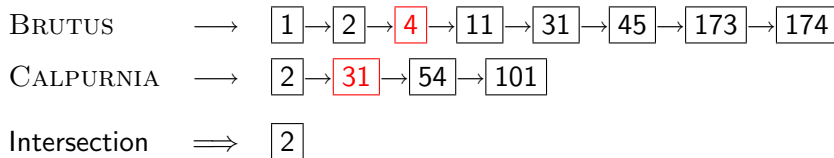
# Intersecting two postings lists



# Intersecting two postings lists

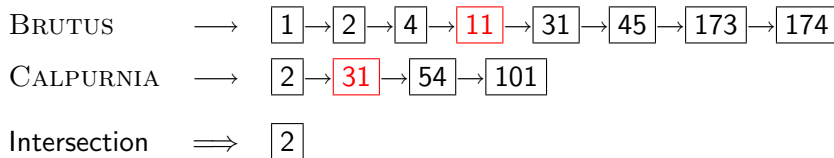


# Intersecting two postings lists

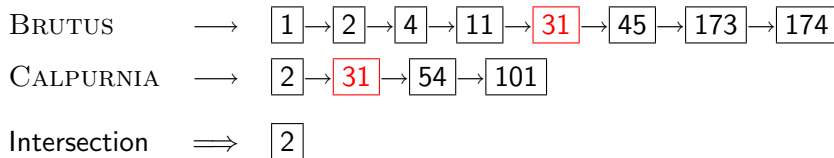




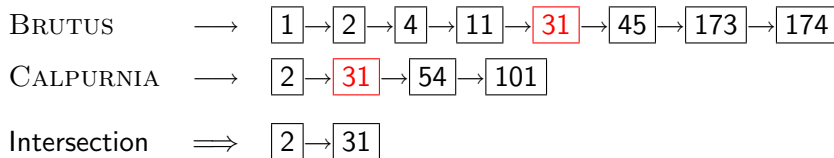
# Intersecting two postings lists



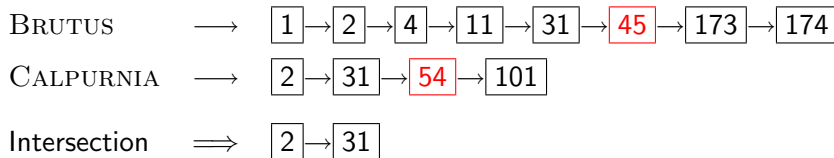
# Intersecting two postings lists



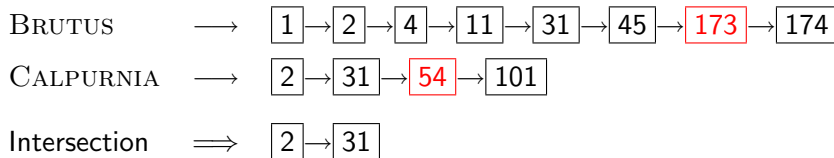
# Intersecting two postings lists



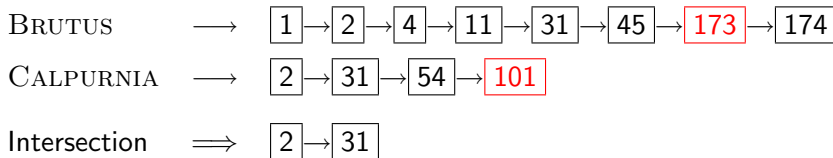
# Intersecting two postings lists



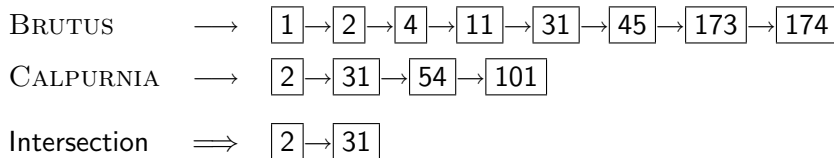
# Intersecting two postings lists



# Intersecting two postings lists



# Intersecting two postings lists



# Intersecting two postings lists

BRUTUS  $\longrightarrow$   $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA  $\longrightarrow$   $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection  $\implies$   $\boxed{2} \rightarrow \boxed{31}$

- This is linear in the length of the postings lists.
- This only works if postings lists are sorted.



# Intersecting two postings lists

```
INTERSECT( $p_1, p_2$ )  
  1   $answer \leftarrow \langle \rangle$   
  2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$   
  3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$   
  4      then  $\text{ADD}(answer, \text{docID}(p_1))$   
  5           $p_1 \leftarrow \text{next}(p_1)$   
  6           $p_2 \leftarrow \text{next}(p_2)$   
  7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$   
  8          then  $p_1 \leftarrow \text{next}(p_1)$   
  9          else  $p_2 \leftarrow \text{next}(p_2)$   
10  return  $answer$ 
```

# Boolean queries

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a [set](#) of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries
  - You know exactly what you are getting.
- Many search systems you use are also Boolean: email, intranet etc.

# Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users ...
- ...although ranked retrieval has been available since 1992.

# Westlaw: Example queries

*Information need:* Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company

*Query:* "trade secret" /s disclos! /s prevent /s employe!

*Information need:* Requirements for disabled people to be able to access a workplace

*Query:* disab! /p access! /s work-site work-place (employment /3 place)

*Information need:* Cases about a host's responsibility for drunk guests

*Query:* host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

# Westlaw: Comments

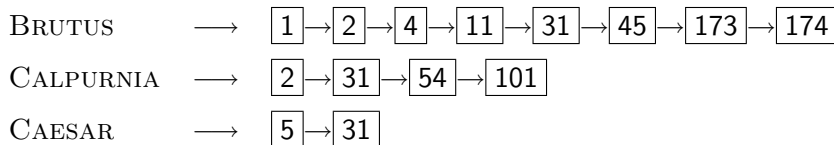
- $/3$  = within 3 words
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: proximity operators, incrementally developed, not like web search
- Why professional searchers often like Boolean search: precision, transparency, control
- When are Boolean queries the best way of searching? Depends on: information need, searcher, document collection, ...

# Query optimization

- What is the best order for query processing?
- Consider a query that is an AND of  $n$  terms,  $n > 2$
- For each of the terms, get its postings list, then AND them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR

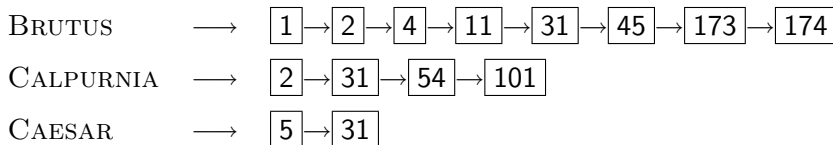
# Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR



# Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: **Process in order of increasing frequency**
- Start with the shortest postings list, then keep cutting further
- In this example, first CAESAR, then CALPURNIA, then BRUTUS





# Optimized intersection algorithm for conjunctive queries

```
INTERSECT( $\langle t_1, \dots, t_n \rangle$ )  
1  terms  $\leftarrow$  SORTBYINCREASINGFREQUENCY( $\langle t_1, \dots, t_n \rangle$ )  
2  result  $\leftarrow$  postings(first(terms))  
3  terms  $\leftarrow$  rest(terms)  
4  while terms  $\neq$  NIL and result  $\neq$  NIL  
5  do result  $\leftarrow$  INTERSECT(result, postings(first(terms)))  
6     terms  $\leftarrow$  rest(terms)  
7  return result
```

# More general optimization

- Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each OR by the sum of its frequencies (conservative)
- Process in increasing order of OR sizes

# Exercise

Recommend a query processing order for: (TANGERINE OR TREES) AND (MARMALADE OR SKIES) AND (KALEIDOSCOPE OR EYES)

# Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries
- 4 Course overview

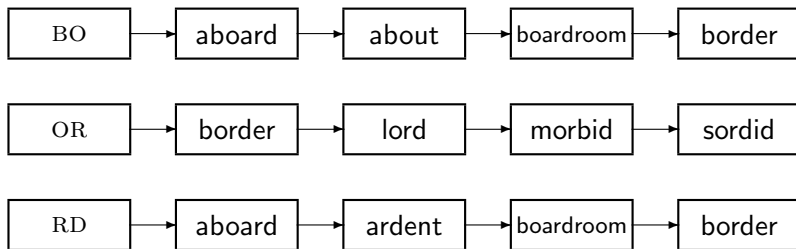
# Course overview

- We are done with Chapter 1 of IIR (IIR 01).
- Plan for the rest of the semester: cover (parts of) 15 chapters of IIR.
- In what follows: one teaser per chapter to give you a sense of what will be covered in this course.

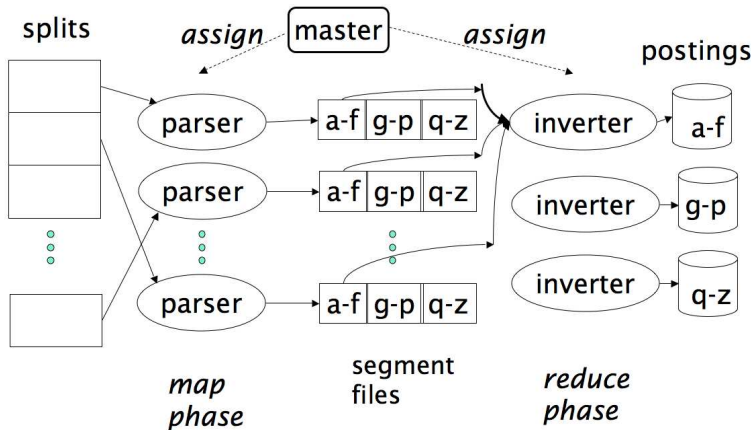
## IIR 02: The term vocabulary and postings lists

- Phrase queries: STANFORD UNIVERSITY
- Proximity: find GATES NEAR MICROSOFT
- We need an index that captures **position information** for phrase queries and proximity queries.

## IIR 03: Dictionaries and tolerant retrieval

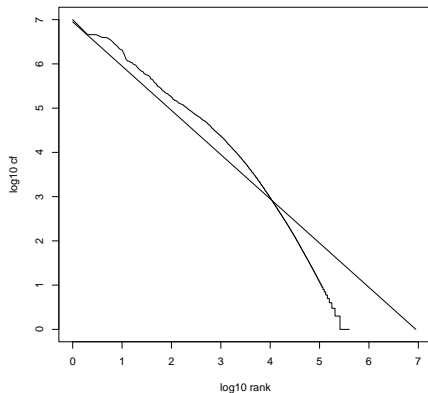


## IIR 04: Index construction





# IIR 05: Index compression

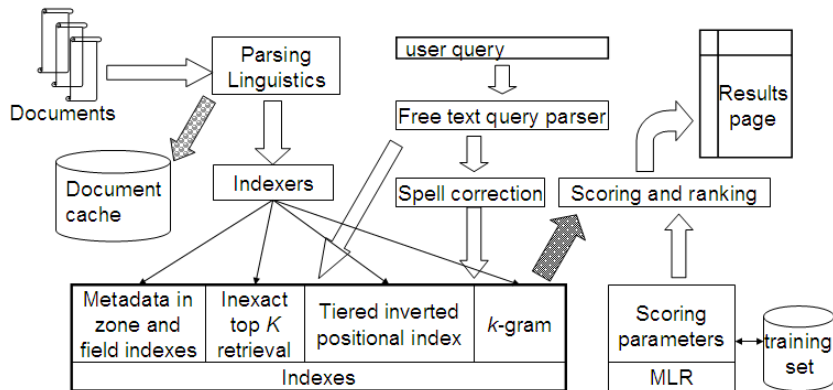


Zipf's law

# IIR 06: Scoring, term weighting and the vector space model

- Ranking search results
  - Boolean queries only give inclusion or exclusion of documents.
  - For ranked retrieval, we measure the proximity from query to each document.
  - One formalism for doing this: the vector space model
- Key challenge in ranked retrieval: evidence accumulation for a term in a document
  - 1 vs. 0 occurrence of a query term in the document
  - 2 vs. 1 occurrences of a query term in the document
  - 3 vs. 2 occurrences of a query term in the document
  - Usually: more is better
  - But by how much?
  - Need a scoring function that translates frequency into score or weight













## IIR 07: Scoring in a complete search system



## IIR 08: Evaluation and dynamic summaries

... **In recent years, Papua New Guinea has faced severe economic difficulties and** economic growth has slowed, partly as a result of weak governance and civil war, and partly as a result of external factors such as the Bougainville civil war which led to the closure in 1989 of the Panguna mine (at that time the most important foreign exchange earner and contributor to Government finances), the Asian financial crisis, a decline in the prices of gold and copper, and a fall in the production of oil. **PNG's economic development record over the past few years is evidence that** governance issues underly many of the country's problems. Good governance, which may be defined as the transparent and accountable management of human, natural, economic and financial resources for the purposes of equitable and sustainable development, flows from proper public sector management, efficient fiscal and accounting mechanisms, and a willingness to make service delivery a priority in practice. ...

## IIR 09: Relevance feedback &amp; query expansion

Browse Search Prev Next Random					
					
(144538, 523493) 0.54182 0.231944 0.309876	(144538, 523835) 0.56319296 0.267304 0.295869	(144538, 523529) 0.584279 0.280881 0.303398	(144456, 253509) 0.64501 0.351395 0.293615	(144456, 253568) 0.650275 0.411745 0.23853	(144538, 523799) 0.66709197 0.358033 0.309059
					
(144473, 16249) 0.6721 0.393922 0.278178	(144456, 249634) 0.675018 0.4639 0.211118	(144456, 253693) 0.676901 0.47645 0.200451	(144473, 16328) 0.700339 0.309002 0.391337	(144483, 265264) 0.70170796 0.36176 0.339948	(144478, 512410) 0.70297 0.469111 0.233859

# IIR 10: XML retrieval

- IR (unstructured data) vs. databases (structured data)

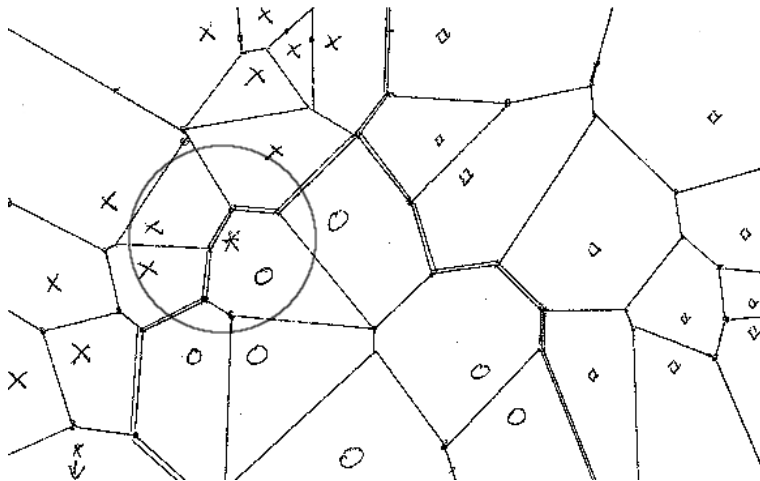
- A typical table in a database:
 

employee	manager	salary
Smith	Jones	50,000
Chang	Smith	60,000
Ivy	Smith	50,000
- Databases support search for numerical range and exact match, e.g., salary < 60,000 AND manager=Smith.
- If your data is structured and you only need precise queries like this (numerical, exact match etc), don't use an IR system.
- XML is between structured databases and unstructured IR.
- We may not get to XML retrieval in this course.

# IIR 13: Text classification & Naive Bayes

- Text classification = assigning documents automatically to predefined classes
- Examples:
  - Language (English vs. French)
  - Adult content
  - Region

# IIR 14: Vector classification





# IIR 16: Flat clustering



Vivísimo®

jaguar the Web

**Search** [Advanced Search](#) [Help](#)

**Clustered Results** Top 208 results of at least 20,373,974 retrieved for the query **Jaguar** (Details)

- ▶ [Jaguar](#) (208)
  - ⊕ ▶ [Cars](#) (74)
  - ⊕ ▶ [Club](#) (34)
  - ⊕ ▶ [Cat](#) (23)
  - ⊕ ▶ [Animal](#) (13)
  - ⊕ ▶ [Restoration](#) (10)
  - ⊕ ▶ [Mac OS X](#) (8)
  - ⊕ ▶ [Jaguar Model](#) (8)
  - ⊕ ▶ [Request](#) (5)
  - ⊕ ▶ [Mark Webber](#) (6)
  - ▶ [Maya](#) (5)
  - ▼ [More](#)

Find in clusters:  
Enter Keywords 

1. [Jag-lovers - THE source for all Jaguar information](#) [new window] [frame] [cache] [preview] [clusters]  
 ... Internet! Serving Enthusiasts since 1993 The Jag-lovers Web Currently with 40661 members The Premier **Jaguar** Cars web resource for all enthusiasts Lists and Forums Jag-lovers originally evolved around its ...  
[www.jag-lovers.org](http://www.jag-lovers.org) - Open Directory 2, Wisenut 8, Ask Jeeves 8, MSN 9, Looksmart 12, MSN Search 18
2. [Jaguar Cars](#) [new window] [frame] [cache] [preview] [clusters]  
 [...] redirected to [www.jaguar.com](http://www.jaguar.com)  
[www.jaguar.com](http://www.jaguar.com) - Looksmart 1, MSN 2, Lycos 3, Wisenut 6, MSN Search 9, MSN 29
3. <http://www.jaguar.com/> [new window] [frame] [preview] [clusters]  
[www.jaguar.com](http://www.jaguar.com) - MSN 1, Ask Jeeves 1, MSN Search 3, Lycos 9
4. [Apple - Mac OS X](#) [new window] [frame] [preview] [clusters]  
 Learn about the new OS X Server, designed for the Internet, digital media and workgroup management. Download a technical factsheet.  
[www.apple.com/macosx](http://www.apple.com/macosx) - Wisenut 1, MSN 3, Looksmart 26

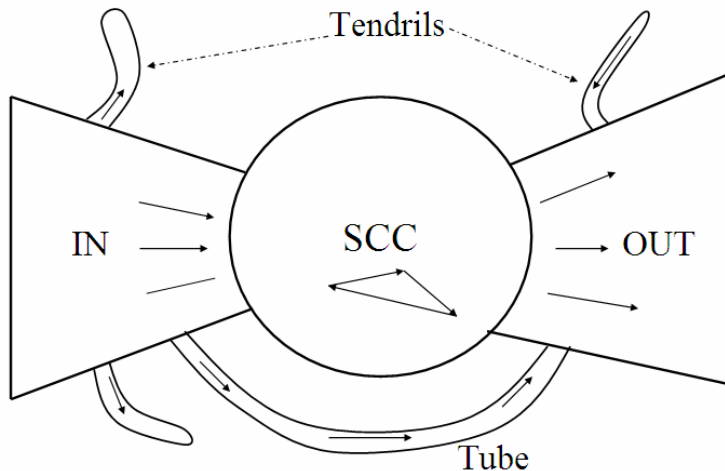
# IIR 17: Hierarchical clustering

<http://news.google.com>

# IIR 19: The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users and information needs
- Beyond terms and text: exploit link analysis, user data
- How do web search engines work?
- How can we make them better?

## IIR 21: Link analysis / PageRank



# IIR 20: Crawling

