Monsoon 2019

Query Processing and Positional Index

Information Retrieval

by

Dr. Rajendra Prasath

Indian Institute of Information Technology

Sri City – 517 646, Andhra Pradesh, India

Lecture – 05

Query Processing and Positional Indexes

Recap: Creating Inverted Index

d1) Turing machines can define computational processes that do not terminate. The informal definitions of algorithms generally require that the algorithm always terminates. This requirement renders the task of deciding whether a formal procedure is an algorithm impossible in the general case

d2) Typically, when an algorithm is associated with processing information, data can be read from an input source, written to an output device and stored for further processing. Stored data are regarded as part of the internal state of the entity performing the algorithm.

d3) For some such computational process, the algorithm must be rigorously defined: specified in the way it applies in all possible circumstances that could arise. Any conditional steps must be systematically dealt with, case-by-case

Recap: Boolean Retrieval

- For each term, we have a vector consisting of 0 / 1
- To answer query: take the vectors for **Brutus**, **Caesar** and **Calpurnia** (complemented) → bitwise AND
  - 110100 AND
  - 110111 AND
  - 101111 =
  - **100100**

Query:
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

Recap: Query processing: AND

- 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):

Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

Query optimization

- What is the best order for query processing?
- Consider a query that is an AND of n terms.
- For each of the n terms, get its postings, then AND them together.

Brutus	2	4	8	16	32	64	128	
Caesar	1	2	3	5	8	16	21	34
Calpurnia	13	16						

Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Query optimization example

- Process in order of increasing frequencies:
  - *start with smallest set, then keep cutting further*

This is why we kept document freq. in dictionary

Brutus →	2	4	8	16	32	64	128	
Caesar →	1	2	3	5	8	16	21	34
Calpurnia →	13	16						

Execute the query as (***Calpurnia*** *AND* ***Brutus***) *AND* ***Caesar***

Query Processing - Exercises

- Exercise: If the query is ***friends*** *AND* ***romans*** *AND (NOT* ***countrymen****)*, how could we use the freq of ***countrymen***?

- Exercise: Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?

- Hint: Begin with the case of a Boolean *formula* query: in this, each query term appears only once in the query

Phrase queries

- We want to be able to answer queries such as "**stanford university**" – as a phrase
- Thus the sentence "**I went to university at Stanford**" is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only

  <*term* : *docs*> entries

A first attempt: Biword indexes

- Index **every consecutive pair of terms** in the text as a phrase
- For example the text "Friends, Romans, Countrymen" would generate the biwords
  - **friends romans**
  - **romans countrymen**
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate

Longer phrase queries

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university AND university palo AND palo alto

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase

Can have false positives!

Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them

- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Solution 2: Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

<***term**,* number of docs containing ***term***;

doc1: position1, position2 … ;

doc2: position1, position2 … ;

etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to***:
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be***:
    - *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
- Same general method for proximity searches

Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here, /*k* means "**within *k* words of**"
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- **Exercise:** Adapt the linear merge of postings to handle proximity queries
- Can you make it work for any value of *k*?
  - This is a little tricky to do correctly and efficiently

Positional index size

- A positional index expands postings storage *substantially*
  - Even though indices can be compressed

- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries
  - used explicitly or implicitly in a ranking retrieval system

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has < 1000 terms

    Why?
  - SEC filings, books, even some epic poems … easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- A positional index is 2–4 as large as a non-positional index

- Positional index size 35–50% of volume of original text

  – Caveat: all of this holds for "English-like" languages

Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (“**Michael Jackson**”, “**Britney Spears**”) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like “**The Who**”
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

Summary

In this class, we focused on:

(a) Words / Terms / Lexical Units

(b) Preparing Term – Document matrix

(c) Boolean Retrieval

(d) Inverted Index Construction

i. Computational Cost

ii. Managing Bigger Collections

iii. How much storage is required?

iv. Boolean Queries: Exact match

Assistance

- ✧ You may post your questions to me at any time

- ✧ You may meet me in person on available time or with an appointment

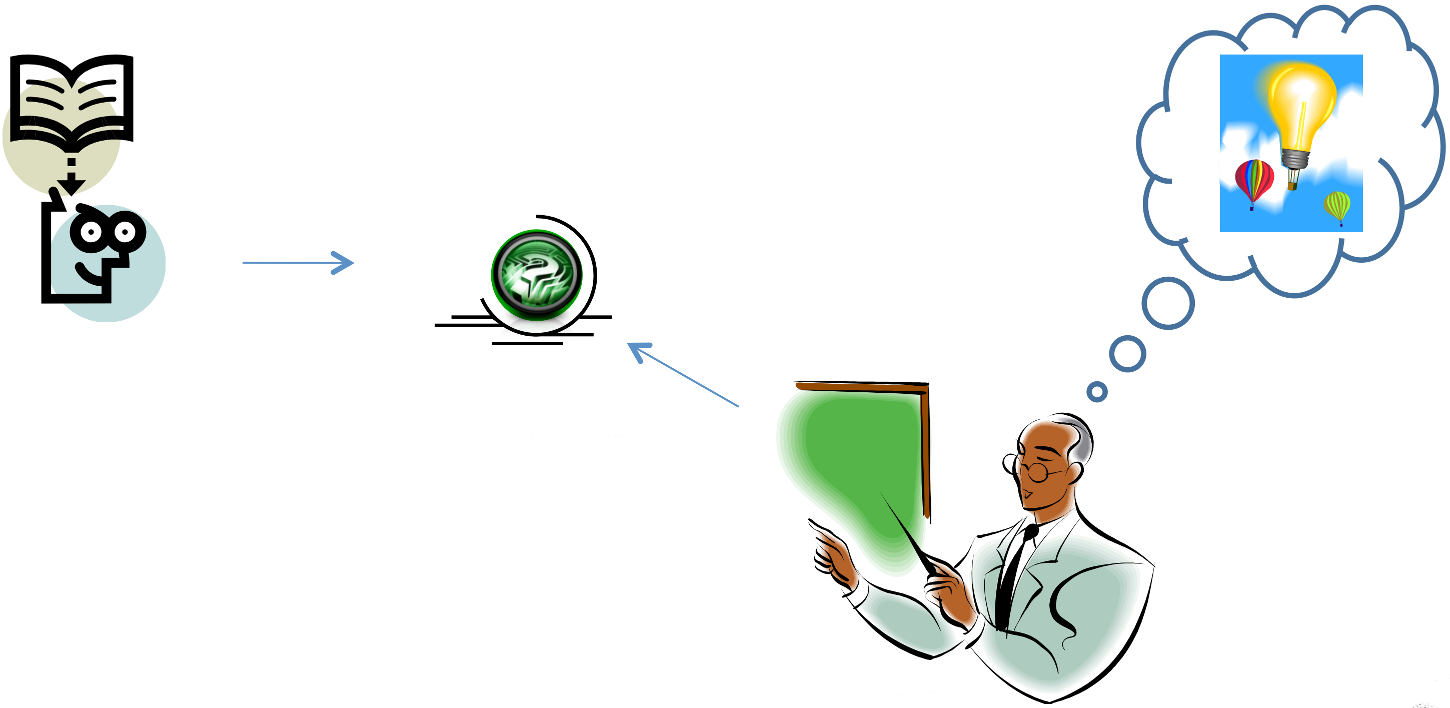
- ✧ You may leave me an email any time (the best way to reach me faster)

Acknowledgements

Thanks to ALL RESEARCHERS:

1. Introduction to Information Retrieval Manning, Raghavan and Schutze, Cambridge University Press, 2008.
2. Search Engines Information Retrieval in Practice W. Bruce Croft, D. Metzler, T. Strohman, Pearson, 2009.
3. Information Retrieval Implementing and Evaluating Search Engines Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, MIT Press, 2010.
4. Modern Information Retrieval Baeza-Yates and Ribeiro-Neto, Addison Wesley, 1999.
5. Many Authors who contributed to SIGIR / WWW / KDD / ECIR / CIKM / WSDM and other top tier conferences
6. Prof. Mandar Mitra, Indian Statistical Institute, Kolkatata (https://www.isical.ac.in/~mandar/)

Thanks …



… Questions ???