

Document Modeling, Indexing & Matching

– Content Based

Peter Dolog

dolog@cs.aau.dk

<http://people.cs.aau.dk/~dolog>

Based (heavily) on Stanford slides by Christopher Manning & Pandu Nayak and on the 'Introduction to Information Retrieval' book (Chap. 1,2) by Christopher Manning, Prabhakar Raghavan & Hinrich Schütze.

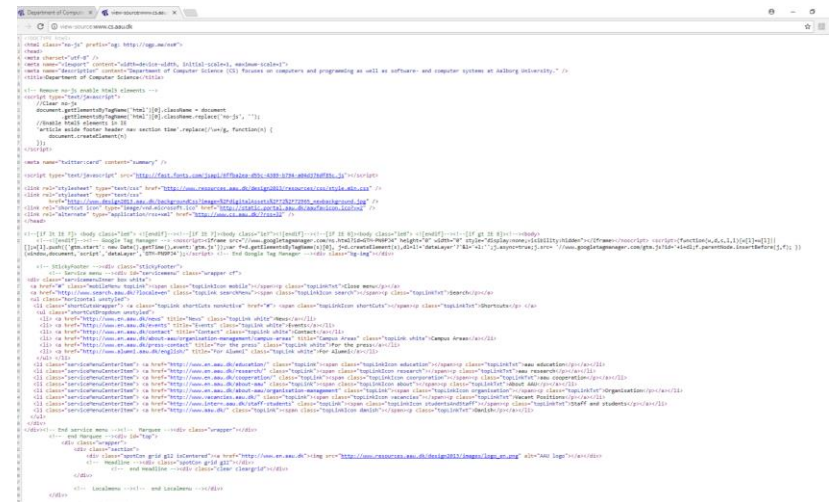
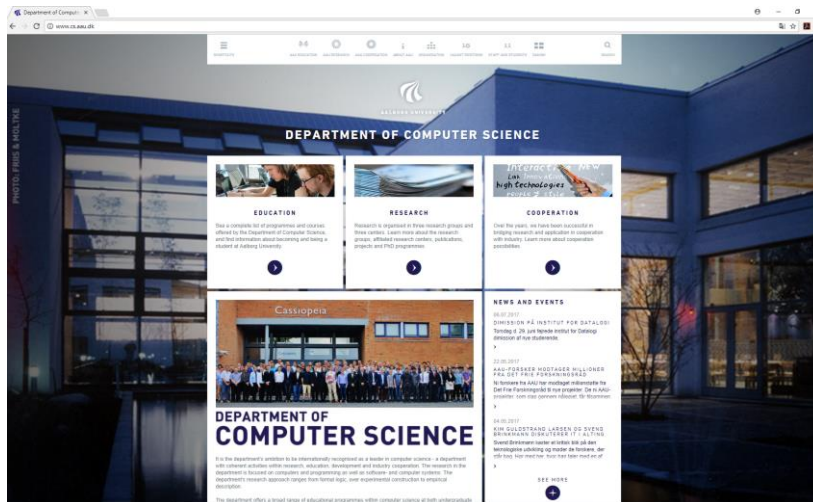
Outline

- Web search basics (short review)
- Document pre-processing / feature construction
- Index construction – the “Inverted index”
- Boolean query processing model
 - standard
 - Phrase
 - Positional

<http://www.cs.aau.dk/en/welcome/>

User's view

Browser/crawler's view



Content-based Information Retrieval

Input: Document collection (aka. Corpus)

Goal: Retrieve documents or text with information **content** that is **relevant** to user's **information need**.

Three aspects:

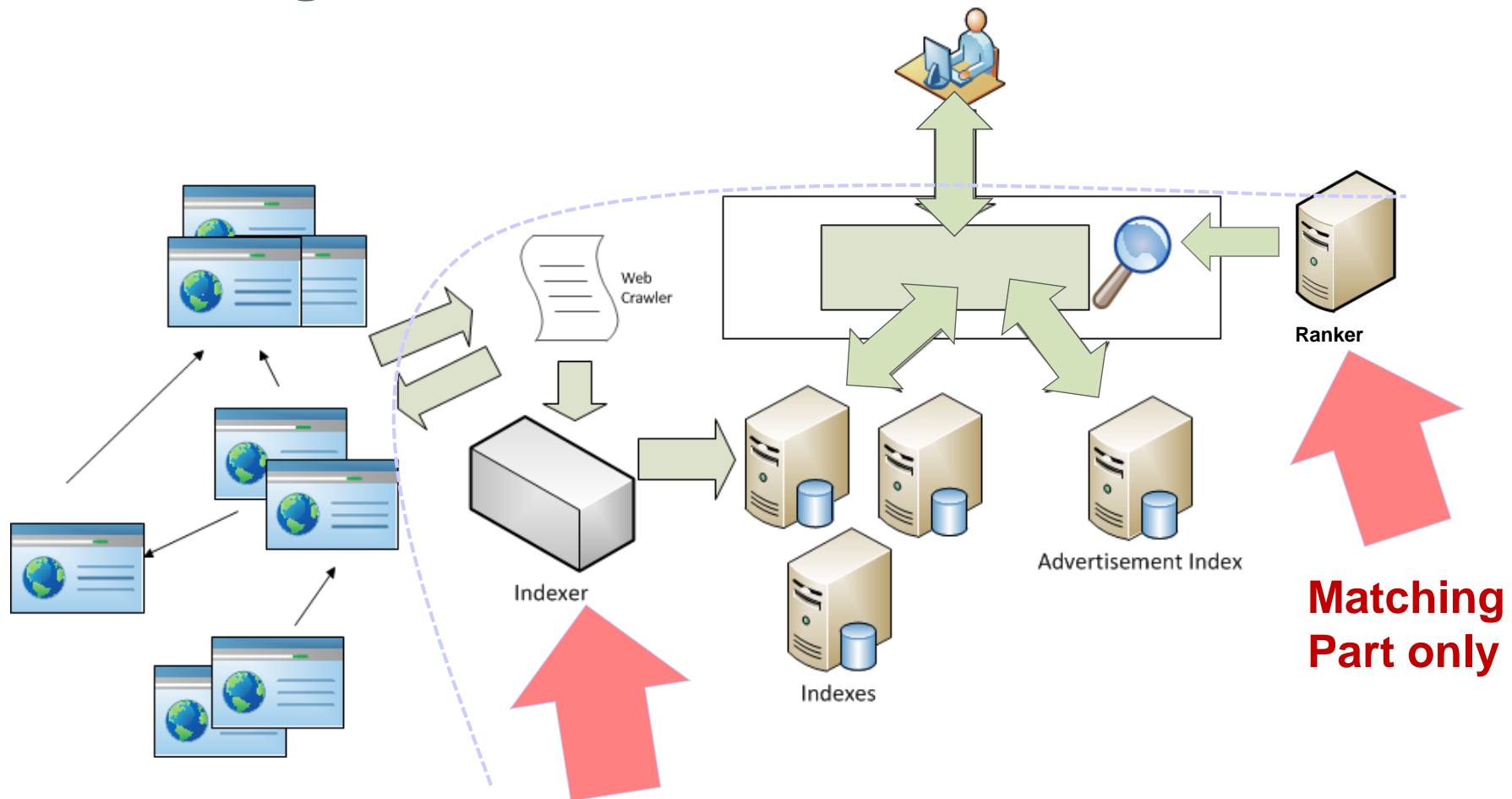
- Document representation (**feature construction**)
- Processing the corpus (**indexing**)
- Processing the queries (**matching** & ranking)

Information needs:

- Web search
- E-mail search
- Searching your laptop
- Corporate knowledge bases
- Legal, health, etc. information retrieval
- ...

Notice: **not** including quality/authoritativeness yet!

Search Engine Architecture



Document Representation

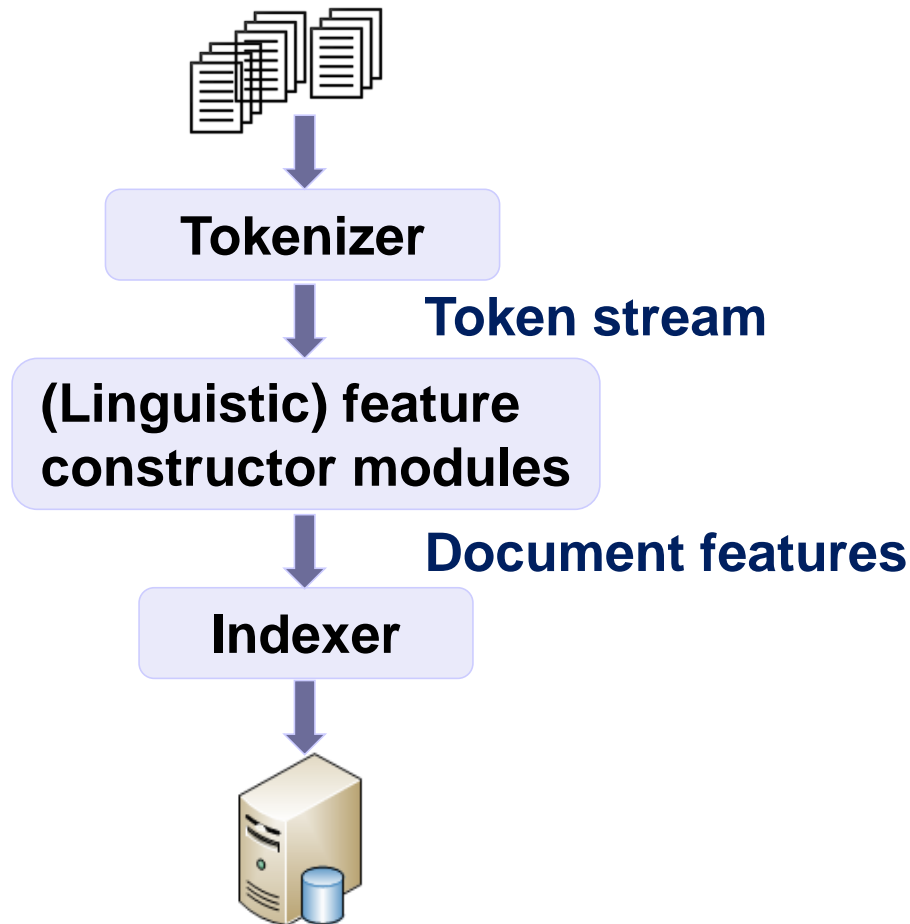
Web documents consist of different formats

- Text, graphics, audio and video
- Represented by intelligently selected **features**
- **We will focus on unstructured text**

Text reduced to a list of “keywords” (**bag-of-words**)

- Simplistic representation of a document
- Yet, the most common representation in Information Retrieval

Basic Representation & Indexing



The department's ambition is to be recognized

The departments ambition is to be recognized

depart ambit recogn

depart	→	3 → 17 → 34
ambit	→	1 → 3 → 9 → 15 → 91
recogn	→	11 → 15

Tokenization

Input: Document

The department's ambition is to be recognized

Output: Tokens

The departments ambition is to be recognized

Def: A **token** is an instance of a sequence of characters from input document that are grouped together as a useful semantic unit (e.g., separated by spaces, tabs, new-line, and other special characters)

Resulting tokens are passed on to further (linguistic) feature construction

Tokenizer: Complications

Parsing a document

- What format is it in
 - html/pdf/word/...?
 - Multiple formats in same documents (e.g., attachments)
- What language is it?
 - Multiple languages in same document?
 - Left-to-right (English), Right to left (Arabic)
- What character set is used?
 - UTF-8, UTF-16, ANSI, Unicode,...?

Tokenizer: Complications (cont.)

- Apostrophes
 - department's \Rightarrow department AND s? departments? department's?
- Hyphens
 - Hewlett-Packard \Rightarrow HewlettPackard? Hewlett Packard? Hewlett-Packard?
 - lowercase, lower-case, lower case?
- Aalborg East \Rightarrow one token or two?
- Numbers - Date/Time/Phone...
 - Sep. 3, 2014, 03-09-2014, 9/3/14...?
- Compounds
 - Lebensversicherungsgesellschaftsangestellter (german)
 - Atombombeprøvesprængning (danish)
- No white spaces between words (East Asian)
 - 部門的志向是國際公認 (The departments ambition is to be recognized internationally???)
- Etc.

Normalization

- Maps class of equivalent tokens into same term
e.g., U.S.A, USA, United States of America \Rightarrow USA
- Normalization of words in indexed text **MUST** be the same as normalization of query
- Which form? Well, what is the preferred form among your users when they write their queries!

Stop words

With a stop list, you exclude the *most common* words.

Intuition:

- They have little semantic content: *the, a, and, to, be*
(*may skew quality search*)
- There are a lot of them: ~30% of postings for top 30 words
(*slows down the search, take up a lot of space*)

On the other hand, you may need them for:

- Phrase queries: “King of Denmark”
- “Relational” queries: “flights to London”
- Etc.

English stopwords list

(<http://www.ranks.nl/stopwords>)

a	could	her	most	should	under	won't
about	couldn't	here	mustn't	shouldn't	until	would
above	did	here's	my	so	up	wouldn't
after	didn't	hers	myself	some	very	you
again	do	herself	no	such	was	you'd
against	does	him	nor	than	wasn't	you'll
all	doesn't	himself	not	that	we	you're
am	doing	his	of	that's	we'd	you've
an	don't	how	off	the	we'll	your
and	down	how's	on	their	we're	yours
any	during	i	once	theirs	we've	yourself
are	each	i'd	only	them	were	yourselves
aren't	few	i'll	or	themselves	weren't	
as	for	i'm	other	then	what	
at	from	i've	ought	there	what's	
be	further	if	our	there's	when	
because	had	in	ours	these	when's	
been	hadn't	into	ourselves	they	where	
before	has	is	out	they'd	where's	
being	hasn't	isn't	over	they'll	which	
below	have	it	own	they're	while	
between	haven't	it's	same	they've	who	
both	having	its	shan't	this	who's	
but	he	itself	she	those	whom	
by	he'd	let's	she'd	through	why	
can't	he'll	me	she'll	to	why's	
cannot	he's	more	she's	too	with	

Danish stopwords list

(<http://www.ranks.nl/stopwords/danish>)

af	fire	ind	næste
alle	flere	ingen	næsten
andet	fleste	intet	og
andre	for	jeg	op
at	fordi	jer	otte
begge	forrige	kan	over
da	fra	kom	på
de	få	kommer	se
den	før	lav	seks
denne	god	lidt	ses
der	han	lille	som
deres	hans	man	stor
det	har	mand	store
dette	hendes	mange	syv
dig	her	med	ti
din	hun	meget	til
dog	hvad	men	to
du	hvem	mens	tre
ej	hver	mere	ud
eller	hvilken	mig	var
en	hvis	ned	
end	hvor	ni	
ene	hvordan	nogen	
eneste	hvorfor	noget	
enhver	hvornår	ny	
et	i	nyt	
fem	ikke	nær	

Example – Stopword removal



The screenshot shows the website of the Department of Computer Science at Aalborg University. The header includes the university logo and name. Below the header is a navigation bar with links: Welcome, Education, Research, Business, Department, and Contact. The main content area features a large image of students holding a sign that says 'ITC' and 'POP'. To the right of the image is a 'Welcome' section with the following text:

Welcome

The departments ambition is to be recognized internationally as a leading computer science environment.

It is a department with many coherent activities within research, education, technology, development and co- operation with the surrounding society and research dissemination within the computer science area.

On the right side of the 'Welcome' section, there are three vertical text elements: 'Late', 'No n', and 'New:'.

Case folding

Reduce all letters to lower case

- exception: upper case in mid-sentence?
 - e.g., Aalborg University
 - Denmark vs denmark
- Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization...

Anecdotic Google example: [believed to be fixed in 2011... but]

- Query C.A.T.
- #1 result in my browser is Wikipedia article for “cats” not Caterpillar Inc.

Thesauri and soundex

Do we handle synonyms and homonyms?

- E.g., by hand-constructed equivalence classes
 - **car** = **automobile** **color** = **colour**
- We can rewrite to form equivalence-class terms
 - When the document contains **automobile**, index it under **car-automobile** (and vice-versa)
- Or we can expand a query
 - When the query contains **automobile**, look under **car** as well

Do we handle spelling mistakes?

- One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Stemming (to the “root”)

A given word may occur in a variety of syntactic forms, such as plurals, past tense, or gerund forms:

- Connect, connector, connection, connections, connected, connecting, connects, preconnection and postconnection

A stem is what is left after its affixes (prefixes and suffixes) are removed

- ed, s, or, ed, ing and ion are suffixes
- pre and post are prefixes
- connect is the stem of connector, connection, etc.

The use of stems will improve retrieval performance

- Users rarely specify the exact forms of the word they are looking for
- Stemming reduces the size of the index
- However, researchers have conflicting opinions on quality from the use of stems (In particular English; better for Spanish, German, Finnish,...)
- Increases recall but harms precision

Porter's Stemming Algorithm

- The most popular algorithm for stemming English text
 - Designed by Martin Porter in 1980
 - 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: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

word sensitive rules:

- $(m > 1)$ *EMENT* →

examples:

- *replacement* → *replac*
- *cement* → *cement*

Example – Porter Stemming

(<http://textanalysisonline.com/nltk-porter-stemmer>)

TextAnalysisOnline

Text Analysis Result -- NLTK Porter Stemmer

Original Text	Analysis Result
The departments ambition is to be recognized internationally as a leading computer science environment. It is a department with many coherent activities within research, education, technology, development and co-operation with the surrounding society and research dissemination within the computer science area.	The depart ambit is to be recogn intern as a lead comput scienc environment. It is a depart with mani coher activ within research , educ , technolog , develop and co-oper with the surround societi and research dissemin within the comput scienc area .

mashape

Consume API

@ 2014 Text Analysis Online

Other stemmers

Other stemmers exist, e.g.:

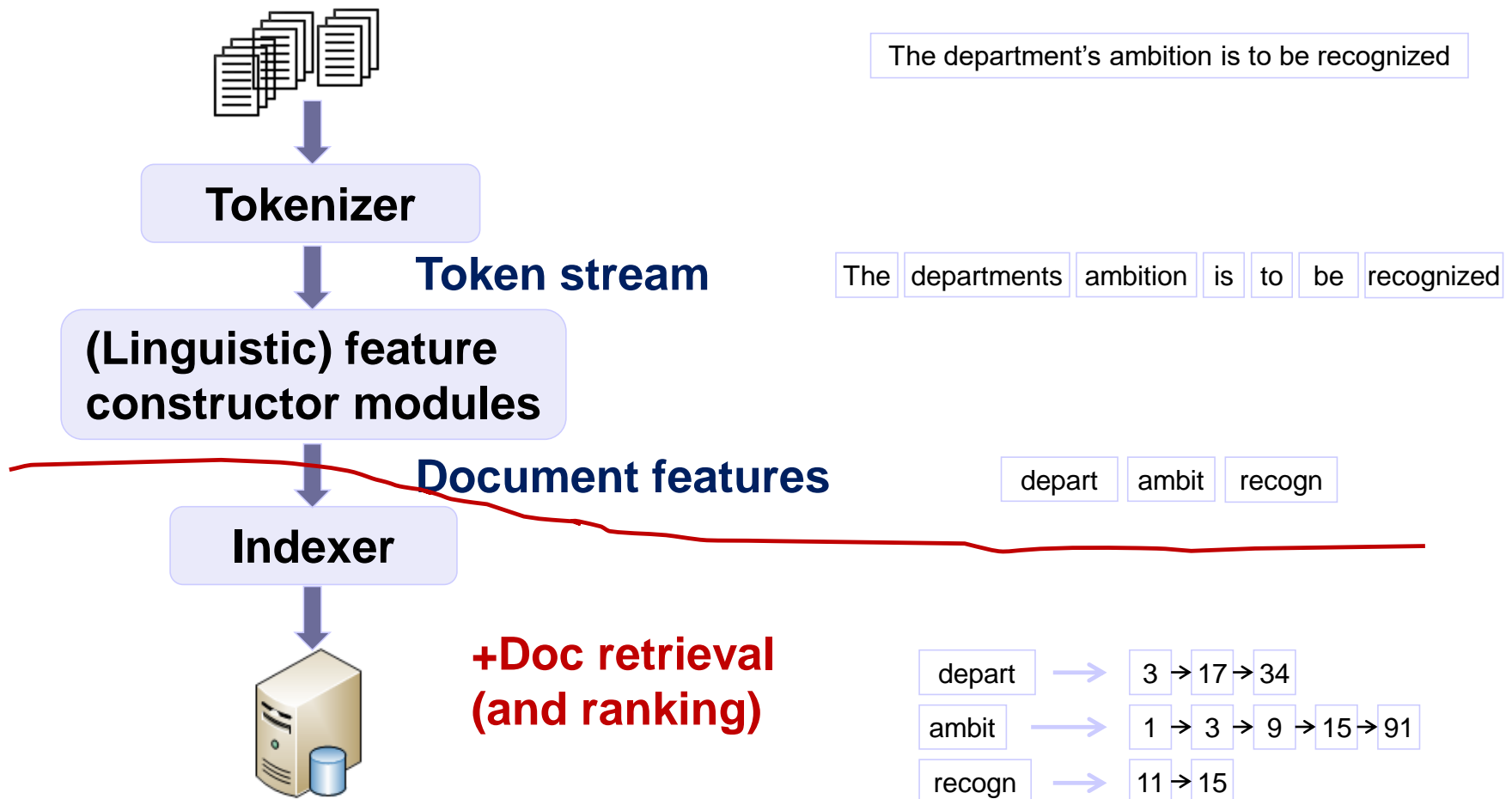
- Lovins stemmer
- Paice/Husk stemmer
- Snowball

Full morphological linguistic analysis (**lemmatization**)

- Closely related to stemming
- Stemmer operates on a single word *without* knowledge of the context, whereas lemmatization may use context
- Stemmers easier to implement and run faster
- No significant benefit for retrieval accuracy (in general)

Good open source and commercial “plug-in”s are available

Basic Representation & Indexing



Indexing

Term Vector

- Corpus contains many documents
- Vocabulary = all distinct words in these documents

Doc

the research in the department has computers, programming, as well as software and computer systems as its field.



Binary

Term	I/O.
:	0
and	1
:	0
as	1
:	0
computer(s)	1
:	0
department	1
:	0
field	1
:	

This lecture

-or-

Frequency

Term	Freq.
:	0
and	1
:	0
as	3
:	0
computer(s)	2
:	0
department	2
:	0
field	1
:	

Next lecture

Binary Term-Document Matrix

Term	Term ID	Doc1	Doc2	Doc3
depart	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
recogn	t4	1	1	1
area	t5	1	0	0
lead	t6	1	0	1
hous	t7	0	1	0

- A 0/1 **term vector** for each document (*typically very large and sparse*)
- A 0/1 **incidence vector** for each term (*typically very large and sparse*)
- Variations – to follow

The Boolean Query Model: Example

Simple model based on **set theory** and **Boolean algebra**

Queries specified as Boolean expressions

To answer query “*depart* \wedge *recogn* \wedge \neg *hous*” perform bitwise AND on incidence vectors

d1	d2	d3
1	1	0
1	1	1
1	0	1
1	0	0

Term	Term ID	Doc1	Doc2	Doc3
depart	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
recogn	t4	1	1	1
area	t5	1	0	0
lead	t6	1	0	1
hous	t7	0	1	0

- 1M documents
- 500K distinct terms among these documents
- 1K words in each document

- 
- Yikes!**

- | | | | | | | | |
|-------------|--------------------------|--|--|--|--|------------|--|
| Kilo | ,000 | | | | | | |
| Mega | ,000 ,000 | | | | | | |
| Giga | ,000 ,000 ,000 | | | | | RAM | |
| Tera | ,000 ,000 ,000 ,000 | | | | | HDD | |
| Peta | ,000 ,000 ,000 ,000 ,000 | | | | | | |

The Inverted Index

The key data structure in most modern information retrieval!

Why INVERTED – what does that mean?

Term	Term ID	Doc1	Doc2	Doc3
depart	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
recogn	t4	1	1	1
area	t5	1	0	0
lead	t6	1	0	1
hous	t7	0	1	0

It simply means that:

- We record 1's in the **incidence vector**
 - i.e., in which documents does a particular term appear.
- And NOT the other way, the 1's in the **term vector**
 - i.e., what are the terms that appear in a particular document

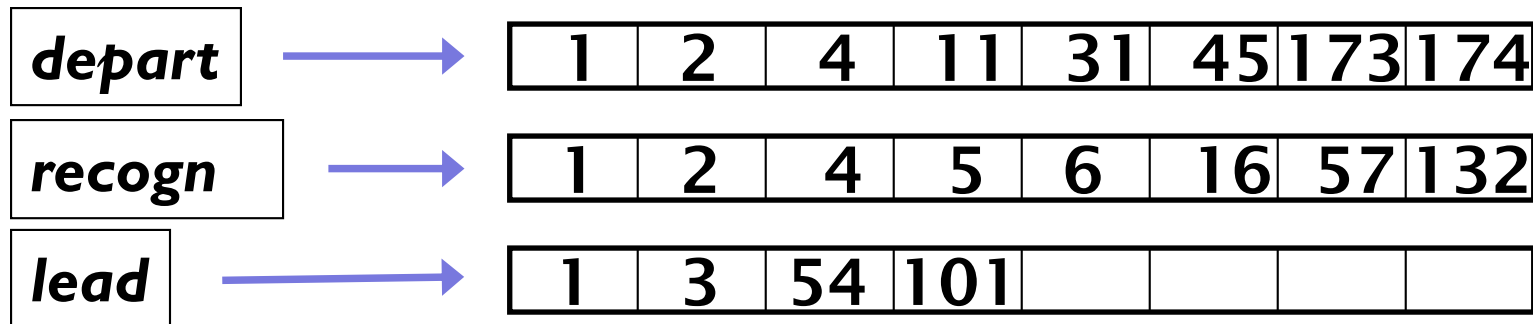
⇒ We do not need to investigate all documents to find matches for a query!!!

Inverted index

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

- Identify each doc by a **docID**

Term	Term ID	Doc1	Doc2	Doc3
depart	t1	1	1	0
activ	t2	0	1	0
play	t3	0	0	0
recogn	t4	1	1	1
area	t5	1	0	0
lead	t6	1	0	1
hous	t7	0	1	0



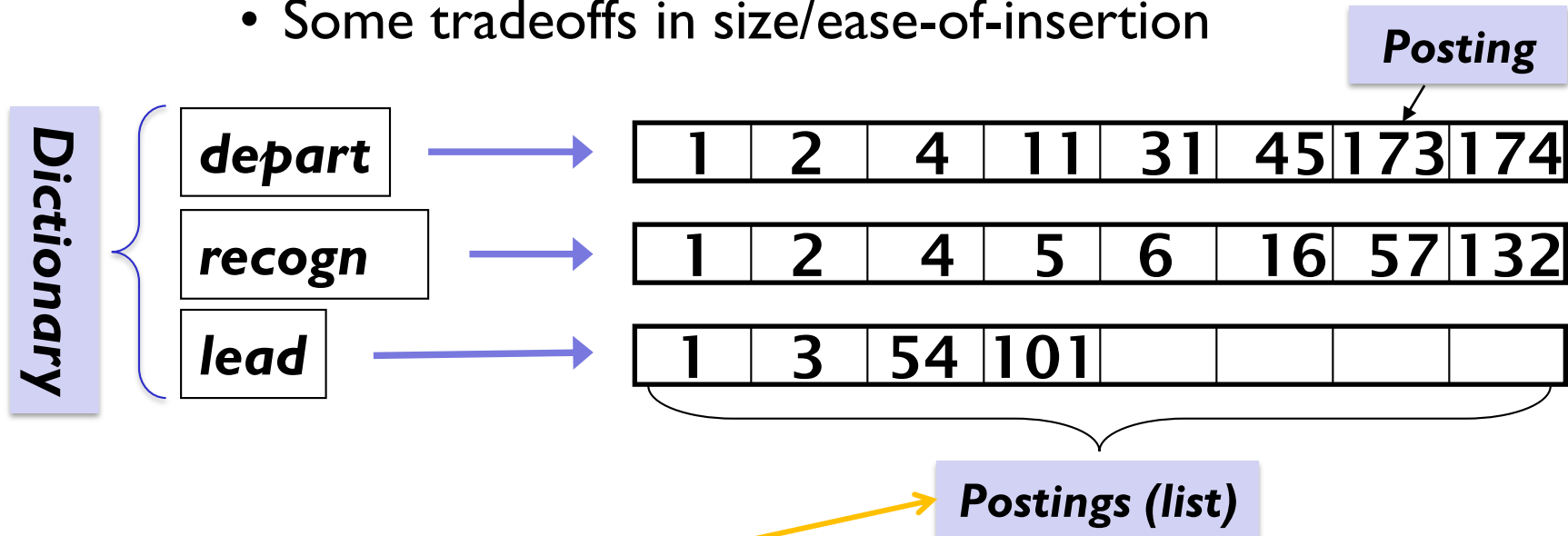
Can we use fixed-size arrays for this? **No, not usually!**

What happens if the term *recogn* is added to document 14?

Inverted index (cont.)

We need variable-size **postings lists**

- On disk, a continuous run of postings is normal and best
- In memory, can use linked lists or variable length arrays
 - Some tradeoffs in size/ease-of-insertion



Inverted Index -- Construction

Index Construction Step I: Term sequence

- Sequence of (Term, Doc ID) pairs.

Doc 1

The departments
ambition is to be
recognized
internationally as
a leading
computer science
environment.

Doc 2

The research in
the department
has computers,
programming, as
well as software
and computer
systems as its
field.



Term	DocID
the	1
departments	1
ambition	1
is	1
to	1
be	1
recognized	1
internationally	1
as	1
a	1
leading	1
computer	1
science	1
environment	1
the	2
research	2
in	2
the	2
department	2
has	2
computers	2
programming	2
as	2
well	2
as	2
software	2
and	2
computer	2
systems	2
as	2
its	2
field	2

Index Construction Step 2: Sort

- Sort by terms
– and then docID

Core indexing step

Term	DocID
the	1
department	1
ambition	1
is	1
to	1
be	1
recognized	1
internationally	1
as	1
a	1
leading	1
computer	1
science	1
environment	1
the	2
research	2
in	2
the	2
department	2
has	2
computer	2
programming	2
as	2
well	2
as	2
software	2
and	2
computer	2
systems	2
as	2
its	2
field	2



Term	DocID
a	1
ambition	1
and	2
as	1
as	2
as	2
as	2
as	2
be	1
computer	1
computer	2
computer	2
department	1
department	2
environment	1
field	2
has	2
in	2
internationally	1
is	1
its	2
leading	1
programming	2
recognized	1
research	2
science	1
software	2
systems	2
the	1
the	2
the	2
to	1
well	2

Index Construction Step 3: Dictionary & Postings

Term	DocID
a	1
ambition	1
and	2
as	1
as	2
as	2
as	2
be	1
computer	1
computer	2
computer	2
department	1
department	2
environment	1
field	2
...	...



Term	Doc. Freq.
a	1
ambition	1
and	1
as	2
be	1
computer	2
department	2
environment	1
field	1

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

Posting lists
→ 1
→ 1
→ 2
→ 1 → 2
→ 1
→ 1 → 2
→ 1 → 2
→ 1
→ 1
→ 2

Why not
1→2→2→2
Will discuss later.

Index Construction: Summary

1. Construct term sequence: (Term, Doc ID) pairs
2. Sort (Term, Doc ID) pairs
 - Primary by Term
 - Secondary by Doc ID
3. Construct Dictionary & Postings

A “trivial”
MapReduce task!

Doc 1

The departments ambition is to be recognized internationally as a leading computer science environment.

Doc 2

The research in the department has computers, programming, as well as software and computer systems as its field.



Term	Doc. Freq.	Posting lists
a	1	→ 1
ambition	1	→ 1
and	1	→ 2
as	2	→ 1 → 2
be	1	→ 1
computer	2	→ 1 → 2
department	2	→ 1 → 2
environment	1	→ 1
field	1	→ 2

Inverted Index – Boolean Query Processing

Boolean queries

- The **Boolean retrieval model** is being able to ask a query that is a Boolean expression:
 - using *AND*, *OR* and *NOT* to join query terms
 - Views each document as a set of words (aka. “bag 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.
- In many search systems you still use Boolean retrieval:
 - Email, library catalog, Mac OS X Spotlight

Boolean Query Processing: AND

- Process the query
 - `aalborg` AND `engineer` result: {13}
- Locate `aalborg` and `engineer` in the dictionary
 - Retrieve postings
- “Intersect”** the two postings (i.e., intersect the document ids)

Term	Doc. Freq.	Posting lists							
<code>aalborg</code>	4	→	1	→	7	→	13	→	54
<code>computer</code>	5	→	1	→	2	→	16	→	32 → 115
<code>department</code>	3	→	1	→	2	→	13		
<code>engineer</code>	5	→	12	→	13	→	15	→	116 → 211

If the list lengths are x and y , the intersection takes $O(x+y)$ operations.

Crucial: postings sorted by docID.

Boolean score algorithm (Intersecting two postings lists)

```
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           $p_1 \leftarrow next(p_1)$   
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  $answer$ 
```

Boolean Query Processing: OR

- Process the query
 - `aalborg` OR `engineer` result: {1,7,12,13,15, 54,116,211}
- Locate `aalborg` and `engineer` in the dictionary
 - Retrieve postings
- “**join**” the two postings (i.e., join the document ids)

Term	Doc. Freq.	Posting lists							
aalborg	4	→	1	→	7	→	13	→	54
computer	5	→	1	→	2	→	16	→	32 → 115
department	3	→	1	→	2	→	13		
engineer	5	→	12	→	13	→	15	→	116 → 211

If the list lengths are x and y , the join takes $O(x+y)$ operations.

Crucial: postings sorted by docID.

Boolean Query Processing: NOT

- Process the query
 - NOTengineer
- Locate engineer in the dictionary
 - Retrieve posting
- Implicit “**complement**” the posting (in principle, now a **very long** posting list!)

Term	Doc. Freq.	Posting lists				
aalborg	4	→ 1	→ 7	→ 13	→ 54	
computer	5	→ 1	→ 2	→ 16	→ 32	→ 115
department	3	→ 1	→ 2	→ 13		
engineer	5	→ 12	→ 13	→ 15	→ 116	→ 211

Boolean Query Processing (cont.)

What about an arbitrary Boolean formula?

(aalborg OR engineer) AND NOT computer

Can we always merge in “linear” time?

- Linear in what?

Mostly yes! – sum of posting lengths for the query terms

But, *NOT* can sometimes create problems!

Boolean Query Processing (cont.)

Evaluation order:

- Join OR's
- **Sequentially process AND's in order of increasing document-freq**
- Recall, that NOT produces a large implicit posting list (can be efficient, if in combination with AND)
- Example: **aalborg** AND **engineer** AND **department**
 \Rightarrow (**department** AND **aalborg**) AND **engineer** result: {13}
- Example: **aalborg** AND NOT **computer** result: {7, 13, 54}

This is why we kept document freq. in dictionary



Term	Doc. Freq.
aalborg	4
computer	5
department	3
engineer	5

Posting lists				
→ 1	→ 7	→ 13	→ 54	
→ 1	→ 2	→ 16	→ 32	→ 115
→ 1	→ 2	→ 13		
→ 12	→ 13	→ 15	→ 116	→ 211

Quiz

(from IIR book)

Recommend a query processing order for

(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)

Which two terms should we process first?

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Phrase & Proximity Queries

Phrase queries

- We want to be able to answer queries such as “**aalborg university**” – as a phrase
- The sentence “*I went to the university in aalborg*” is not a match.
 - 10% of all web queries are **phrase queries**
 - Many more are **implicit phrase queries**
 - E.g., person names entered without surrounding “ ”
 - Proven easy to understand by users; one of the few “advanced search” ideas that actually works
- To accommodate, we need to further expand the inverted index

Solution 1: Biword indexes (aka. bi-grams)

- Index every consecutive pair of terms in the text as a phrase
- For example the text “...software engineer, aalborg university...” would generate the biwords
 - software engineer
 - engineer aalborg
 - aalborg university
- Each of these biwords are now added as dictionary term
- Two-word phrase query-processing is now immediate.
- Longer phrases can be processed by breaking them down
- Example: “software engineer, aalborg university can be broken into the Boolean query on biwords:
software engineer AND engineer aalborg AND aalborg university

Can have false positives!

Issues for biword indexes

False positives, as noted before

Index blowup due to bigger dictionary

Pragmatic solution

Keep N-gram phrases for most used phrase queries in index

- demands processing of query logs (*web usage mining*)

Solution 2: Positional indexes

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

$\langle \text{term}, \text{doc-freq}; \text{doc1}: \text{pos1}, \text{pos2} \dots; \text{doc2}: \text{pos1}, \text{pos2} \dots; \text{etc.} \rangle$

- Is “computer engineer” a match? **Yes**

Term	Doc. Freq.	Posting lists					
aalborg	4	→ 1:11,25	→ 7:10,15,29	→ 13:2,7	→ 54:5,7,16		
computer	5	→ 1:5,9	→ 2:31	→ 16:9	→ 32:4	→ 115:19	
department	3	→ 1:19	→ 2:3,5,9	→ 13:8			
engineer	5	→ 12:6	→ 13:9,16	→ 32:1,5,86	→ 116:2,6,20	→ 211:2,7	

Positional index example

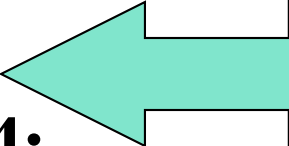
<be: 993427;

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 intersection 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**.
- Intersect 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

Positional index size

A positional index expands postings storage *substantially*

Rules of thumb (for “English-like” languages)

- 2–4 times as large as a non-positional index
- 35–50% of volume of original text

Phrase & Positional Indexing Combination

- Most expensive phrase queries to evaluate in positional index are those where
 - Individual words are common
 - Phrase is rare
 - Implies intersection of large positional posting lists
- Use phrase index terms for
 - Common phrases (“Michael Jackson”, “Britney Spears”)
 - Phrases with common individual terms (“The Who”)



**Learned from
query logs**

Outline

- Web search basics (short review)
- Document pre-processing / feature construction
- Index construction – the “Inverted index”
- Boolean query processing model
 - standard
 - Phrase
 - Positional