مبانی بازیابی اطلاعات و جستجوی وب

۲- بازیابی Boolean

Original slides by: Hinrich Schütze and Christina Lioma

Outline

1. Inverted index
2. Processing Boolean queries
3. Query optimization

Boolean retrieval

- ساده ترین مدل برای یک سیستم بازیابی اطلاعات
- پرس و جوها عبارات Boolean هستند، مثلا

CAESAR AND BRUTUS

- موتور جستجو تمامی اسنادی که عبارت بولی را برآورده میسازد، برمی گرداند.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.

Does Google use the Boolean model?

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, petabyte)
  - Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible
  - Ranked Retrieval

متراکم نمودن داده (Indexing)

- avoid linearly scanning the texts for each query:
  - index the INDEX documents in advance.
- Using the index instead of linearly scanning the docs that is computationally expensive for large collections
  - Indexing depends on the query language and IR model
- **Term** (index unit): A word, phrase, and other groups of symbols used for retrieval

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
WORSER	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 NOT CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) and on the three vectors
  - 110100 AND 110111 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
. . .						
result:	1	0	0	1	0	0

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens/words ⇒ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation ⇒ 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.)

Can’t build the incidence matrix

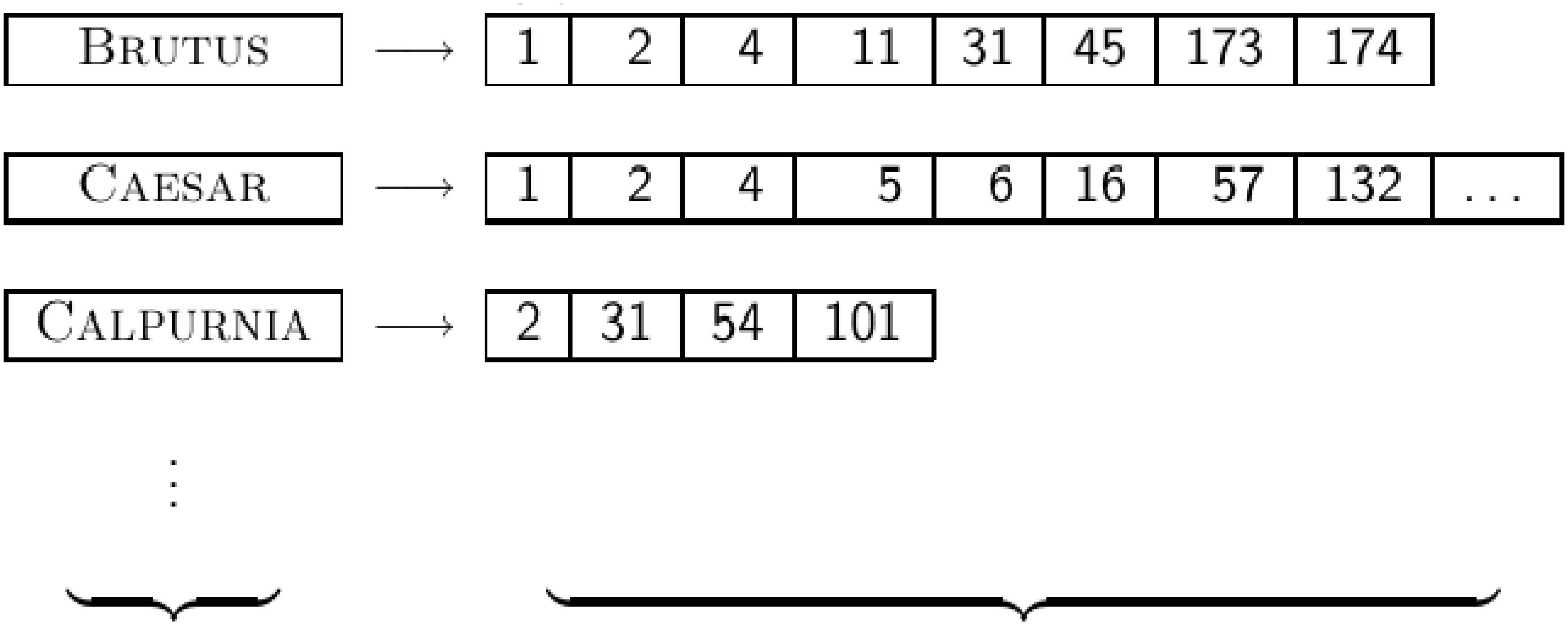
- M = 500,000 × 10^6 = half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?

Can’t build the incidence matrix

- M = 500,000 × 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 construction

1. جمع آوری اسناد

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

2. Tokenize the text: هر سند به لیستی از توکن ها تبدیل می شود

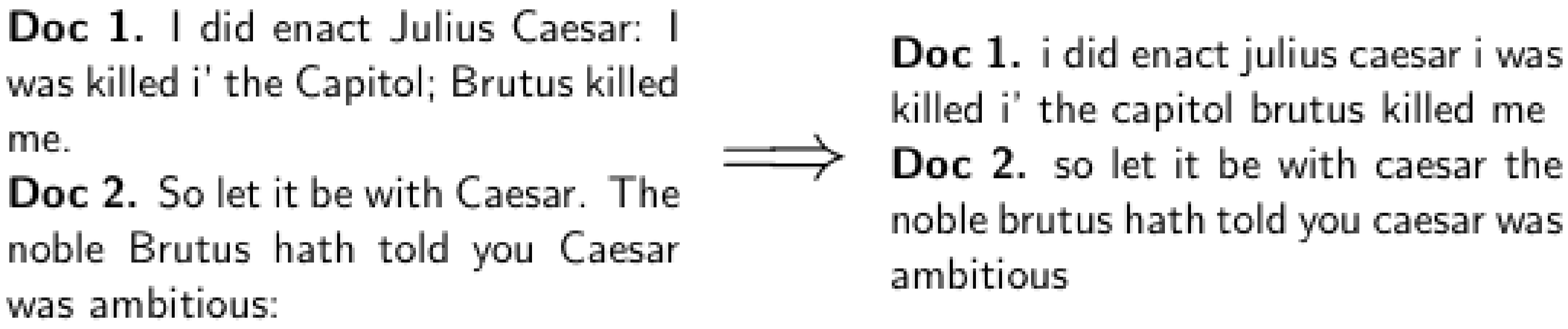
Friends | Romans | countrymen | So . . .

3. انجام پردازش زبانی برای تولید مجموعه توکن ها نرمال شده که همان indexing
term هستند.

friend | roman | countryman | so . . .

4. شاخص گذاری اسنادی که هر ترم در آنها رخ می دهد یا ایجاد یک شاخص معکوس
شامل دیکشنری و postings

Tokenizing and preprocessing

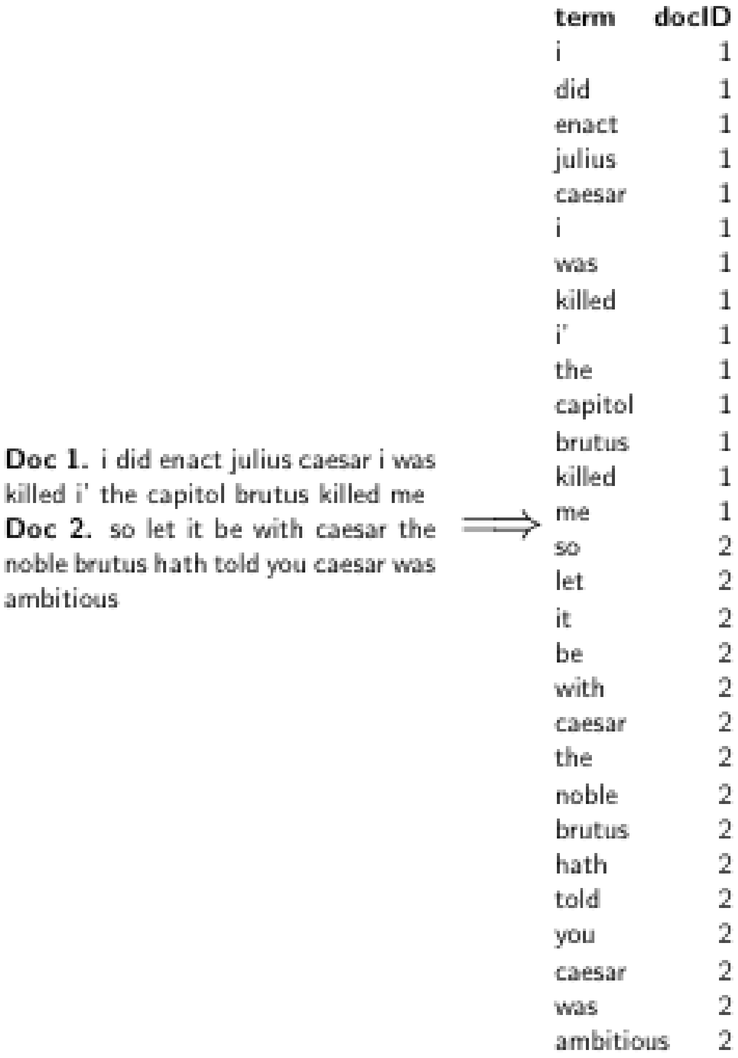


Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

⟹

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

Generate posting

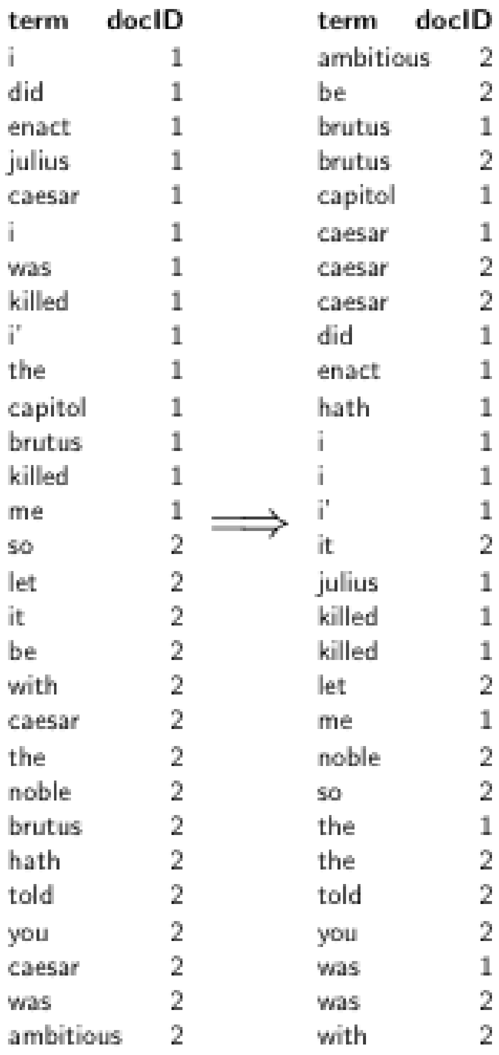


Doc 1. i did enact julius caesar i was killed i' the capitol brutus killed me
Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious

⟹

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

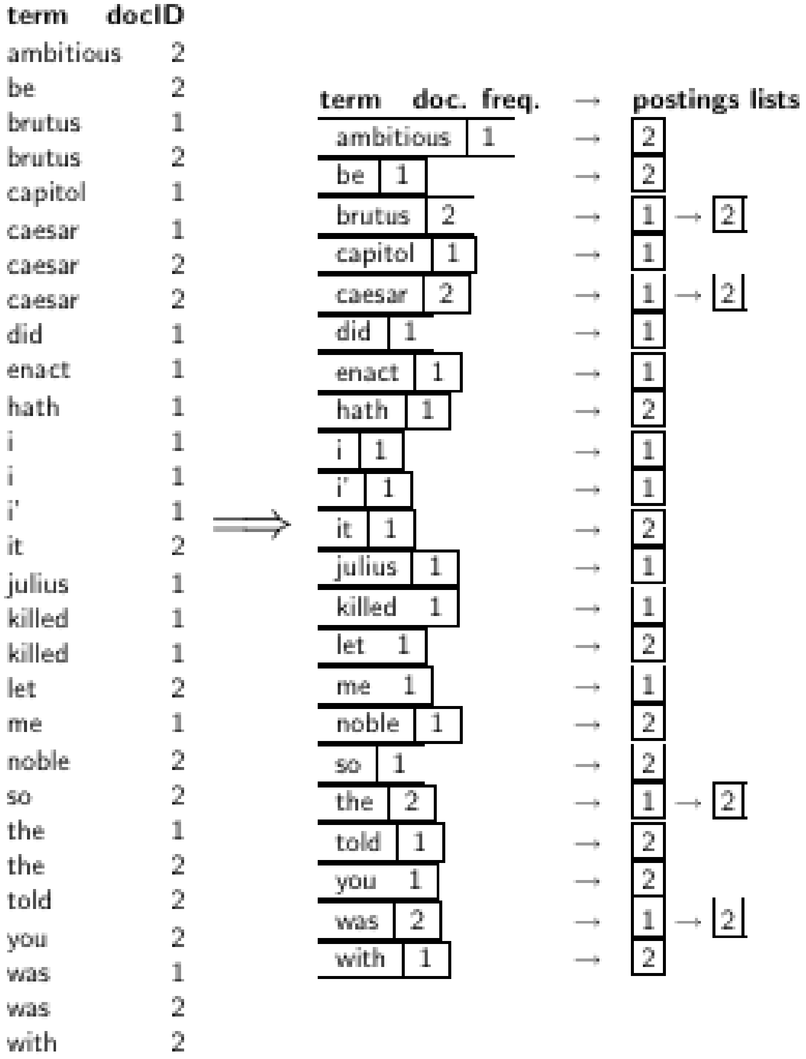


term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⟹

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

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
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

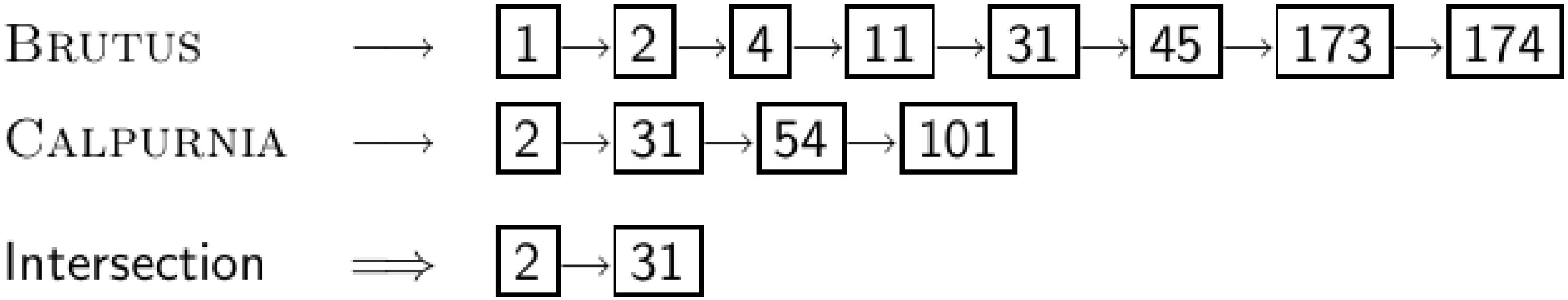
Outline

1. Inverted index
2. Processing Boolean queries
3. Query optimization

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 posting lists



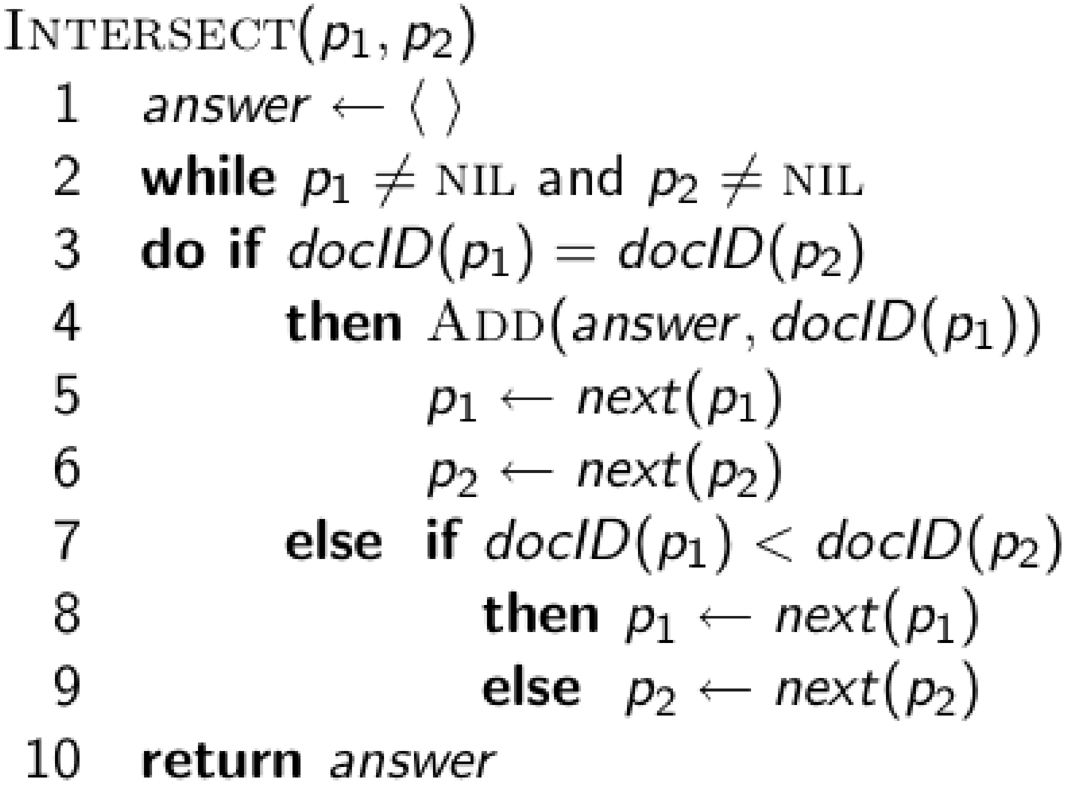
Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

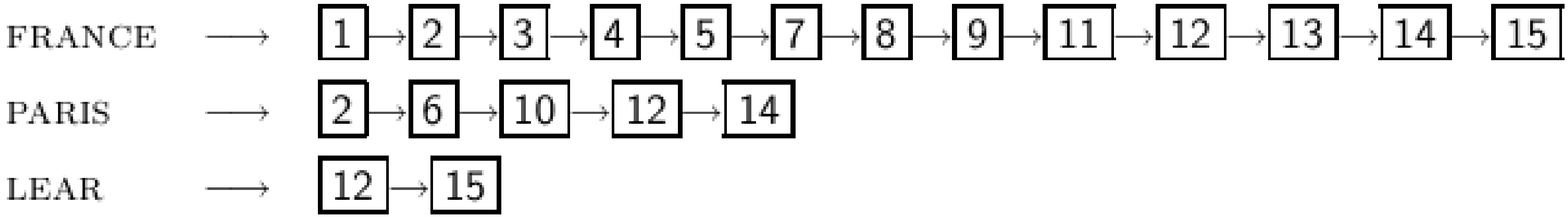
- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Intersecting two posting lists



```
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 $answer$
```

Query processing: Exercise



FRANCE → 1 → 2 → 3 → 4 → 5 → 7 → 8 → 9 → 11 → 12 → 13 → 14 → 15

PARIS → 2 → 6 → 10 → 12 → 14

LEAR → 12 → 15

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

Outline

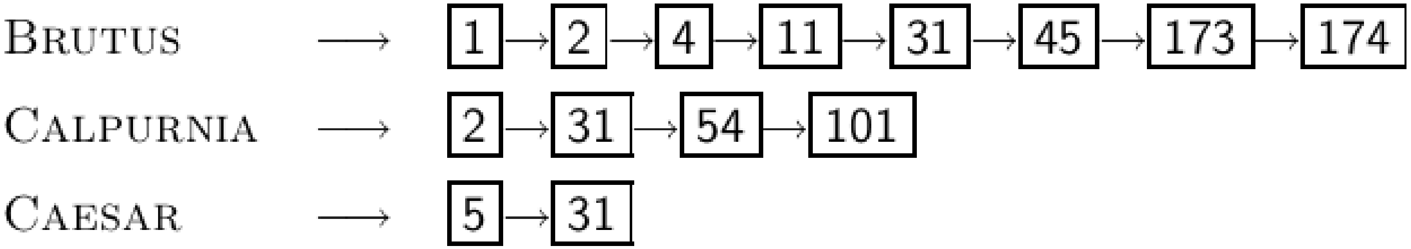
1. Inverted index
2. Processing Boolean queries
3. Query optimization

Query optimization

- پرس و جویی را در نظر بگیرید که شامل and تعداد n ترم است.
- برای هر ترم، لیست posting آن را گرفته و آنها را باهم and می کنیم.
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- بهترین ترتیب اجرای این پرس و جو چیست؟

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

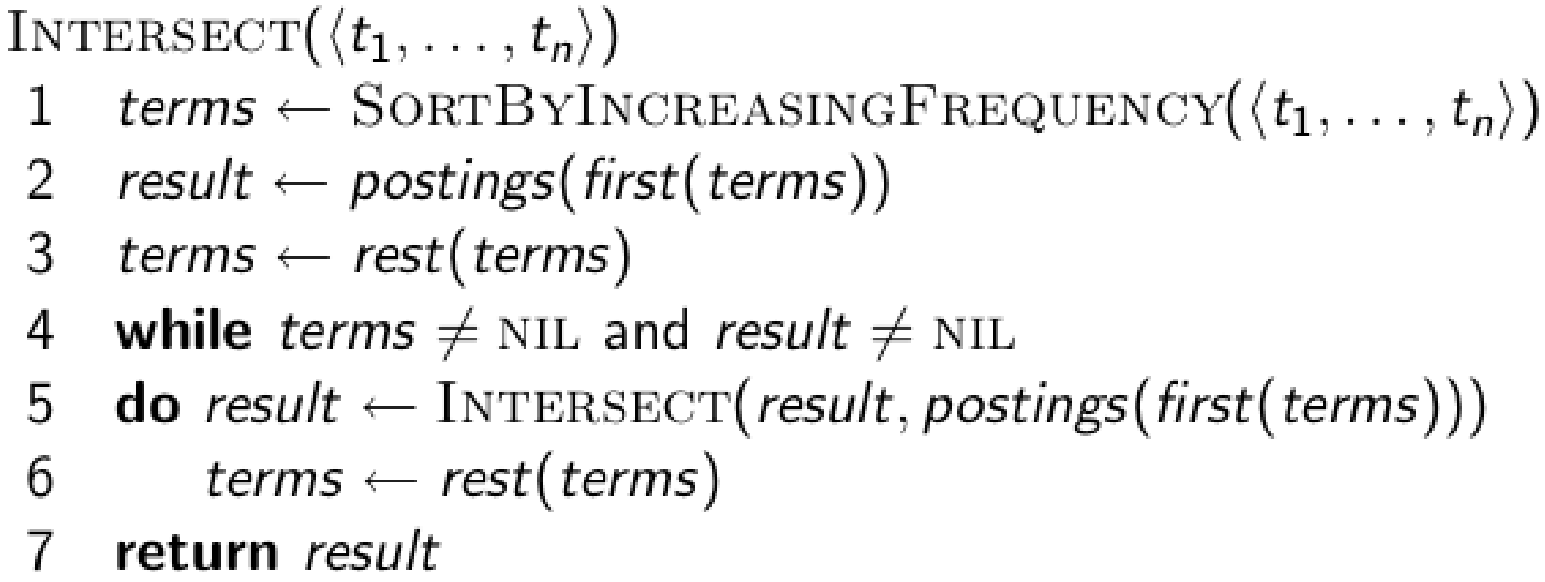


BRUTUS → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA → 2 → 31 → 54 → 101

CAESAR → 5 → 31

Optimized intersection algorithm for conjunctive queries



```
INTERSECT($\langle t_1, \ldots, t_n \rangle$)
  terms ← SORTBYINCREASINGFREQUENCY($\langle t_1, \ldots, t_n \rangle$)
  result ← postings(first(terms))
  terms ← rest(terms)
  while terms ≠ NIL and result ≠ NIL
  do result ← INTERSECT(result, postings(first(terms)))
     terms ← rest(terms)
  return result
```

Optional: Westlaw: Example queries

Largest commercial legal search service in terms of the number of paying subscribers, uses Boolean search as default

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)

Optional: Other concepts

- 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?

منابع

- فصل اول کتاب An introduction to information retrieval