

# Introduction to Information Retrieval

<http://informationretrieval.org>

## IIR 2: The term vocabulary and postings lists

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.04.28

- 1 Recap
- 2 The term vocabulary
- 3 Skip pointers
- 4 Phrase queries

1 / 60

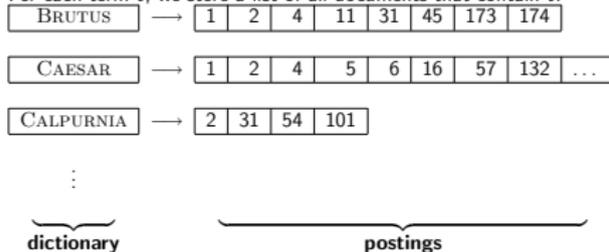
2 / 60

## Outline

- 1 Recap
- 2 The term vocabulary
- 3 Skip pointers
- 4 Phrase queries

## Inverted index

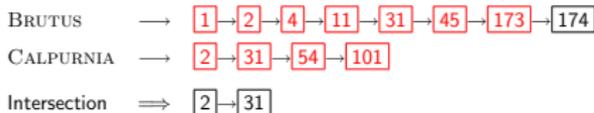
For each term  $t$ , we store a list of all documents that contain  $t$ .



3 / 60

4 / 60

## Intersecting two postings lists



- Linear in the length of the postings lists.

## Constructing the inverted index: Sort postings

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

5 / 60

6 / 60

## 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

## Outline

- 1 Recap
- 2 The term vocabulary
- 3 Skip pointers
- 4 Phrase queries

7 / 60

8 / 60

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
- We'll look a little bit at what a document is.
- But mostly at terms: How do we define and process the vocabulary of terms of a collection?

- Before we can even start worrying about terms . . .
- . . . need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
- What character set is in use?
- Each of these is a classification problem, which we will study later in this course (IIR 13).
- Alternative: use heuristics

9 / 60

10 / 60

## Format/Language: Complications

- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.
  - French email with Spanish pdf attachment
- What is the document unit for indexing?
- A file?
- An email?
- An email with 5 attachments?
- A group of files (ppt or latex in HTML)?

## Terms

11 / 60

12 / 60

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.

- *In June, the dog likes to chase the cat in the barn.*
- How many tokens? How many types?

13 / 60

14 / 60

## Recall: Inverted index construction

## Why tokenization is difficult – even in English

- Input:

Friends, Romans, countrymen. So let it be with Caesar ...

- Output:

friend roman countryman so ...

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Example: *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.* Tokenize this sentence

15 / 60

16 / 60

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers, but generally it's a useful feature.

17 / 60

## Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年4月9日，莎拉波娃在美国第一大城市纽约度过了18岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

19 / 60

## Ambiguous segmentation in Chinese

The two characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

18 / 60

20 / 60

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnangittualuujunga (I can't hear very well.)
- Swedish, Finnish, Greek, Urdu, many other languages

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAI NA Iキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様の日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを800字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて10月20日までにお送りください。大賞受賞者には、50万円相当の旅行券とエコ製品2点の副賞が贈られます。

different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese). End user can express query entirely in hiragana!

21 / 60

22 / 60

ك ت ا ب ← كتاب  
 un b ā t i k  
 /kitābun/ ‘a book’

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

‘Algeria achieved its independence in 1962 after 132 years of French occupation.’ Bidirectionality is not a problem if text is coded in Unicode.

23 / 60

24 / 60

### Back to English

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient
- Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?

25 / 60

26 / 60

### Normalization: Other languages

- Accents: *résumé* vs. *resume* (simple omission of accent)
- Umlauts: *Universität* vs. *Universitaet* (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)
- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

27 / 60

### Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit
- Fed vs. fed
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

28 / 60

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. "King of Denmark"
- Most web search engines index stop words.

29 / 60

## What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

31 / 60

- Soundex: IIR 3 (phonetic equivalence, Tchebyshev = Chebysheff)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

30 / 60

## Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car's, cars'* → *car*
- Example: *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the [lemma](#)).
- Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

32 / 60

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational: *automate*, *automatic*, *automation* all reduce to *automat*

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
    - replacement → replac
    - cement → cement
  - Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

33 / 60

34 / 60

## Porter stemmer: A few rules

Rule		Example	
SSES	→ SS	caresses	→ caress
IES	→ I	ponies	→ poni
SS	→ SS	caress	→ caress
S	→	cats	→ cat

## 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 stemmer:* such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

*Lovins stemmer:* such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

*Paice stemmer:* such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

35 / 60

36 / 60

# Does stemming improve effectiveness?

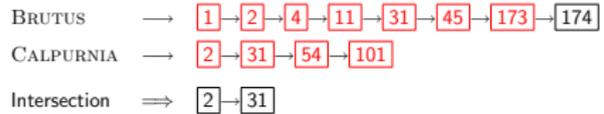
- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operational*.
- Queries where stemming hurts: "operational AND research", "operating AND system", "operative AND dentistry"

Interesting issues in your native language?

## Outline

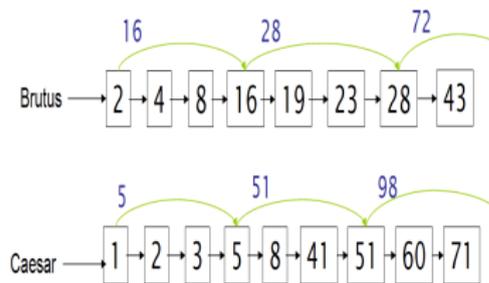
- 1 Recap
- 2 The term vocabulary
- 3 Skip pointers
- 4 Phrase queries

## Recall basic intersection algorithm



- Can we do better?

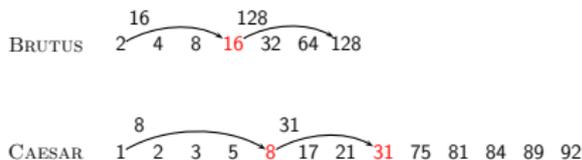
- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if though basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure results don't change?



41 / 60

42 / 60

## Basic idea



## Intersecting with skip pointers

```

INTERSECTWITHSKIPS( $p_1, p_2$ )
1  answer  $\leftarrow$  ( )
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4     then ADD(answer,  $\text{docID}(p_1)$ )
5      $p_1 \leftarrow \text{next}(p_1)$ 
6      $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8     then if hasSkip( $p_1$ ) and ( $\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2)$ )
9         then  $p_1 \leftarrow \text{skip}(p_1)$ 
10        else  $p_1 \leftarrow \text{next}(p_1)$ 
11  else if hasSkip( $p_2$ ) and ( $\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1)$ )
12        then  $p_2 \leftarrow \text{skip}(p_2)$ 
13        else  $p_2 \leftarrow \text{next}(p_2)$ 
14  return answer

```

43 / 60

44 / 60

## Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

## Where do we place skips? (cont)

- Simple heuristic: for postings list of length  $P$ , use  $\sqrt{P}$  evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help lot.
- With today's fast CPUs, they don't help that much anymore.

45 / 60

46 / 60

## Outline

- 1 Recap
- 2 The term vocabulary
- 3 Skip pointers
- 4 **Phrase queries**

## Phrase queries

- We want to answer a query such as "stanford university" – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn't be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: no longer suffices to store docIDs in postings lists.
- Any ideas?

47 / 60

48 / 60

- Index every consecutive pair of terms in the text as a phrase.
- For example, *Friends, Romans, Countrymen* would generate two biwords: “*friends romans*” and “*romans countrymen*”
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.

- A long phrase like “*stanford university palo alto*” can be represented as the Boolean query “STANFORD UNIVERSITY” AND “UNIVERSITY PALO” AND “PALO ALTO”
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

49 / 60

50 / 60

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX\*N to be an *extended biword*
- Examples: catcher in the rye king of Denmark  
N X X N N X N
- Include extended biwords in the term vocabulary
- Queries are processed accordingly

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

51 / 60

52 / 60

## Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a **list of positions**
- Example:  $t_{01} b_{e2} o_{r3} n_{o4} t_{o5} b_{e6}$

TO, 993427:

( 1, 6: (7, 18, 33, 72, 86, 231);  
2, 5: (1, 17, 74, 222, 255);  
4, 5: (8, 16, 190, 429, 433);  
5, 2: (363, 367);  
7, 3: (13, 23, 191); ...)

BE, 178239:

( 1, 2: (17, 25);  
4, 5: (17, 191, 291, 430, 434);  
5, 3: (14, 19, 101); ... ) Document 4 is a match!

## Exercise

Shown below is a portion of a positional index in the format: term: doc1: (position1, position2, ...); doc2: (position1, position2, ...); etc.

ANGELS: 2: (36,174,252,651); 4: (12,22,102,432); 7: (17);  
FOOLS: 2: (1,17,74,222); 4: (8,78,108,458); 7: (3,13,23,193);  
FEAR: 2: (87,704,722,901); 4: (13,43,113,433); 7: (18,328,528);  
IN: 2: (3,37,76,444,851); 4: (10,20,110,470,500); 7: (5,15,25,195);  
RUSH: 2: (2,66,194,321,702); 4: (9,69,149,429,569); 7: (4,14,404);  
TO: 2: (47,86,234,999); 4: (14,24,774,944); 7: (199,319,599,709);  
TREAD: 2: (57,94,333); 4: (15,35,155); 7: (20,320);  
WHERE: 2: (67,124,393,1001); 4: (11,41,101,421,431); 7: (16,36,736);

Which document(s) if any match each of the following two queries, where each expression within quotes is a phrase query? "fools rush in", "fools rush in" AND "angels fear to tread"

55 / 60

54 / 60

## 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 /3 place
- Find all documents that contain EMPLOYMENT and PLACE within 3 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that help place healthcare workers are seeing growth* is not a hit.

## Proximity search

- 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.
- This is important for dynamic summaries etc.

55 / 60

54 / 60

## “Proximity” intersection

```
1 answer ← {}
2 while p1 ≠ NIL and p2 ≠ NIL
3 do if docID(p1) = docID(p2)
4 then I ← {}
5     pp1 ← positions(p1)
6     pp2 ← positions(p2)
7     while pp1 ≠ NIL
8     do while pp2 ≠ NIL
9         do if |pos(pp1) - pos(pp2)| ≤ k
10            then ADD(I, pos(pp2))
11            else if pos(pp2) > pos(pp1)
12                then break
13                pp2 ← next(pp2)
14            while I ≠ {} and |I[0] - pos(pp1)| > k
15                do DELETE(I[0])
16                for each ps ∈ I
17                    do ADD(answer, (docID(p1), pos(pp1), ps))
18                pp1 ← next(pp1)
19            p1 ← next(p1)
20            p2 ← next(p2)
21 else if docID(p1) < docID(p2)
22     then p1 ← next(p1)
23     else p2 ← next(p2)
24 return answer
```

## Combination 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.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.

57 / 60

58 / 60

## “Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?

## Resources

- Chapter 2 of IIR
- Resources at <http://ifnlp.org/ir>
- Porter stemmer

59 / 60

60 / 60