PV211: Introduction to Information Retrieval
`https://www.fi.muni.cz/~sojka/PV211`

IIR 1: Boolean Retrieval
Handout version

Petr Sojka, Hinrich Schütze et al.

Faculty of Informatics, Masaryk University, Brno
Center for Information and Language Processing, University of Munich

2020-02-19

Take-away

- Basic information about the course, teachers, evaluation, exercises
- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class (overview)?

Overview

1. Introduction

2. History of information retrieval

3. Boolean model

4. Inverted index

5. Processing queries

6. Query optimization

7. Course overview and agenda

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).

Prerequisites

Curiosity about how Information Retrieval works.
But seriously:

- Chapters 1–5 benefit from basic course on algorithms and data structures.
- Chapters 6–7 need in addition linear algebra, vectors and dot products.
- For Chapters 11–13 basic probability notions are needed.
- Chapters 18–21 demand course in linear algebra, notions of matrix rank, eigenvalues and eigenvectors.

PV211 course design: proactive rather than reactive learning, diversity is good, learning by doing, skillful rather than bag of facts, Stanford inspired

- Student [soft skills and programming] activities are *explicitly welcomed* and built as part of classification system (10 pts).
- Mentoring rather than 'ex cathedra' lectures: "The *flipped classroom* is a pedagogical model in which the typical lecture and homework elements of a course are reversed."
- Respect to individual learning speed and knowledge.
- Questions are welcome—on PV211 IS discussion forum *before* lectures, and also during lectures.
- Richness of materials available in advance: MOOC (Massive open online course) becoming widespread, parts of IIR Stanford courses being available, together with other freely available teaching materials, including the whole IIR book, Google Colab notebooks,....

Teachers

- Petr Sojka, `sojka@fi.muni.cz`
- Consulting hours Spring 2020:
  Thursday 9:00–9:50 or by appointment by email.
- Room C523 (or C522 or A502), fifth floor, Botanická 68a.
- Course web page:
  `https://www.fi.muni.cz/~sojka/PV211/`
- Teaching assistants: Vít Novotný, `witiko@mail.muni.cz`,
  Dávid Lupták, `422640@mail.muni.cz`, Michal Štefánik,
  `stefanik.m@mail.muni.cz` and Eniafe Festus Ayetiran,
  `243532@mail.muni.cz`. All TAs are seated in B203/B206
  and are ready for consultations by appointment.

Evaluation of students

Classification is based on points you could get a) **50 pts** during the term: **28 pts** for the term programming project, **12 pts** for term project peer review, **10 pts** for your activities in discussion forum and during lectures evaluated by teacher(s) at the end of term, and b) **50 pts** for the final test: **30 pts** for open exercises, similar to those practiced at exercises, **20 pts** for multiple-choice test. *In addition*, one can get additional *premium* points based on activities during lectures, exercises (good answers) or negotiated related projects. Classification scale lower bounds for passing z/k/zk are 50/57/64 points and E–A grading will be based on ECTS suggestion in IS, E/D/C/B/A corresponds $\approx$ to 71/78/85/92/100 points. Dates of [final] exams will be announced via IS.muni.cz (at least three terms).

Questions?

Term projects and their student peer reviews I

Until 29. 4. 12:00, your tasks awarded up to **28 pts** will be the following:

1. Individually implement a ranked retrieval system for Cranfield collection.

2. Document your code and stick to an organized, consistent, human-readable coding style.

3. Reach at least 35% mean average precision.

4. Upload a link to your Google Colaboratory document to the homework vault in IS MU.
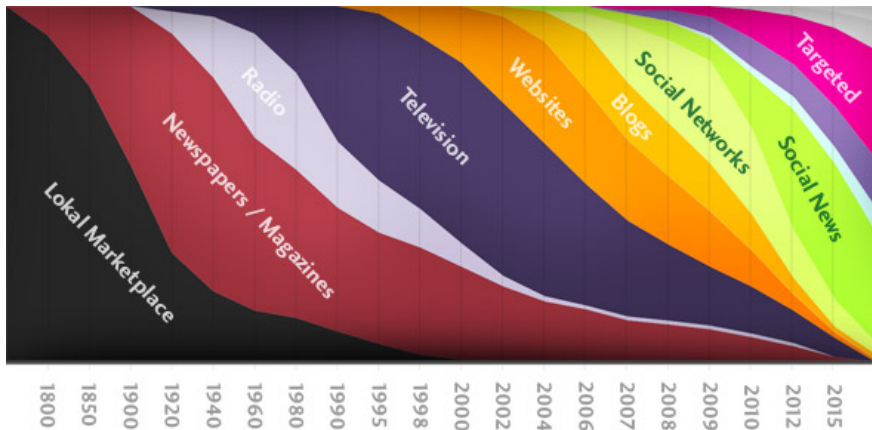
Term projects and their student peer reviews II

For detailed instructions and an example solution, see the Google Colaboratory document linked from the interactive course syllabus in IS MU. The link to a project leaderboard is also in the syllabus. Between 29. 4. and 6. 5., your task awarded up to $3 \times 4 = $ **12 pts** will be to review the term projects of three of your colleagues. One point will be awarded for handing in a review of your colleague's term project. Three points will be awarded for reviewing the completion of tasks 1, 2, and 3 in your colleague's term project. You can get up to extra 50/20/10/9/8/7/6/5/4/3/2/1 point(s) for the 1st/2nd/3rd/4th/.../12th place in the leaderboard on 29. 4. 12:00.

Can we proceed [Y/N]?
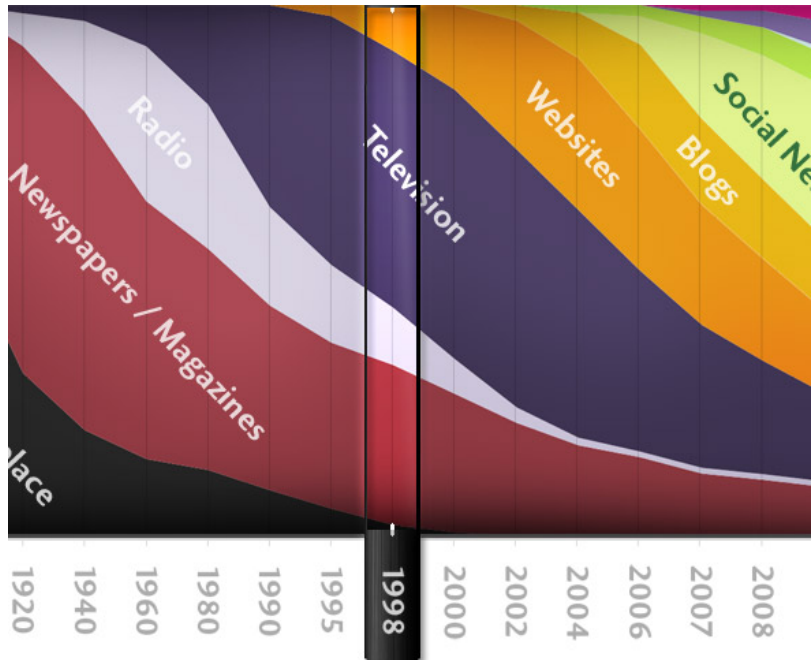
Questions?

Python?   Jupyter Notebook?   Google Colab?   Deepnote project?   Bc./Mgr./Ph.D.?   Mandatory course?   Erasmus? Nationalities: CZ?, SK?, EN=C2 (mother tongue)?, other?   2 midterms or a programming project/challenge?   Student peer reviews?   Mikolov?   Řehůřek?   Materna?   Jurových? Presentation style? traditional? or agile/interactive [warm ups, Kahoot])?   PV211 discussion forum in IS!

History of *information retrieval*: gradual changes of channels

Gradual *speedup* of changes in IR

Newspapers / Magazines

Radio

Television

Websites

Blogs

Social Ne[twork]

...lace

1920 1940 1960 1980 1990 1995 1998 2000 2002 2004 2006 2007 2008
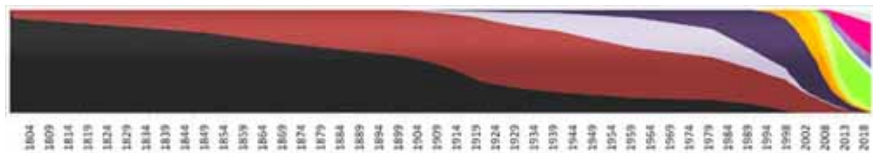
"Google" Circa 1997 (google.stanford.edu)

Google

1998: google.stanford.edu

- collaborative project with Stanford faculty ('flipped IS' :-)
- on collected disks
- Google 1998 'Anatomy paper' (Page, Brin)

Television
Websites
Blogs
Social Networks
Social News
Podcasts / Vodcasts
Gaming
Targeted

1990 1995 1998 2000 2002 2004 2006 2007 2008 2009 2010 2015 2020
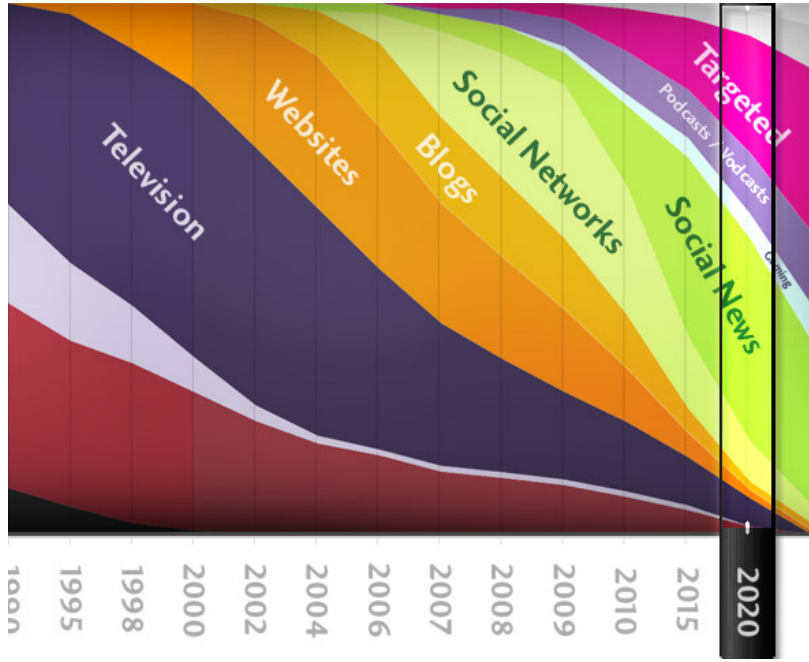
Unstructured (text) vs. structured (database) data in 1996
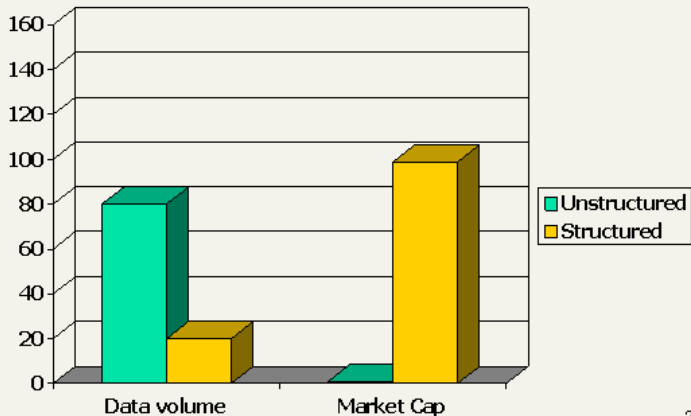
Unstructured (text) vs. structured (database) data in 2006
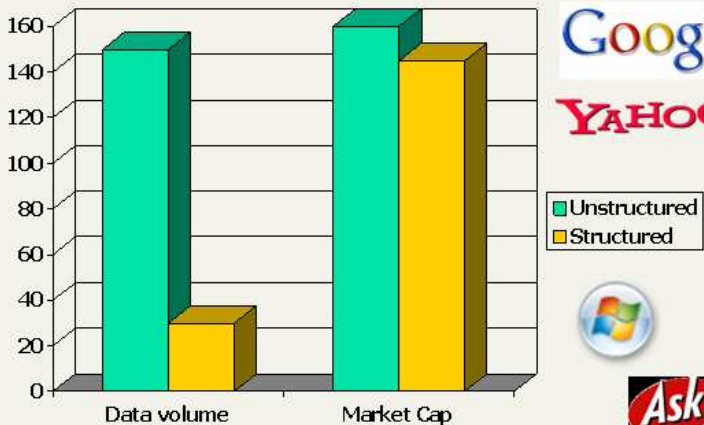
Boolean retrieval

- 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?

Does Google use the Boolean model?

- On Google, the default interpretation of a query $[w_1\ w_2 \ldots w_n]$ is w_1 AND w_2 AND ... AND w_n
- Cases where you get hits that do not contain one of the w_i:
  - anchor text
  - page contains variant of w_i (morphology, spelling correction, synonym)
  - long queries (n large)
  - boolean expression generates very few hits
- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Unstructured data in 1650: collective works of 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?
  - 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
  - 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	
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:
  - Take the vectors for BRUTUS, CAESAR, and 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	

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.

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.)

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!

Inverted index

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

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:

$\Longrightarrow$

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

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

$\Longrightarrow$

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

$\Longrightarrow$

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

$\Longrightarrow$

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

Split the result into dictionary and postings file
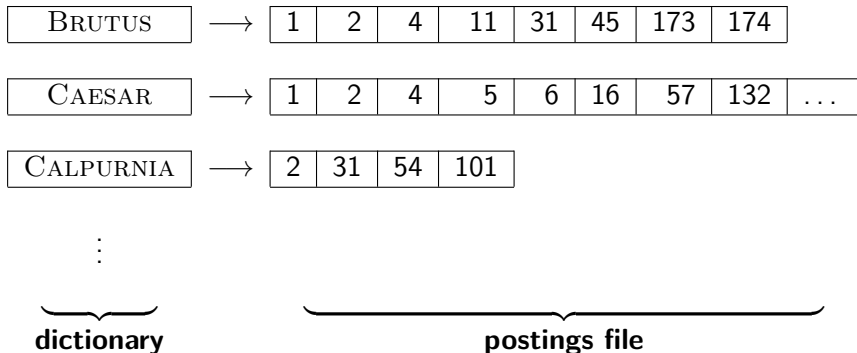
Later in this course

- 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?

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

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

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

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

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

Intersecting two postings lists

$\textsc{Intersect}(p_1, p_2)$

1    $answer \leftarrow \langle \, \rangle$
2   **while** $p_1 \neq \textsc{nil}$ and $p_2 \neq \textsc{nil}$
3   **do if** $docID(p_1) = docID(p_2)$
4   **then** $\textsc{Add}(answer, docID(p_1))$
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** $answer$

Query processing: Exercise

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

PARIS $\longrightarrow$ $2 \rightarrow 6 \rightarrow 10 \rightarrow 12 \rightarrow 14$

LEAR $\longrightarrow$ $12 \rightarrow 15$
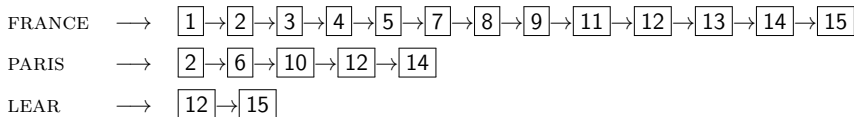
Compute hit list for ((paris AND NOT france) OR lear)

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.
  - 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.

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

- 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,. . .

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?

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

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

CAESAR $\longrightarrow$ $\boxed{5} \to \boxed{31}$
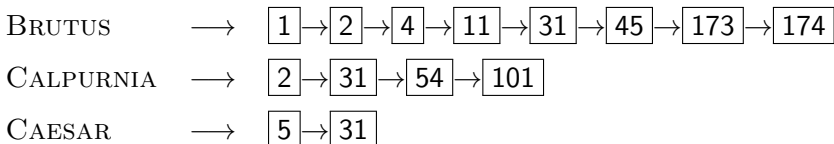
Optimized intersection algorithm for conjunctive queries

Intersect($\langle t_1, \ldots, t_n \rangle$)
1   *terms* ← SortByIncreasingFrequency($\langle t_1, \ldots, t_n \rangle$)
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

Exercise

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

Course overview and agenda

- We are done with Chapter 1 of IIR (IIR 01).
- Plan for the rest of the semester: some 14 of the 21 chapters of IIR
- In what follows: teasers for most chapters – to give you a sense of what will be covered.
- One or two bonus invited lecture(s), and lecture(s) on IR topics researched in my research group MIR.fi.muni.cz and on state-of-the art achievements in the area (vector space embeddings, transformers, etc.).

IIR 02: The term vocabulary and postings lists

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

IIR 03: Dictionaries and tolerant retrieval
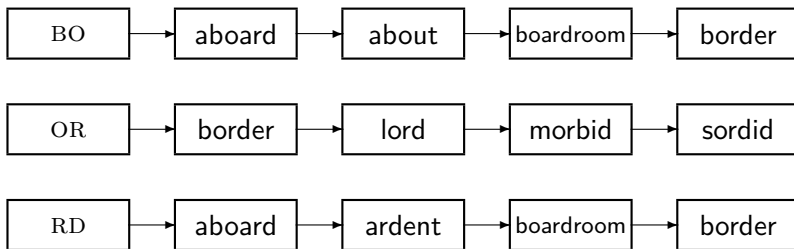
IIR 04: Index construction
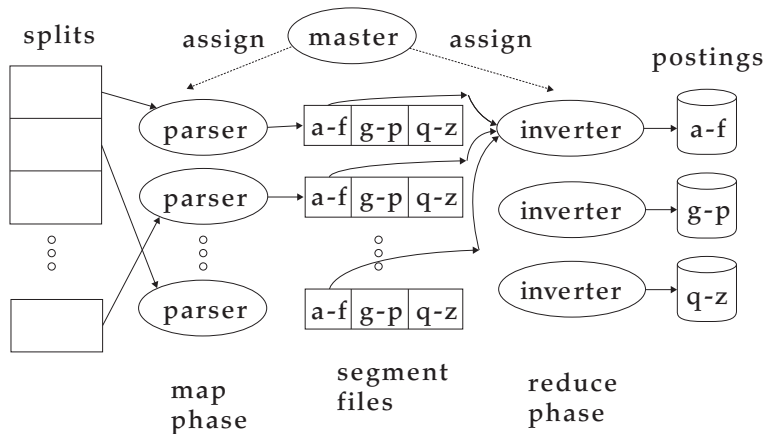
IIR 05: Index compression
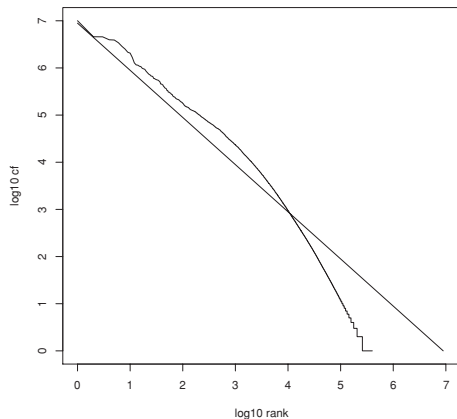
Zipf's law

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 between the query and 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
  - 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
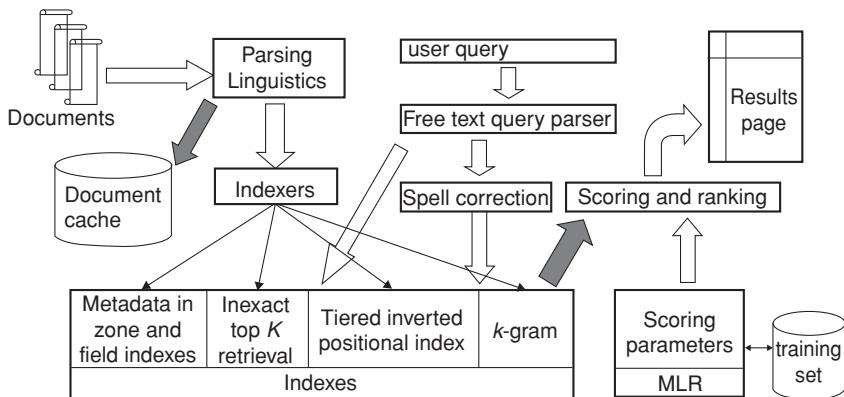
IIR 08: Evaluation and dynamic summaries

IIR 09: Relevance feedback & query expansion

IIR 12: Language models

w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		. . .	. . .

$\rightarrow$ q_1

This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1.

STOP is not a word, but a special symbol indicating that the automaton stops.

frog said that toad likes frog STOP

$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$
$= 0.0000000000048$

IIR 13: Text classification & Naive Bayes

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

IIR 11: Probabilistic information retrieval

document	relevant ($R = 1$)	nonrelevant ($R = 0$)
Term present $x_t = 1$	p_t	u_t
Term absent $x_t = 0$	$1 - p_t$	$1 - u_t$

$$O(R|\vec{q}, \vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0, q_t=1} \frac{1 - p_t}{1 - u_t} \qquad (1)$$

IIR 14: Vector classification, kNN search
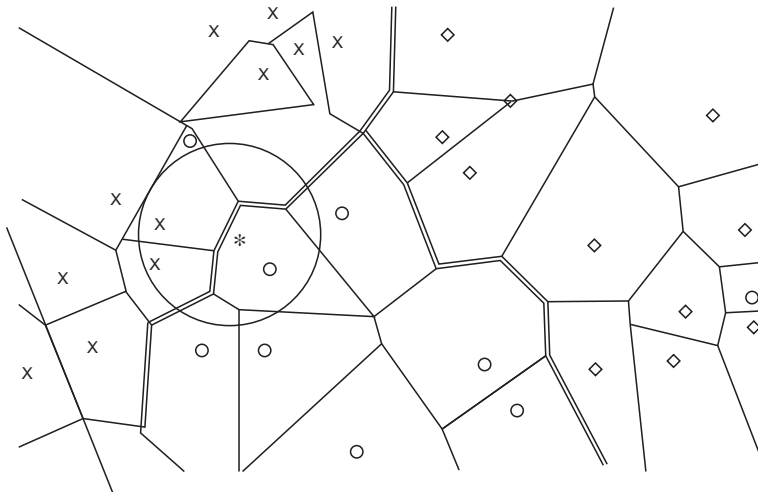
IIR 15: Support vector machines

IIR 16: Flat clustering

IIR 17: Hierarchical clustering
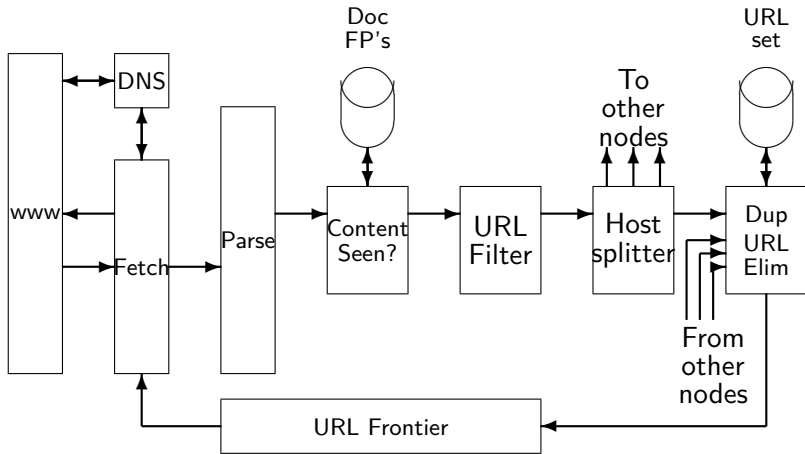
`http://news.google.com`
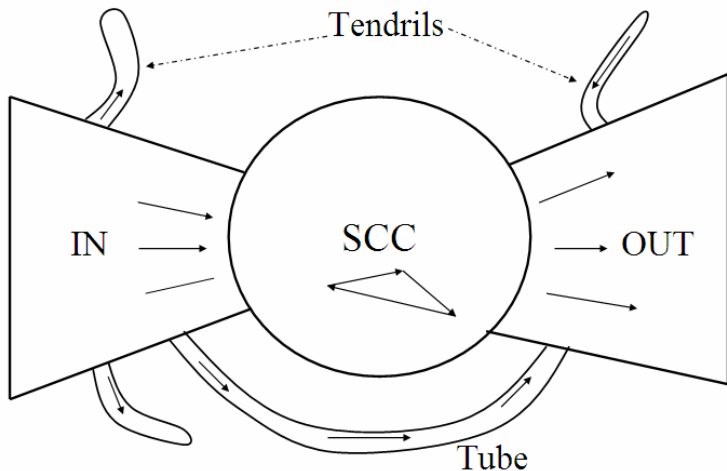
IIR 18: Latent Semantic Indexing
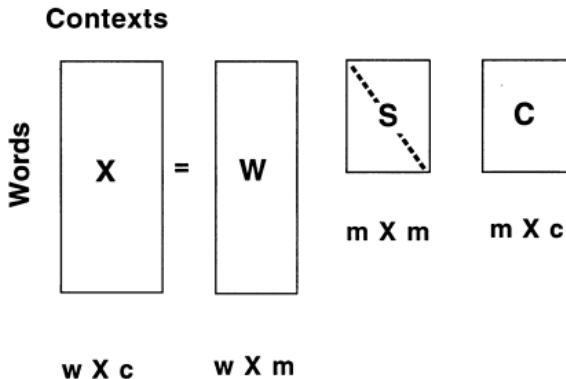
IIR 19: The web and its challenges

- 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 20: Crawling

IIR 21: Link analysis / PageRank

Invited/bonus lecture(s)

Under consideration, yet to be decided:

- MIR group's solution for ARQMath CLEF 2020 tasks: Math information Retrieval Question Answering and Formula searching
- FI MU alumni's talk (Řehůřek, Materna, Jurových,. . .)?
- Informatics colloquium related talk(s): Tomáš Mikolov?

Take-away

- Basic information about the course, teachers, evaluation, exercises
- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class (overview)?

Resources

- Chapter 1 of IIR
- Resources at `https://www.fi.muni.cz/~sojka/PV211/` and `http://cislmu.org`, materials in MU IS and FI MU library
  - course schedule and overview
  - information retrieval links
  - Google Colab environment with examples
  - Shakespeare search engine
    `https://www.rhymezone.com/shakespeare/`