More on indexing and text operations

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2015

Most slides have been adapted from: Profs. Manning, Nayak & Raghavan (CS-276, Stanford)

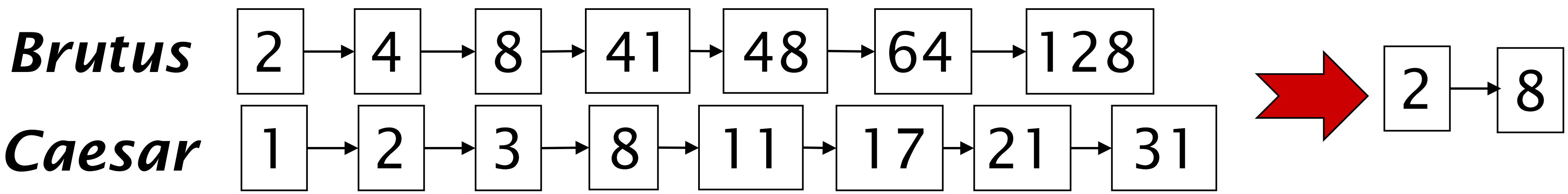
Plan for this lecture

- Elaborate basic indexing
  - Faster merges: skip lists
  - Positional postings and phrase queries

- Text operations: Preprocessing to form the term vocabulary

Recall basic merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries

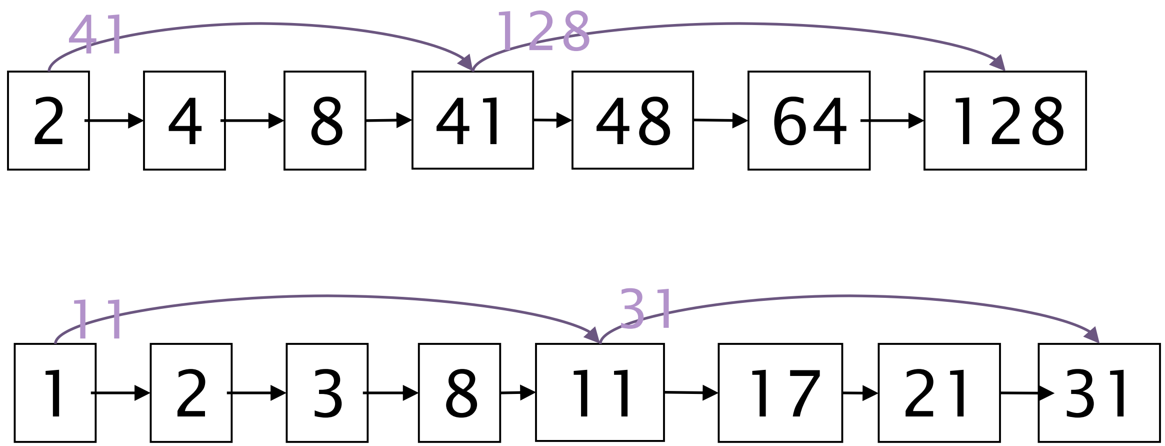


list lengths *m* and *n*, merge takes O(*m*+*n*) operations.

Can we do better?
Yes (if index isn't changing too fast).

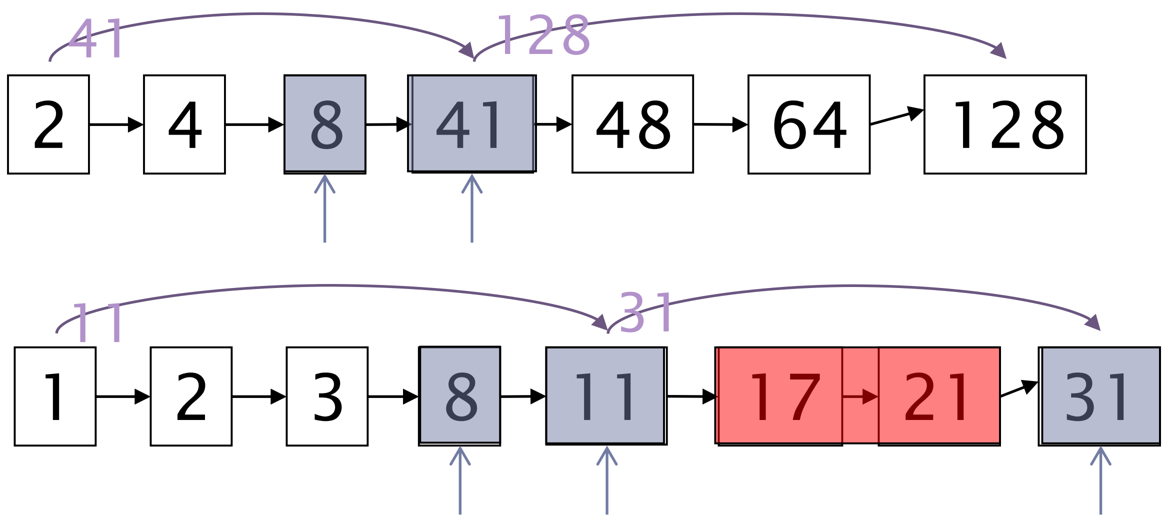
Faster postings merges: skip lists

Augment postings with skip pointers



- To skip postings that will not figure in the results.
- Where do we place skip pointers?

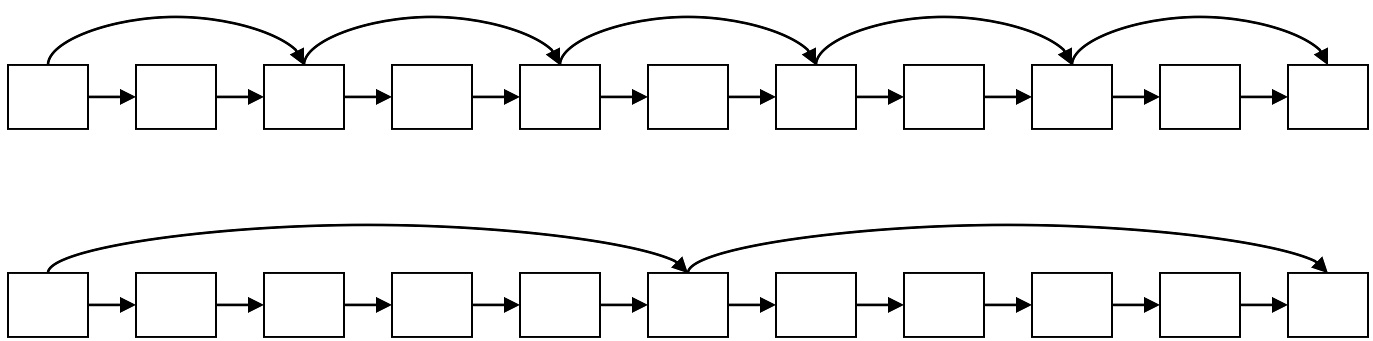
Query processing with skip pointers



- Suppose we are processing **8** on each list. We match it and advance.
- We then have **41** and **11**.
- The skip successor of **11** is **31 (31<41)**. So, we can skip ahead past the intervening postings.

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans
    - More likely to skip but lots of comparisons to skip pointers (and also more space for them)
  - Fewer skips → long skip spans
    - few successful skips but also few pointer comparison (and also less space for them)



Placing skips

- Simple heuristic
  - For posting of length L, use $\sqrt{L}$ evenly-spaced skip pointers
  - Easy if the index is relatively static
- This ignores the distribution of query terms
- This definitely used to help; with modern hardware it may not unless you're memory-based (Bahle et. al 2002)
  - The I/O cost of loading bigger postings list can outweigh the gains from in memory merging

Phrase and proximity queries: positional indexes

Phrase queries

- Example: "***stanford university***"
  - *"I went to university at Stanford"* is not a match.

- Easily understood by users
  - One of the few "advanced search" ideas that works
  - 10% of web queries are phrase queries
  - Many more queries are *implicit phrase queries*
    - such as person names entered without use of double quotes.

- It is not sufficient to store only the doc IDs in the posting lists

Approaches for phrase queries

- Indexing bi-words (two word phrases)
- Positional indexes
  - Full inverted index

Biword indexes

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

Biword indexes: Longer phrase queries

- Longer phrases are processed as conjunction of biwords

Query: ***"stanford university palo alto"***

- can be broken into the Boolean query on biwords:

"stanford university" *AND* ***"university palo"*** *AND* ***"palo alto"***

- Can have false positives!
  - Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POST).
  - Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
- Any string of terms of the form NX*N as an <u>extended biword</u>.
  - Each such extended biword is now made a term in the dictionary.
- Example: "***catcher in the rye***"

  N X X N
  - Look up in index: "***catcher rye***"

Issues for biword indexes

- False positives (for phrases with more than two words)
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for biwords
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Positional index

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

<***term***, doc freq.;

doc1: position1, position2 … ;

doc2: position1, position2 … ; …>

<***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***"?

Positional index

- For phrase queries, we use a merge algorithm recursively at the doc level

- We need to deal with more than just equality of docIDs:
  - **Phrase query**: The lists are traversed in synchronization to find places where all the words appear in sequence
  - **Proximity query**: to find places where all the words close enough

Processing a phrase query: Example

- Query: "***to be or not to be***"
- Extract inverted index entries for: ***to, be, or, not***
- Merge: find all positions of **"to"**, i, i+4, **"be"**, i+1, i+5, **"or"**, i+2, **"not"**, i+3.
  - ***to***:
    - <2:1,17,74,222,551>; <4:8,16,190,429,433, 512>; <7:13,23,191>; ...
  - ***be***:
    - <1:17,19>; <4:17,191,291,430,434>; <5:14,19,101>; ...
  - ***or***:
    - <3:5,15,19>; <4:5,100,251,431,438>; <7:17,52,121>; ...
  - ***not***:
    - <4:71,432>; <6:20,85>; ...

Positional index: Proximity queries

- Example: LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - /*k*: "within *k* words of".

- Clearly, positional indexes can be used for such queries (as opposed to biword indexes)

- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?
  - This is a little tricky to do correctly and efficiently
  - There's likely to be a problem on it!

```
PositionalIntersect(p1, p2, k)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then l ← ⟨ ⟩
           pp1 ← positions(p1)
           pp2 ← positions(p2)
           while pp1 ≠ NIL
           do while pp2 ≠ NIL
                do if |pos(pp1) − pos(pp2)| ≤ k
                      then Add(l, pos(pp2))
                      else if pos(pp2) > pos(pp1)
                              then break
                   pp2 ← next(pp2)
                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
                do Delete(l[0])
                for each ps ∈ l
                do Add(answer, ⟨docID(p1), pos(pp1), ps⟩)
                pp1 ← next(pp1)
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
             then p1 ← next(p1)
             else p2 ← next(p2)
return answer
```

Positional index: size

- You can compress position values/offsets
  - Nevertheless, a positional index expands postings storage *substantially*

- Positional index is now standardly used because of the power and usefulness of phrase and proximity queries …
  - whether used explicitly or implicitly in a ranking retrieval system.

Positional index: size

- Need an entry for each occurrence, not just once per doc

- Index size depends on average doc size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems … easily 100,000 terms

Why?

- Consider a term with frequency 0.1%

Doc size (# of terms)	Expected Postings	Expected entries in Positional postings
1000	1	1
100,000	1	100

Positional index: size (rules of thumb)

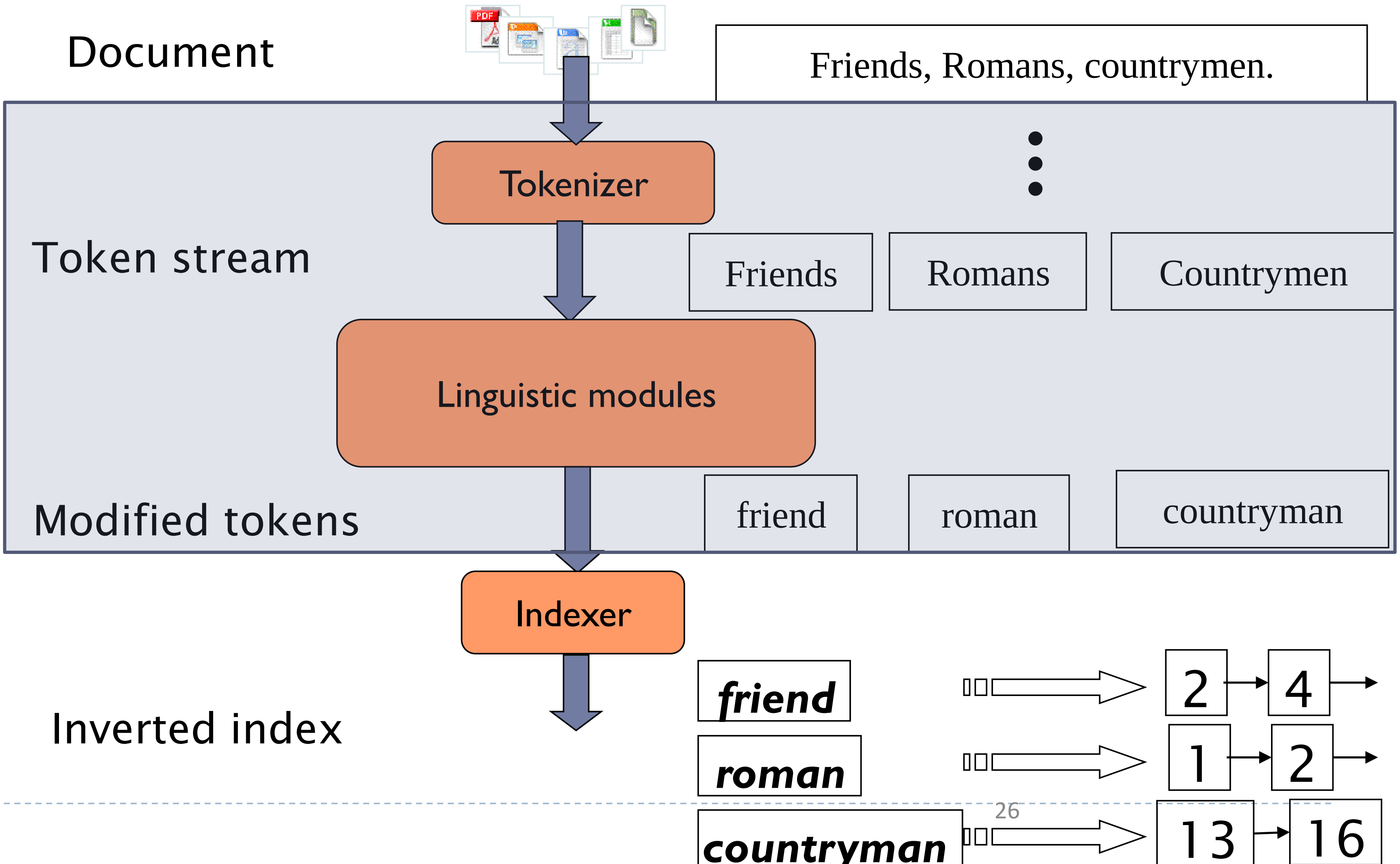
- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text
- Caveat: all of this holds for “English-like” languages

Phrase queries: Combination schemes

- Combining two approaches
  - For particular phrases (***"Michael Jordan"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like ***"The Who"***
  - Good queries to include in the phrase index are ones known to be common based on recent querying behavior.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

Text operations

Recall the basic indexing pipeline



Text operations

- Tokenization
- Stop word removal
- Normalization
  - Stemming or lemmatization
  - Equivalence classes
    - Example1: case folding
    - Example2: using thesauri (or Soundex) to find equivalence classes of synonyms and homonyms

Parsing a 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 …

Complications: Format/language

- Corpus can include docs from different languages
  - A single index may have to contain terms of several languages.

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

- What is a unit document? (indexing granularity)
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)

Tokenization

- Input: "***Friends, Romans, Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***

- Each such token is now a candidate for an index entry, after further processing

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** → ***Finland? Finlands? Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***co-education***
    - ***lower-case***
    - ***state-of-the-art***: break up hyphenated sequence.
    - It can be effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

Tokenization: Numbers

- Examples
  - ***3/12/91*** ***Mar. 12, 1991*** ***12/3/91***
  - ***55 B.C.***
  - ***B-52***
  - ***My PGP key is 324a3df234cb23e***
  - ***(800) 234-2333***
    - Often have embedded spaces
- Older IR systems may not index numbers
  - But often very useful
    - e.g., looking up error codes/stack traces on the web
- Will often index "meta-data" separately
  - Creation date, format, etc.

Tokenization: Language issues

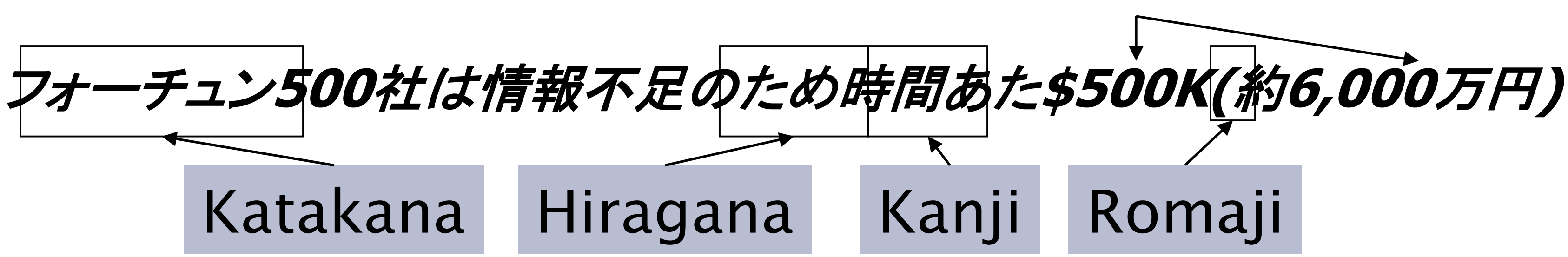
- French
  - ***L'ensemble:*** one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
    - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module
    - Can give a 15% performance boost for German

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



End-user can express query entirely in hiragana!

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

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

  'Algeria achieved its independence in 1962 after 132 years of French occupation.'

- With Unicode, the surface presentation is complex, but the stored form is straightforward

Stop words

- Stop list: exclude from dictionary the commonest words.
  - They have little semantic content: '*the*', '*a*', '*and*', '*to*', '*be*'
  - There are a lot of them: ~30% of postings for top 30 words

- But the trend is away from doing this:
  - Good compression techniques (IIR, Chapter 5)
    - the space for including stopwords in a system is very small
  - Good query optimization techniques (IIR, Chapter 7)
    - pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: "King of Denmark"
    - Various song titles, etc.: "Let it be", "To be or not to be"
    - Relational queries: "flights to London"

Normalization to terms

- Normalize words in indexed text (also query)
  - U.S.A. USA
- Term is a (normalized) word type, which is an 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***
- Crucial: Need to "normalize" indexed text as well as query terms into the same form

Normalization to terms

- 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 doc contains ***automobile***, index it under ***car-automobile*** (and/or vice-versa)

Alternative to creating equivalence classes

- Or we can expand a query
  - When the query contains ***automobile***, look under ***car*** as well

Normalization to terms

- An alternative to equivalence classing is to do asymmetric expansion

- An example of where this may be useful
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***

- Potentially more powerful, but less efficient

Normalization: Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…
- Google example: Query ***C.A.T.***
  - #1 result was for "cat" *not* Caterpillar Inc.

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?
  - Users often may not type accents even in languages that standardly have accents
    - Often best to normalize to a de-accented term
      - ***Tuebingen, Tübingen, Tubingen*** → ***Tubingen***

- For foreign names, the spelling may be unclear or there may be variant transliteration standards giving different spellings
  - Soundex forms equivalence classes of words based on phonetic heuristics

Lemmatization

- Reduce inflectional/variant forms to base form, e.g.,
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form
- It needs a complete vocabulary and morphological analysis to correctly lemmatize words

Stemming

- Reduce terms to their "roots" before indexing
  - Stemmers use language-specific rules, but they require less knowledge than a lemmatizer
- The exact stemmed form does not matter, only the equivalence classes it forms.
- Stemming: 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 equival 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: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Porter's algorithm: Typical rules

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

- Rules sensitive to the *measure* of words
  - *(m>1) EMENT* →
    - *replacement* → *replac*
    - *cement* → *cement*

Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
  - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)

- Full morphological analysis – at most modest benefits for retrieval

- Do stemming and other normalizations help?
  - English: very mixed results. Helps recall but harms precision
    - operative (dentistry) ⇒ oper
    - operational (research) ⇒ oper
    - operating (systems) ⇒ oper
  - Definitely useful for Spanish, German, Finnish, …
    - 30% performance gains for Finnish!

Lemmatization vs. Stemming

- Lemmatization produces at most very modest benefits for retrieval.
- Either form of normalization tends not to improve English information retrieval performance in aggregate
- The situation is different for languages with much more morphology (such as Spanish, German, and Finnish).
  - quite large gains from the use of stemmers

Language-specificity

- Many of the above features embody transformations that are
  - Language-specific
  - Often, application-specific
- These are "plug-in" addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Dictionary entries – first cut

ensemble.french
時間.japanese
MIT.english
mit.german
guaranteed.english
entries.english
sometimes.english
tokenization.english

These may be grouped by language (or not…).

More on this in ranking/query processing.

Resources

- IIR 2
- MIR 9.2
- Porter's stemmer:

  http://www.tartarus.org/~martin/PorterStemmer/