WEB AND SOCIAL INFORMATION EXTRACTION
About this course

- [http://twiki.di.uniroma1.it/twiki/view/Estrinfo/WebHome](http://twiki.di.uniroma1.it/twiki/view/Estrinfo/WebHome)
- (Slides and course material)
- Course is organized as follows:
  - 2/3 “standard” lectures
  - 1/3 Lab:
    - design of an IR system with Lucene,
    - Using Twitter API
    - Implementing a crawler
Use Dropbox to upload your projects homeworks

- Create on [www.dropbox.com](http://www.dropbox.com) a folder
- Name it *NameFamilynameWS2014* (e.g. PaolaVelardiWS2014)
- Share the folder with me and the assistant professor:
  - velardi@di.uniroma1.it
  - stilo@di.uniroma1.it
- DO THAT TODAY
- As I receive your email, I can create a mailing list, to send you homeworks by email
- Sign to Google group on web page
Lectures

- Part I: web information retrieval
  - Architecture of an information retrieval system
  - Text processing, indexing
  - Ranking: vector space model, latent semantic indexing
  - Web information retrieval: browsing, scraping
  - Web information retrieval: link analysis (HITS, PageRank)

- Part II: social network analysis
  - Modeling a social network: local and global measures
  - Community detection
  - Mining social networks: opinion mining, temporal mining, user profiling
PART I
INFORMATION RETRIEVAL: DEFINITION AND ARCHITECTURE
Information Retrieval is:

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from large collections (usually stored on computers).
- "Usually" text, but more and more: images, videos, data, services, audio..
- "Usually" unstructured (= no pre-defined model) but: Xml (and its dialects e.g. Votecxml..), RDF, html are "more structured" than txt or pdf
- "Large" collections: how large?? The Web! (The Indexed Web contains at least 1.78 billion pages (Tuesday, 18 February, 2014).)
Indexed pages (Google)
Unstructured (text) vs. structured (database) data in 1996 (volume & capital)

The business was on structured data

26/02/15
Unstructured (text) vs. structured (database) data from 2007 to 2014 (exabyte)

Blue= unstructured
Yellow=structured
Total Enterprise Data Growth 2005-2015

The business is now unstructured data!
IR vs. databases:
Structured vs unstructured data

- Structured data tends to refer to information in “tables”

<table>
<thead>
<tr>
<th>Employee</th>
<th>Manager</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Jones</td>
<td>50000</td>
</tr>
<tr>
<td>Chang</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>Ivy</td>
<td>Smith</td>
<td>50000</td>
</tr>
</tbody>
</table>

Typically allows numerical range and **exact match** (for text) queries, e.g.,

*Salary < 60000 AND Manager = Smith.*
Unstructured data

- Typically refers to free-form text
- Allows:
  - Keyword queries including operators
    - \((\text{information} \land (\text{retrieval} \lor \text{extraction}))\)
  - More sophisticated “concept” queries, e.g.,
    - find all web pages dealing with drug abuse
Semi-structured data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- This structure allows for “semi-structured” search such as
  - *Title* contains “data” AND *Bullets* contain “search”
  - Only **plain txt format** is truly unstructured (though even natural language does have a structure..)

26/02/15
Not only text retrieval: other IR tasks

- **Clustering**: Given a set of docs, group them into clusters based on their contents.
- **Classification**: Given a set of topics, plus a new doc $D$, decide which topic(s) $D$ belongs to (e.g. spam-nospam).
- **Information Extraction**: Find all snippets dealing with a given topic (e.g. company merges)
- **Question Answering**: deal with a wide range of question types including: fact, list, definition, How, Why, hypothetical, semantically constrained, and cross-lingual questions
- **Opinion Mining**: Analyse/summarize sentiment in a text (e.g. TripAdvisor) (Hot Topic!!)
- All the above, applied to images, video, audio

26/02/15
Terminology

**Searching:** Seeking for specific information within a body of information. The result of a search is a set of **hits** (e.g. the list of web pages matching a query).

**Browsing:** Unstructured exploration of a body of information (e.g. a web browser traverses and retrieves info on the WWW).

**Crawling:** Moving from one item to another following links, such as citations, references, etc.

**Scraping:** pulling content from pages
Terminology (2)

- **Query:** A string of text, describing the information that the user is seeking. Each word of the query is called a search term or keyword.

- A query can be a single search term, a string of terms, a phrase in natural language, or a stylized expression using special symbols.

- **Full text searching:** Methods that compare the query with every word in the text, without distinguishing the function (meaning, position) of the various words.

- **Fielded searching:** Methods that search on specific bibliographic or structural fields, such as author or heading.
Examples of Search Systems

**Find file** on a computer system (e.g. *Spotlight* for Macintosh).

**Library catalog** for searching bibliographic records about books and other objects (e.g. *Library of Congress catalog*).

**Abstracting and indexing system** for finding research information about specific topics (e.g. *Medline* for medical information).

**Web search service** for finding web pages (e.g. *Google*).
Find file
Library Catalogue

Basic Search

Using a fill-in box, search by:
- Title or Author/Creator
- Subject
- Call number
- LCCN, ISSN, or ISBN
- Keywords

Note: Search limits are available only for title and keyword searches.

Guided Search

Using a series of forms and menus:
- Construct keyword searches
- Restrict all or part of the search to a particular index
- Combine search words or phrases with Boolean operators

Note: Search limits are available for all searches.

Information about the images: Two pendentive paintings by Edward J. Holslag are displayed from the Librarian's Room (Librarian's Ceremonial Office) located in the Thomas Jefferson Building of the Library of Congress. On the left, "Efficient clarum studio" (Study, the watchword of fame); on the right, "Dulce ante omnia musae" (The Muses, above all things, delightful).

26/02/15

The Library of Congress
August 19, 2002
Abstracting & Indexing

MEDLINEplus
- Find answers to your health questions ([MEDLINEplus en español](http://www.nlm.nih.gov/hiinfo.html))

MEDLINE/PubMed
- References and abstracts from 4600 biomedical journals

ClinicalTrials.gov
- Provides information for patients about clinical research studies

DIRLINE
- Directory of health organizations

LOCATORplus
- Catalog of books, journals, and audiovisuals in the NLM collections

NIHSeniorHealth
- Health information for older adults

NLM Gateway
- A single Web interface that searches multiple NLM retrieval systems

PubMed Central
- A digital archive of life sciences journal literature

TOXNET
- Databases on toxicology, environmental health, and hazardous chemicals

Tox Town
- An interactive guide to commonly encountered toxic substances, your health, and the environment.
Web Search
ARCHITECTURE OF AN IR SYSTEM
The IR Black Box
Inside The IR Black Box

Query

Representation Function

Query Representation

Comparison Function

Indices

Documents

Representation Function

Document Representation

Hits
More in detail (representation, indexing, comparison, ranking)

How many pages on the web in 2014?

1. User

2. Text Operations

3. Query Operations

4. Indexing

5. Searching

6. Index

7. Ranking

8. User feedback

DB Manager Module

Text Database

More in detail (representation, indexing, comparison, ranking)

How AND many AND page AND web AND 2014

Query Operations

Searching

Retrieved docs

Ranked docs

User feedback
Inside The IR Black Box

[Diagram showing the process of information retrieval with boxes labeled as Query, Representation Function, Query Representation, Comparison Function, Index, Documents, and Document Representation, leading to Hits.]

26/02/15
**Representation:** a data structure describing the content of a document

*In the beginning God created the heaven and the earth. And the earth was without form and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters. And God said, Let there be light: and there was light.*

**The New York Times**

**Tables**

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>beginning</td>
<td>1</td>
</tr>
<tr>
<td>earth</td>
<td>2</td>
</tr>
<tr>
<td>God</td>
<td>3</td>
</tr>
</tbody>
</table>
Inside The IR Black Box

Query Representation 

Representation Function 

Query Representation 

Comparison Function 

Index 

Document Representation 

Representation Function 

Documents 

Hits
Indexing:

a data structure that improves the speed of word retrieval
Inside The IR Black Box
Sorting & Ranking: how well a retrieved document matches the user’s needs?

Eclipse

Solar and Lunar Eclipses of 2010

- Annular Solar Eclipse of 2010 Jan 15
- Partial Lunar Eclipse of 2010 Jun 26
- Total Solar Eclipse of 2010 Jul 11
- Total Lunar Eclipse of 2010 Dec 21

Eclipse

Explore the Eclipse universe...

Enterprise
Java
Eclipse RT
Modeling

Eclipse (I) (2010)
124 min - Adventure | Drama
30 June 2010 (Italy)

Users: (62,911 votes) 467 reviews | C Metascore: 58/100 (based on 38 revie

As a string of mysterious killings high school graduation is fast ap choose between her love for van friendship with werewolf Jacob.

Director: David Slade
Writers: Melissa Rosenberg (sc
(novel)
Eclipse, Twilight Saga: Eclipse

26/02/15
Sorting & ranking

When a **user** submits a **query** to a **search system**, the system returns a set of **hits**. With a large collection of documents, the set of hits may be very large.

The value to the user depends on the order in which the hits are presented.

Three main methods:

- **Sorting** the hits, e.g., by date
- **Ranking** the hits by **similarity** between query and document
- **Ranking** the hits by the **importance** of the documents
More details on

- Representation
- Indexing
- Ranking

(next 3-4 lessons)
1. Document Representation

- Objective: given a document in whatever format (txt, html, pdf..) provide a formal, structured representation of the document (e.g. a vector whose attributes are words, or a graph, or..)
- Several steps from document downloading to the final selected representation
- The most common representation model is “bag of words”
The bag-of-words model

\[ d_i = (\ldots, \ldots \text{after}, \ldots \text{attend}, \ldots \text{both}, \ldots \text{build}, \ldots \text{before}, \ldots \text{center}, \ldots \text{college}, \ldots \text{computer}, \ldots \text{dinner}, \ldots \ldots \ldots \ldots \ldots \text{university}, \ldots \text{work}) \]

WORD ORDER DOES NOT MATTER!!!
Bag of Words Model

- This is the most common way of representing documents in information retrieval

- Variants of this model include:
  - **How to weight a word** within a document (boolean, tf*idf, etc.)
    - **Boolean**: 1 is the word i is in doc j, 0 else
    - **Tf*idf** and others: the weight is a function of the word **frequency** in the document, and of the frequency of documents with that word
  - **What is a “word”:**
    - single, inflected word (“going”),
    - lemmatised word (going, go, gone→go)
    - Multi-word, proper nouns, numbers, dates (“board of directors”, “John Wyne”, “April, 2010”)
    - **Meaning**: (plan, project, design→PLAN#03)
Bag of Words “works” also for images (“words” are now image features)
Phases in document processing (document representation)

1. Document parsing
2. Tokenization
3. Stopwords/Normalization
4. POS Tagging
5. Stemming
6. Deep Analysis
7. Indexing

Notice that intermediate steps can be skipped
1. Document Parsing

Document parsing implies scanning a document and transforming it into a “bag of words” but: which words?

- We need to deal with format and language of each document.
- What format is it in?
- pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically ...
(Doc parsing) Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.

- Sometimes a document or its components can contain multiple languages/formats
  - ex: French email with a German pdf attachment.

- What is a unit document?
  - A file?
  - An email/message?
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)
2. Tokenization

- **Input**: “Friends, Romans and Countrymen”
- **Output**: Tokens
  - *Friends*
  - *Romans*
  - *Countrymen*

A token is an instance of a sequence of characters.
Each such token is now a candidate for an index entry, after further processing.
Described below.

But what are valid tokens to emit?
2. Tokenization (cont’d)

- Issues in tokenization:
  - *Finland’s capital* → *Finland? Finlands? Finland’s?*
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
    - *state-of-the-art*: break up hyphenated sequence.
    - *co-education*
    - *lowercase, lower-case, lower case*?
  - *San Francisco*: one token or two?
    - How do you decide it is one token?
    - *cheap San Francisco-Los Angeles fares*
2. Tokenization : Numbers

- 3/12/91
- Mar. 12, 1991
- 12/3/91
- 55 B.C.
- B-52
- (800) 234-2333
- 1Z9999W99845399981 (package tracking numbers)
  - Often have embedded spaces (ex. IBAN/SWIFT)
  - Older IR systems may not index numbers
    - Since their presence greatly expands the size of the vocabulary
  - Will often index separately as document “meta-data”
    - Creation date, format, etc.
2. Tokenization: language issues

- French & Italian apostrophes
  - *L'ensemble* → one token or two?
    - *L ? L’ ? Le ?*
    - We may want *l’ensemble* to match with *un ensemble*

- German noun compounds are not segmented
  - *Lebensversicherungsgesellschaftsangestellter*
  - ‘life insurance company employee’
  - German retrieval systems benefit greatly from a *compound splitter* module
2. Tokenization: language issues

- Chinese and Japanese have **no spaces between words**:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あたり$500K（約6,000万円）

Katakana  Hiragana  Kanji  Romaji
2. Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right.
- Words are separated, but letter forms within a word form complex ligatures.
- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- Bidirectionality is not a problem if text is coded in Unicode.
Unicode

Unicode

From Wikipedia, the free encyclopedia

For the 1889 Universal Telegraphic Phrase-book, see Commercial code (communications).

Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. Developed in conjunction with the Universal Character Set standard and published as The Unicode Standard, the latest version of Unicode contains a repertoire of more than 110,000 characters covering 100 scripts and multiple symbol sets. The standard consists of a set of code charts for visual reference, an encoding method and set of standard character encodings, a set of reference data computer files, and a number of related items, such as character properties, rules for normalization, decomposition, collation, rendering, and bidirectional display order (for the correct display of text containing both right-to-left scripts, such as Arabic and Hebrew, and left-to-right scripts).[1] As of June 2014, the most recent version is Unicode 7.0. The standard is maintained by the Unicode Consortium.
3.1 Stop words

- With a stop list, you exclude from the dictionary entirely **the commonest words**. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
  - Stop word **elimination used to be standard in older IR systems.**

- But the trend is away from doing this:
  - Good compression techniques means the space for including stopwords in a system is very small
  - Good query optimization techniques mean you **pay little** at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song/books titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”
Example

- Years ago the query “how many pages are there on the web in 2014?” would have been simplified in: “page web 2014” but now all words are preserved (you can check)
3.2. Normalization to terms

- We need to “normalize” words in indexed text as well as query words into the same form
  - We want to match **U.S.A.** and **USA**
- Result is terms: a **term** is a (normalized) word type, which is a **single entry** in our IR system dictionary
- We most commonly implicitly define **equivalence classes** of terms by, e.g.,
  - deleting periods to form a term
    - **U.S.A., USA** ➔ **USA**
  - deleting hyphens to form a term
    - **anti-discriminatory, antidiscriminatory** ➔ **antidiscriminatory**
  - Synonyms (this is rather more complex..)
    - car , automobile
3.2 Normalization: other languages

- Accents: e.g., French *résumé* vs. *resume*.
- Umlauts: e.g., German: *Tuebingen* vs. *Tübingen*
  - Should be equivalent
- Most important criterion:
  - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often *may not type them*.
  - Often best to normalize to a de-accented term
    - *Tuebingen, Tübingen, Tubingen* → *Tubingen*
3.2 Normalization: other languages

- Normalization of other strings like date forms
  - 7月30日 vs. 7/30
  - Japanese use of kana vs. Chinese characters

- Tokenization and normalization may depend on the language and so is interweaved with language detection

- Crucial: Need to “normalize” indexed text as well as query terms into the same form

\[\text{Morgen will ich in MIT} \ldots\]

Is this German “mit”? 

26/02/15
3.2 Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence
    - e.g., *General Motors*
    - *Fed* vs. *fed*
    - *MIT* vs. *mit*
  - Often **best to lower case everything**, since users will use lowercase regardless of ‘correct’ capitalization...

- This may cause different senses to be merged. Often the most relevant is simply the most frequent on the WEB, rather than the most intuitive
Circa 2.080.000.000 risultati (0,32 secondi)

Cat | Prodotti e Servizi - Europe | Caterpillar
www.cat.com/it_IT.html ▼
Questo sito utilizza e installa dei "cookie" sul computer che contribuiscono a migliorarne la qualità. Per informazioni più dettagliate su questi cookie e ...

Caterpillar: Cat | global-selector
www.cat.com/ ▼ Traduci questa pagina
Manufacturer of construction and mining equipment, diesel and natural gas engines, industrial gas turbines, and a wide offering of related services.

Immagini relative a C.A.T.

Segnala immagini non appropriate

Altre immagini per C.A.T.
3.2 Normalization: Synonyms

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - \( car = \text{automobile} \quad \text{color} = \text{colour} \)
  - We can rewrite to form equivalence-class terms
    - When the document contains \( \text{automobile} \), index it under \( \text{car-automobile} \) (and vice-versa)
  - Or we can expand a query
    - When the query contains \( \text{automobile} \), look under \( \text{car} \) as well

- What about spelling mistakes?
  - One approach is \textbf{Soundex}, a phonetic algorithm to encode homophones to the same representation so that they can be matched despite minor differences in spelling
  - Google → Googol
Ricerca

Circa 14.980.000.000 risultati (0,34 secondi)

Tutto

Immagini

Maps

Video

Notizie

Shopping

Più contenuti

Risultati visualizzati per Google

Cerca invece Googel

Google

www.google.it/

Versione italiana del popolare motore e directory. Utilizza inoltre i dati di ODP.

Google Maps

maps.google.it/

Visualizza mappe e trova attività commerciali locali sul Web.

Hai visitato questa pagina 4 volte. Ultima visita: 26/02/12

Googel
4. Stemming/Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
  - Lemmatization implies doing “proper” reduction to dictionary form (the *lemma*).
4. Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggest crude affix chopping
  - language dependent
  - e.g., *automate*(s), *automatic*, *automation* all reduced to *automat*.

*for example compressed and compression are both accepted as equivalent to compress.*

*for exampl compress and compress ar both accept as equivel to compress*
Porter’s algorithm

- Commonest algorithm for stemming English
  - Results suggest it’s at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: out of the rules in a compound command, select the one that applies to the longest suffix.
Typical rules in Porter

- $sses \rightarrow ss$  
  caresses $\rightarrow$ caress
- $ies \rightarrow l$  
  ponies $\rightarrow$ poni
- $SS \rightarrow SS$  
  caress $\rightarrow$ caress
- $S \rightarrow$  
  cats $\rightarrow$ cat

- Weight of word sensitive rules
  
  $\ (m>1) \ EMENT \rightarrow$
  - replacement $\rightarrow$ replac
  - cement $\rightarrow$ cement
Three stemmers: A comparison

**Sample text:** Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

**Porter’s:** such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

**Lovins’s:** such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

**Paice’s:** such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.
5. Deep Analysis

- Has to do with more detailed Natural Language Processing algorithms
- E.g. semantic disambiguation, phrase indexing (board of directors), named entities (President Obama= Barak Obama) etc.
- Standard search engines increasingly use deeper techniques (e.g. Google’s Knowledge Graph [http://www.google.com/insidesearch/features/search/knowledge.html](http://www.google.com/insidesearch/features/search/knowledge.html))
- More (on deep NLP techniques) in NLP course!
Leonardo da Vinci

Leonardo di ser Piero da Vinci è stato un pittore, ingegnere e scienziato italiano. Uomo d'ingegno e talento universale del Rinascimento, incarnò in pieno lo spirito della sua epoca, portandolo alle ... Wikipedia

Nascita: 15 aprile 1452, Vinci

Opere d'arte

- Giocconda: 1506
- Ultima Cena: 1498
- Vergine delle Rocce: 1486
- Battaglia di Anghiari: 1505
- Dama con l'ermellino: 1490

Le persone hanno cercato anche

- Michelangelo Buonarroti
- Raffaello Sanzio
- Pablo Picasso
- Vincent van Gogh
- Sandro Botticelli
1. Document Representation

2. Document Indexing
Why indexing

- The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query.
- Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power.
- For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours.
Inverted index

For each term, we have a list that records which documents the term occurs in. The list is called **posting list**.

What happens if the word *Caesar* is added to document 14?

We need **variable-size postings lists**
Inverted index construction

Documents to be indexed

Token stream

Linguistic modules

Modified tokens

Inverted index

Friends, Romans, Countrymen.

26/02/15
Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

I did enact Julius Caesar; I was killed i'the Capitol; Brutus killed me.

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.
Indexer steps: Sort

- Sort by terms
  - And then “docID”

Core indexing step
Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency? Will discuss later.

Why frequency? Will discuss later.
Where do we pay in storage?

<table>
<thead>
<tr>
<th>term</th>
<th>doc. freq.</th>
<th>postings lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>told</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Lists of doc IDs

Terms and counts

Pointers
The index we just built

- How do we process a query?
Query processing: AND

- Consider processing the query:

**Brutus AND Caesar**

- Locate **Brutus** in the Dictionary;
  - Retrieve its postings (e.g. pointers to documents including Brutus).
- Locate **Caesar** in the Dictionary;
  - Retrieve its postings.
- “Merge” the two postings:
The “merge” operation

- Walk through the two postings simultaneously from right to left, in time linear in the total number of postings entries.

If list lengths are $x$ and $y$, merge takes $O(x+y)$ operations. **Crucial:** postings sorted by docID.
Intersecting two postings lists (a “merge” algorithm)

**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. $p_1 \leftarrow \text{next}(p_1)$
6. $p_2 \leftarrow \text{next}(p_2)$
7. **else if** $docID(p_1) < docID(p_2)$
   8. **then** $p_1 \leftarrow \text{next}(p_1)$
   9. **else** $p_2 \leftarrow \text{next}(p_2)$
10. **return** $answer$
What is the best order of words 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.

Query: **Brutus AND Calpurnia AND Caesar**
Query optimization example

- Process words in order of increasing freq:
  - *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.*
More general optimization

- e.g., *(madding OR crowd) AND (ignoble OR strife)*
- Get doc. freq.’s for all terms.
- Estimate the **size of each** OR by the **sum** of its doc. freq.’s (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)

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>eyes</td>
<td>213312</td>
</tr>
<tr>
<td>kaleidoscope</td>
<td>87009</td>
</tr>
<tr>
<td>marmalade</td>
<td>107913</td>
</tr>
<tr>
<td>skies</td>
<td>271658</td>
</tr>
<tr>
<td>tangerine</td>
<td>46653</td>
</tr>
<tr>
<td>trees</td>
<td>316812</td>
</tr>
</tbody>
</table>

(kaleidoscopeOReyes)AND(tangerineORtrees)AND(marmaladeORskies)
Skip pointers

- Intersection is the most important operation when it comes to search engines.

- This is because in web search, most queries are implicitly intersections: e.g. "car repairs", "britney spears songs" etc. translates into –"car AND repairs", "britney AND spears AND songs", which means it will be intersecting 2 or more postings lists in order to return a result.

- Because intersection is so crucial, search engines try to speed it up in any possible way. One such way is to use skip pointers.
Augment postings with **skip pointers** (at indexing time)

- **Why?**
  - To skip postings that will not figure in the search results.
- **Where do we place skip pointers?**
Query processing with **skip pointers**

Start using the normal intersection algorithm.

Continue until the match **12** and advance to the next item in each list. At this point the "car" list is on **48** and the "repairs" list is on **13**, but **13** has a **skip pointer**.

Check the value the skip pointer is pointing at (i.e. **29**) and **if** this value is **less than the current value of the "car" list** (which it is **48** in our example), we follow our skip pointer and jump to this value in the list.

26/02/15
Where do we place skips?

- Tradeoff:
  - More skips $\rightarrow$ shorter skip spans $\Rightarrow$ more likely to skip. But **lots of comparisons** to skip pointers.
  - Fewer skips $\rightarrow$ few pointer comparison, but then long skip spans $\Rightarrow$ **few successful skips**.

![Diagram of skip list structure]
Placing skips

- Simple heuristic: for postings of length $L$, use $\sqrt{L}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if $L$ keeps changing because of updates.

- How much do skip pointers help?
  - Traditionally, CPUs were slow, they used to help a lot.
  - But today’s CPUs are fast and disk is slow, so reducing disk postings list size dominates.
Phrase queries

- Want to be able to answer queries such as "red brick house"—as a phrase
- red AND brick AND house match phrases such as "red house near the brick factory" which is not what we are searching for
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
  - About 10% of web queries are phrase queries.
- For this, it no longer suffices to store only `<term : docs>` entries
A first attempt: Bi-word 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 are processed as we did with wild-cards:
- *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 4-gram phrase.

Can have false positives!
Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POS Tagging).
- Identify Nouns (N) and articles/prepositions (X).
- Call any string of terms of the form NX*N an **extended biword**.
  - Each such extended biword is now made a **term** in the dictionary.
- Example: *catcher in the rye*
  
  \[ \begin{array}{cccc}
  N & X & X & N \\
  \end{array} \]
- Query processing: parse it into N’s and X’s
  - Segment query into enhanced biwords
  - Look up in index: *catcher rye*
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

- Positional indexes are a more efficient alternative to biword indexes.
- In the postings, store, for each **term** the position(s) in which tokens of it appear:

```plaintext
<term, number of docs containing term;
doc1: position1, position2 ... ;
doc2: position1, position2 ... ; etc.>
```
For phrase queries, we use a merge algorithm recursively at the document level.

But we now need to deal with more than just equality.

Which of docs 1, 2, 4, 5 could contain “to be or not to be”? 

<be: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>
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; ...
- Use NX operator (e.g. N1 if `pos(w2)-pos(w1)=1`)
Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: employment /4 place: Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
  - “Employment agencies that place healthcare workers are seeing growth“ is a hit.
  - “Employment agencies that have learned to adapt now place healthcare workers” is not a hit.
Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
An algorithm for proximity intersection of postings lists $p_1$ and $p_2$. The algorithm finds places where the two terms appear within $k$ words of each other and returns a list of triples giving docID and the term position in $p_1$ and $p_2$. 

```plaintext
POSITIONALINTERSECT($p_1$, $p_2$, $k$)
1  answer ← ∅
2  while $p_1$ ≠ NIL and $p_2$ ≠ NIL
3    do if $docID(p_1) = docID(p_2)$
4      then $l ← ∅$
5        $pp_1 ← positions(p_1)$
6        $pp_2 ← positions(p_2)$
7        while $pp_1$ ≠ NIL
8          do while $pp_2$ ≠ NIL
9            do if $|pos(pp_1) - pos(pp_2)| ≤ k$
10               then ADD($l$, $pos(pp_2)$)
11           else if $pos(pp_2) > pos(pp_1)$
12               then break
13           $pp_2 ← next(pp_2)$
14        while $l ≠ ∅$ and $|l[0] - pos(pp_1)| > k$
15          do DELETE($l[0]$)
16        for each $ps ∈ l$
17          do ADD(answer, ⟨$docID(p_1)$, $pos(pp_1)$, $ps$⟩)
18             $pp_1 ← next(pp_1)$
19        $p_1 ← next(p_1)$
20        $p_2 ← next(p_2)$
21    else if $docID(p_1) < docID(p_2)$
22      then $p_1 ← next(p_1)$
23        else $p_2 ← next(p_2)$
24  return answer
```
Example (a,b k=2)

1: 1 2 3 4 5 6 7 8 9

a x b x x b a x b

l=<3>

<1,1,3>

l=<3,6>

l=<6>, <1,8,6>

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

<table>
<thead>
<tr>
<th>Document size</th>
<th>Postings</th>
<th>Positional postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100,000</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Sec: 2.4.2 26/02/15
Positional index size

- Positional index expands postings storage *substantially*
  - some rough rules of thumb are to expect a positional index to be 2 to 4 times as large as a non-positional index
- Positional index is now *standardly used* because of the power and usefulness of phrase and proximity queries
Combined scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
Google indexing system

- Google is changing the way to handle its index continuously
- See an history on: [http://moz.com/google-algorithm-change](http://moz.com/google-algorithm-change)
Caffeine+Panda, Google Index

- Major recent changes have been Caffeine & Panda
- Caffeine:
  - Old index had several layers, some of which were refreshed at a faster rate than others (they had different indexes); the main layer would update every couple of weeks ("Google dance")
  - Caffeine analyzes the web in small portions and update search index on a continuous basis, globally. As new pages are found, or new information on existing pages, these are added straight to the index.
- Panda: aims to promote the high quality content site by dooming the rank of low quality content sites.