Introduction to Information Retrieval

(Manning, Raghavan, Schutze)

Chapter 2
The term vocabulary and postings lists

Assumptions in simple Boolean retrieval system

- We know what a document is
- We know what a term is

- Both issues can be complex in reality
- We'll discuss a little on what a document is
- But mostly on terms: how do we define and process the vocabulary of terms of a collection?

Parsing a document

- Before we start worrying about terms … need to know 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,
  - But often done heuristically

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
  - French email with a German pdf attachment.

What is a unit of document?

- A file?
  - Traditional Unix stores a sequence of emails in one file, but you might want to regard each email as a separate document
- An email with 5 attachments?

- **Indexing granularity**, e.g. a collection of books
  - Each book as a document?
  - Each chapter? Each paragraph? Each sentence?
- Precision recall tradeoff
  - Small unit: good precision, poor recall
  - Big unit: good recall, poor precision

Determining the vocabulary of terms

Definitions

- Token: an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing
- Type: the class of all tokens containing the same character sequence
- **Term**: a (perhaps normalized) type that is included in the IR system's dictionary

- E.g., to sleep perchance to dream
  - 5 tokens
  - 4 types
  - 3 terms (if to is omitted from the index as stop word)

Tokenization

- Input: "***Friends, Romans, Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

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

Numbers

- ***3/12/91*** ***Mar. 12, 1991***
- ***55 B.C.***
- ***B-52***
- ***My PGP key is 324a3df234cb23e***
- ***(800) 234-2333***
  - Often have embedded spaces
  - Will often index “meta-data” separately
    - Creation date, format, etc.

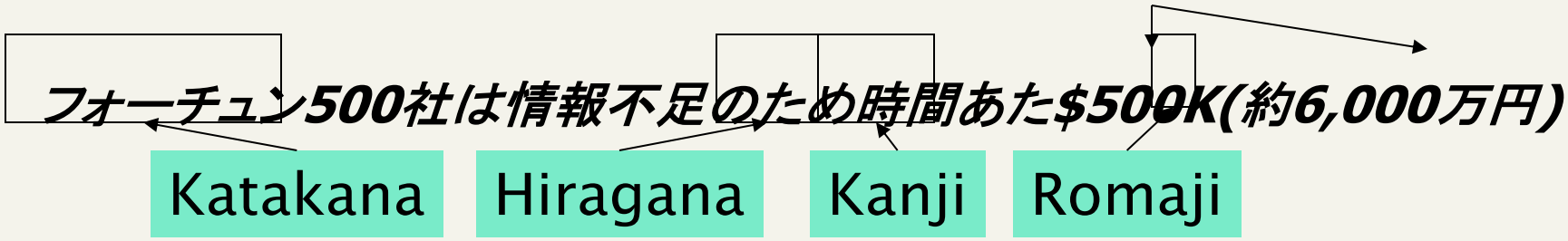
Tokenization: language issues

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

Tokenization: language issues

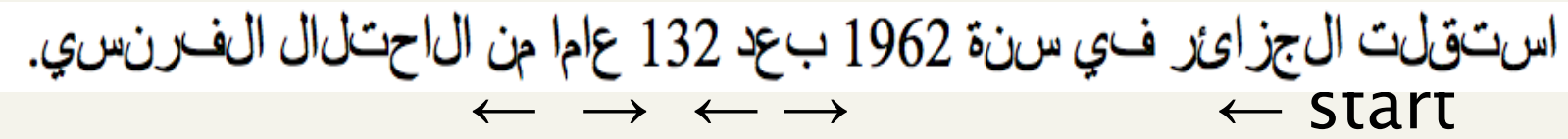
- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



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



- ‘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

Common terms: stop words

- **Stop words** = extreme common words which would appear of little value in helping select document in matching a user need
  - *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*
  - There are a lot of them: ~30% of postings for top 30 words

- Stop word elimination used to be standard in older IR systems
  - Size of stop list: 200-300; 7-12

Current trend

- The trend is away from doing this:
  - Good compression techniques (lecture 5) 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 titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”

- Nowadays search engines generally do not eliminate stop words

Normalization

- Need to "normalize" terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- We most commonly implicitly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows*** (no expansion)
- Two approaches for the (more powerful) alternative
  - Index unnormalized tokens and expand query terms
  - Expand during index construction
  - Both less efficient than equivalent claassing

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…

More equivalent classing

- Soundex: phonetic equivalence
  - Traditional class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - Invented for the US Census
  - E.g., ***chebyshev*** → ***tchebycheff***

- Thesauri: semantic equivalence
  - Hand-constructed equivalence classes
    - e.g., ***car*** = ***automobile***
    - ***color*** = ***colour***

Lemmatization

- Reduce inflectional/variant forms to base form
  - *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 headword form

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 equival to compress

Porter's algorithm (1980)

- Most common algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- 5 phases of reductions
  - phases applied sequentially
  - With each phase, there are various conventions of selecting rules
  - E.g., Sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*
- http://www.tartarus.org/~martin/PorterStemmer/

Typical rules in Porter

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

- Weight of word sensitive rules: loosely check the number of syllables to see whether a word is long enough …
- *(m>1) EMENT* →
  - *replacement* → *replac*
  - *cement* → *cement*

Other stemmers exist

- Older Lovins stemmer (1968)
- Newer Paice/Husk stemmer (1990)

- Rather than stemmer, we can use lemmatizer: a NLP tool that does full morphological analysis to accurately identify the lemma for each word
  - at most modest benefits for retrieval

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

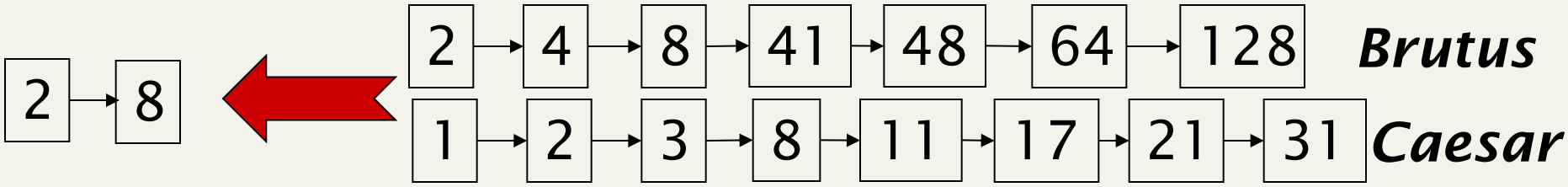
Discussion

- Do stemming and other normalizations help?
  - English: very mixed results
  - Helps recall but harms precision
    - operate operating operates operation operative operatives operational => oper (by Porter)
  - Definitely useful for Spanish, German, Finnish, …

Faster postings merges: Skip pointers/Skip lists

Recall basic merge

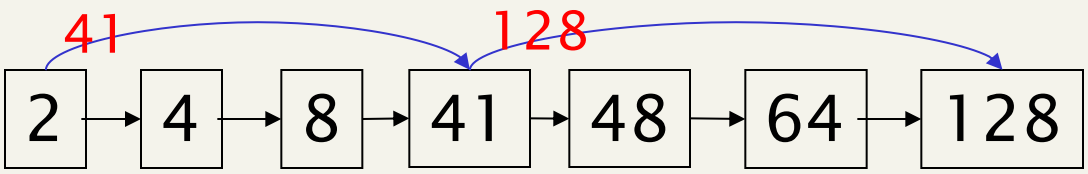
- Walk through the two postings simultaneously, in time linear in the total number of postings entries

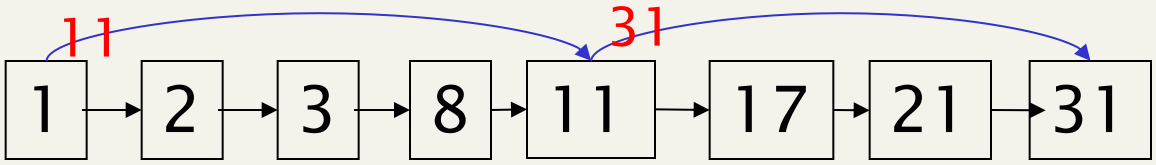


If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

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

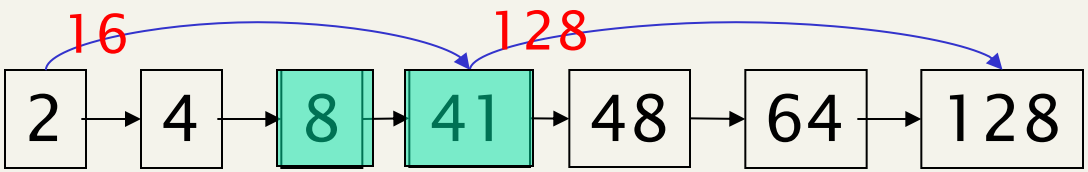
Augment postings with skip pointers (at indexing time)

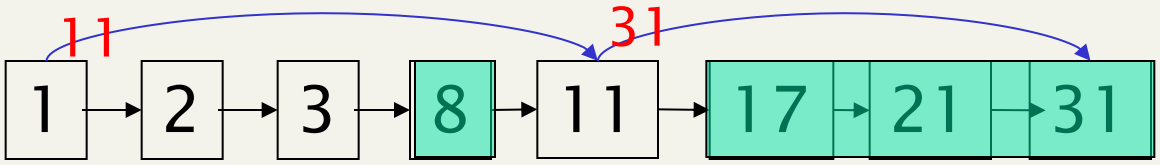




- Why?
  - To skip postings that will not figure in the search results
  - This makes intersecting postings lists more efficient
  - Some postings contains several million entries, so efficiency can be an issue even if basic intersection is linear
- How?
- Where do we place skip pointers?

Query processing with skip pointers





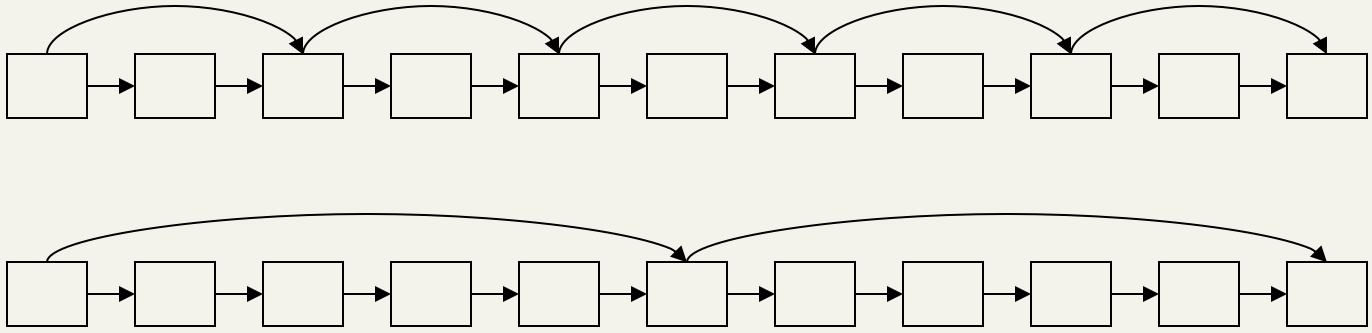
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

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.



Placing skips

- Simple heuristic: for postings 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 if P keeps changing because of updates.

- This definitely used to help
- With modern hardware (fast CPUs) it may not
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

Phrase queries and positional indexes

Phrase queries

- Want to be able to answer queries such as "***stanford university***" – as a phrase
- *"The inventor Stanford Ovshinsky never went to univerisity"* is not a match.
  - The concept of phrase queries has proven easily understood by users
  - 10% of web queries are phrase queries

- For this, it no longer suffices to store only <*term* : *docs*> entries

any ideas?

A first attempt: Biword 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 phrase.

Can have false positives!

Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POST).
- 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.
  - Each such extended biword is now made a term in the dictionary.
- Example: ***catcher in the rye***

  N X X N
- Query processing: parse it into N's and X's
  - Segment query into enhanced biwords
  - Look up index

Issues for biword indexes

- Why biword indexes rarely used?
- False positives, as noted before
- Index blowup due to bigger dictionary

- For extended biword index, parsing longer queries into conjunctions:
  - E.g., the query ***tangerine trees and marmalade skies*** is parsed into
  - ***tangerine trees*** *AND* ***trees and marmalade*** *AND* ***marmalade skies***

Solution 2: Positional indexes

- In the postings, store, for each ***term***, entries of the form:

  <***term***, number of docs containing ***term***;

  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  etc.>

Positional index example

be: 178239:
<*1*, 2: <17, 25>;
4, 5: <17, 191, 291, 430, 434>;
5, 3: <14, 19, 101>; …>

- The word be has a document frequency 178239, and occurs 2 times in document 1 at positions 17 and 25

Processing a query phrase

- We use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality
- Returns the actual matching positions, not just a list of documents
- Very inefficient for frequent words, especially stop words

Example query: to be or not to be

- 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*:17,25; *4*:17,191,291,430,434; *5*:14,19,101; ...

Proximity queries: same idea

- employment /3 place
- Find all document that contain employment and place within 3 words of each other
- Employment agencies that place healthcare workers are seeing growth
  - hit
- Employment agencies that help place healthcare workers are seeing growth
  - not a hit
- Clearly, positional indexes can be used for such queries; biword indexes cannot.

Positional index size

- You can compress position values/offsets: covered in chapter 5
- Nevertheless, a positional index expands postings storage *substantially*
  - Need an entry for each occurrence, not just once per document
  - Compare to biword: "Index blowup due to bigger dictionary"
- Nevertheless, a **positional index is now standardly used** because of the power and usefulness of **phrase and proximity** queries

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: the above holds for English-like languages.

Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (***"Michael Jackson", "Britney Spears"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like ***"The Who"***
- 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