Web Information Retrieval

Textbook by
Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze

Notes Revised by X. Meng for SEU
May 2014

Outlines

- We will discuss two more topics.
  - Boolean retrieval
  - Posting list

Posting Lists

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

4

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

5

Sort Postings

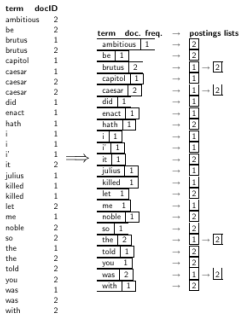
term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⟹

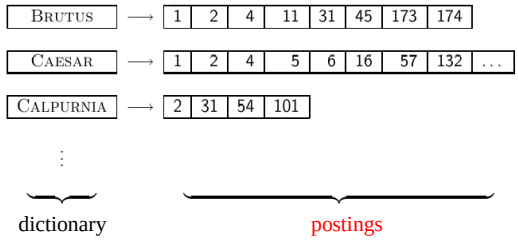
term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

6

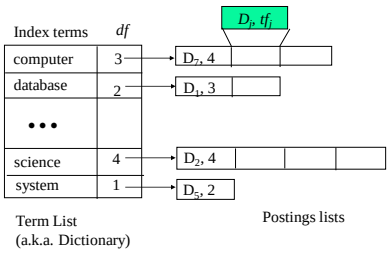
Creating Postings Lists, Determine Document Frequency



Split the Result into Dictionary and Postings File



Inverted Index



Creating an Inverted Index

Create an empty index term list I;
For each document, D, in the document set V
  For each (non-zero) token, T, in D:
    If T is not already in I
      Insert T into I;
    Find the location for T in I;
    If (T, D) is in the posting list for T
      increase its term frequency for T;
    Else
      Create (T, D);
      Add it to the posting list for T;

In-Class Work

- Creating an inverted index for the following documents
  - "The quick brown fox jumps over the lazy dog"
  - "The quick fox run"
  - "The lazy dog sleep"
- Assume:
  - "the" and "over" are stopwords that are not indexed
  - various forms of verbs are not indexed separately, only the original form (stem) is indexed

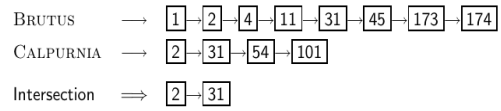
Boolean Retrieval

Processing Boolean queries
Query optimization

Simple Conjunctive Query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  - Locate BRUTUS in the dictionary (term list)
  - Retrieve its postings list from the postings file
  - Locate CALPURNIA in the dictionary
  - Retrieve its postings list from the postings file
  - Intersect the two postings lists
  - Return intersection to user

Intersecting Two Posting Lists



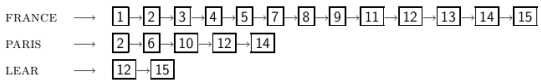
- The complexity is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Algorithm of Intersecting Two Posting Lists

Assume posting lists are sorted by docID

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```


Query Processing: In-Class Work



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

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.
  - Precise: either document matches condition or not, nothing in between.
- Primary commercial retrieval tool for three 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.

Outline

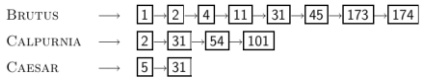
- ~~Processing Boolean queries~~
- Query optimization

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? That is, should we process BRUTUS first? CALPURNIA first? Or CEASAR first?

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(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← 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