

Introduction to Information Retrieval

Christopher D. Manning, Prabhakar Raghavan
and Hinrich Schütze

Lecture 1: Boolean Retrieval

(slides annotated by Mizuho Iwaihara)

1

1

Outline

- Introduction
- Inverted index
- Processing Boolean queries
- Query optimization

2

2

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).

3

3

Structured and Unstructured data

Structured data:

- Typically formatted as database tables, with rows and columns.
- Stored in databases and data warehouses
- Queried by database query languages (SQL)
- Business data, customer data, scientific data

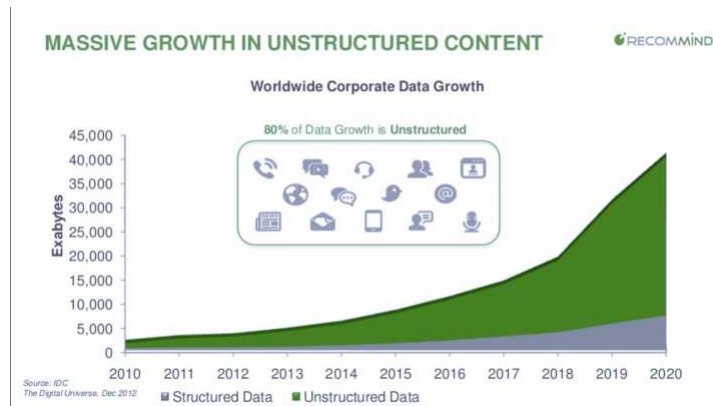
Unstructured data:

- Not conforming to a predefined schema.
- Example: e-mails, social media posts, web contents, video, audio, IoT device data

4 4

4

Growth of unstructured data



5

5

5

Semi-structured data

Semi-structured data:

- Markup languages - Hyper texts
 - tag names indicate semantics of data items.
 - XML, JSON formats
- Hierarchical structure of tagged texts
 - Schema can (partially) constrain hierarchal structure
 - More flexible than structured data
- Suitable for information exchange.

6

6

6

Big Data

Big Data's 3V:

- **Variety**
 - Unstructured data: mail, social media, audio, video, sensor data
 - Semi-structured data: XML, JSON
 - Structured data: databases, data warehouse
- **Volume**
 - Tera byte, peta byte, zeta byte
- **Velocity**
 - (near) real-time processing
 - Online transactions, market price, e-commerce, live streaming, online gaming,...

7 7

7

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- 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?
Whitespace (AND), OR, "-"(NOT)

8

8

Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries
- 4 Query optimization

9

9

Unstructured data in 1650: Shakespeare



10

10

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)
 - grep is line-oriented, IR is document-oriented
 - "NOT CALPURNIA" is non-trivial
 - Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible

11

11

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*.

12

12

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 NOT CALPURNIA
 - Complement the vector of CALPURNIA
 - Do a (bitwise) and on the three vectors
 - 110100 AND 110111 AND 101111 = 100100

13

13

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
...						
result:	1	0	0	1	0	0

14

14

Answers to query

Anthony 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.

15

15

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- \Rightarrow total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation \Rightarrow size of document collection is about $6 \cdot 10^9 = 6$ GB
- Assume there are $M = 500,000$ distinct terms in the collection
- (Notice that we are making a term/token distinction.)

16

16

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.

17

17

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

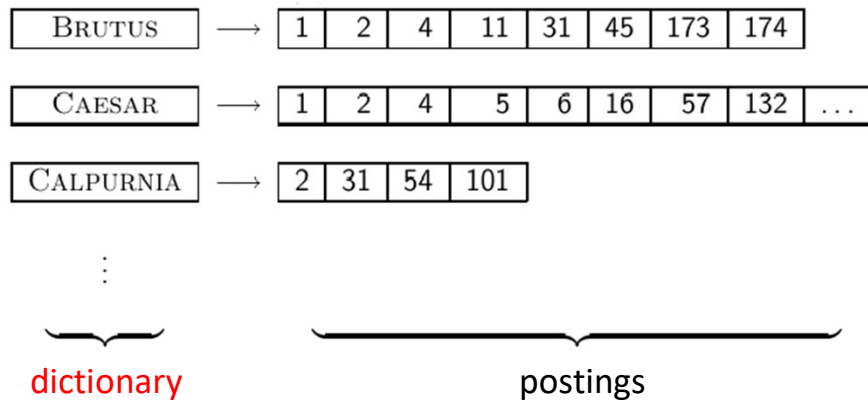
⏟
postings

18

18

Inverted Index

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

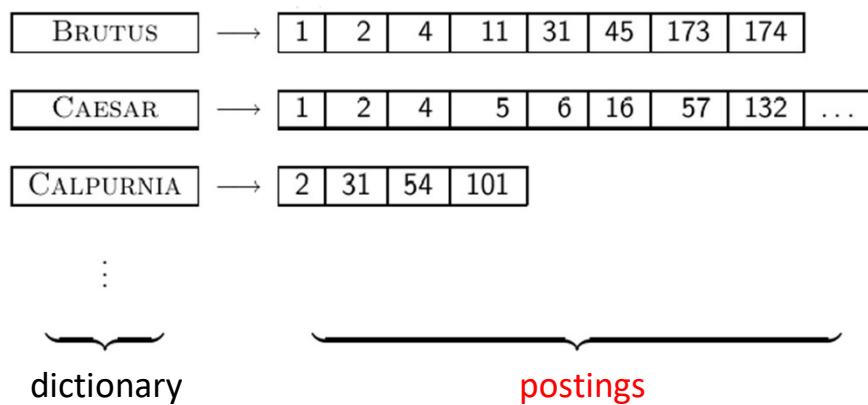


19

19

Inverted Index

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



20

20

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.

21

21

Tokenizing 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

22

22

Generate posting

	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

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



23

23

Sort postings

term	docID	term	docID
i	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
i	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	i	1
killed	1	i	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



24

24

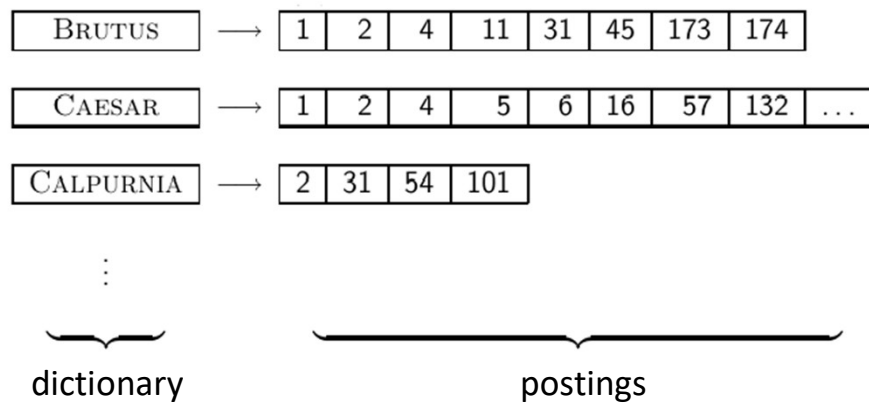
Create postings lists, determine document frequency

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

25

25

Split the result into dictionary and postings file



26

26

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?

27

27

Outline

- ① Introduction
- ② Inverted index
- ③ Processing Boolean queries
- ④ Query optimization

28

28

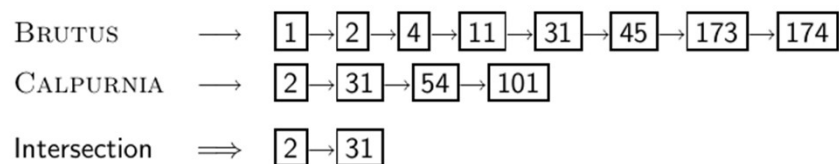
Simple conjunctive query (two terms)

- 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

29

29

Intersecting two posting lists



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

30

30

Intersecting two posting 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  $docID(p_1) = docID(p_2)$ 
4      then  $\text{ADD}(answer, docID(p_1))$ 
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7  else if  $docID(p_1) < docID(p_2)$ 
8      then  $p_1 \leftarrow next(p_1)$ 
9  else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 

```

31

31

Query processing: Exercise

FRANCE \rightarrow [1] \rightarrow [2] \rightarrow [3] \rightarrow [4] \rightarrow [5] \rightarrow [7] \rightarrow [8] \rightarrow [9] \rightarrow [11] \rightarrow [12] \rightarrow [13] \rightarrow [14] \rightarrow [15]

PARIS \rightarrow [2] \rightarrow [6] \rightarrow [10] \rightarrow [12] \rightarrow [14]

LEAR \rightarrow [12] \rightarrow [15]

Compute hit list for ((paris AND NOT france) OR lear)

32

32

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: spotlight, email, intranet etc.

33

33

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.

34

34

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

35

35

Westlaw: Comments

- Proximity operators: /3 = within 3 words, /s = within a sentence, /p = within a paragraph
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: 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, . . .

36

36

Outline

- ① Introduction
- ② Inverted index
- ③ Processing Boolean queries
- ④ Query optimization

37

37

Query optimization

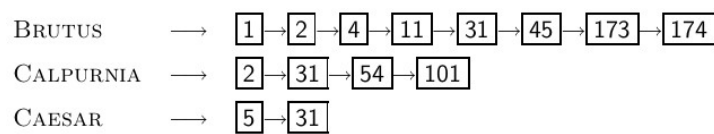
- Consider a query that is an and of n terms, $n > 2$
- For each of the terms, get its postings list, then and them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- What is the best order for processing this query?

38

38

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



39

39

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

```

40

40

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

41

41

42

42