Information Retrieval

Lecture 2

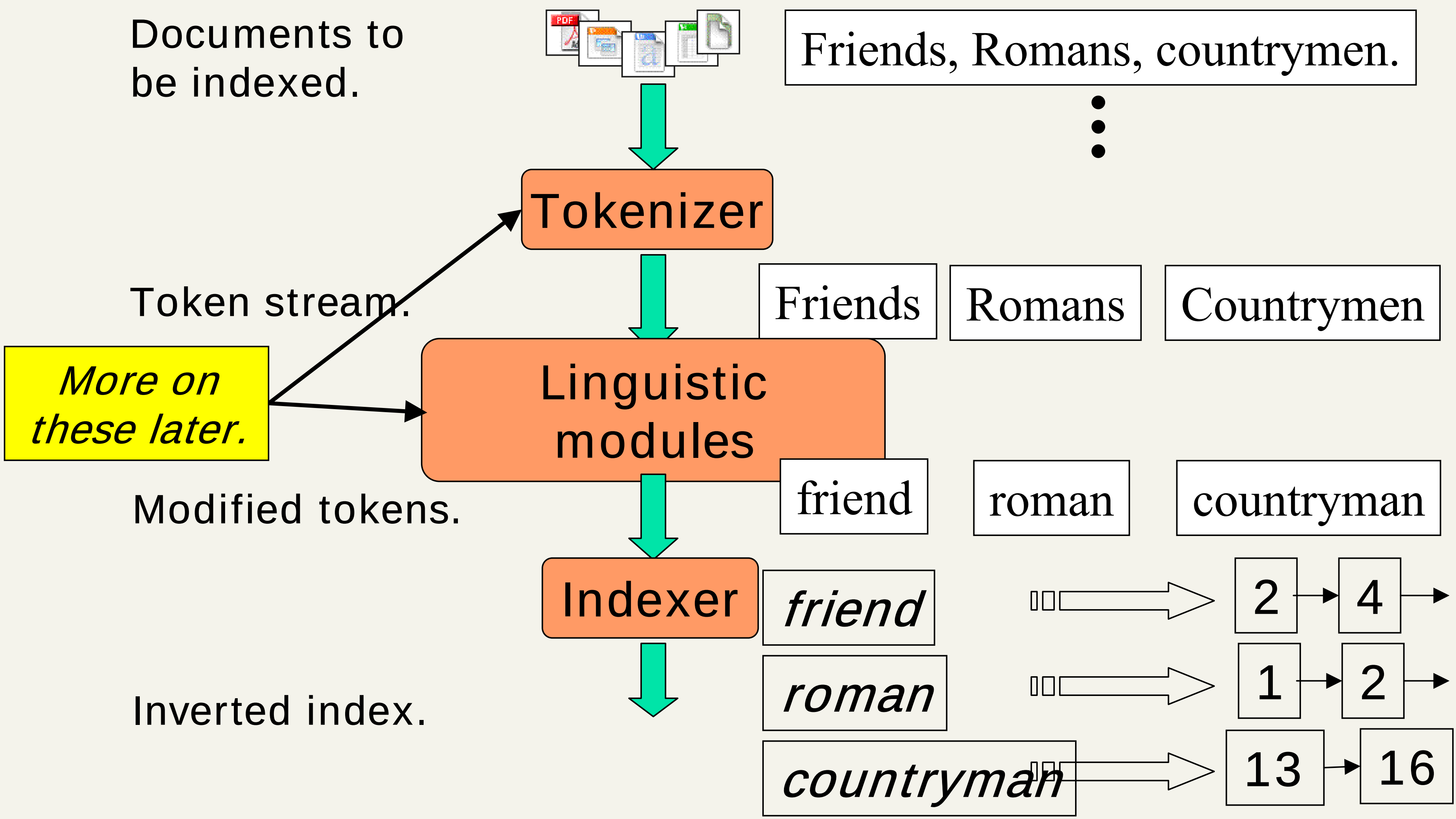
Recap of the previous lecture

- Basic inverted indexes:
  - Structure – Dictionary and Postings
  - Key steps in construction – sorting
- Boolean query processing
  - Simple optimization
  - Linear time merging
- Overview of course topics

Plan for this lecture

- Finish basic indexing
  - Tokenization
  - What terms do we put in the index?
- Query processing – more tricks
- Proximity/phrase queries

Recall basic indexing pipeline



Tokenization

Tokenization

- Input: "*Friends, Romans and 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?

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 there are complications …

Format/language stripping

- 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 Portuguese pdf attachment.
- <u>What is a unit document</u>?
  - An email?
  - With attachments?
  - An email with a zip containing documents?

Tokenization

- Issues in tokenization:
  - *Finland's capital* → *Finland? Finlands? Finland's*?
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
  - *San Francisco*: one token or two? How do you decide it is one token?

Language issues

- Accents: *résumé* vs. *resume*.
- *L'ensemble* → one token or two?
  - *L* ? *L'* ? *Le* ?
- How are your users like to write their queries for these words?

Tokenization: language issues

- Chinese and Japanese have no spaces between words:
  - 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**$500K(**約**6,000**万円**)***

Katakana Hiragana Kanji "Romaji"

End-user can express query entirely in (say) Hiragana!

Normalization

- In “right-to-left languages” like Hebrew and Arabic: you can have “left-to-right” text interspersed (e.g., for dollar amounts).
- Need to “normalize” indexed text as well as query terms into the same form

7月30日 vs. 7/30

- Character-level alphabet detection and conversion
  - Tokenization not separable from this.
  - Sometimes ambiguous:

Morgen will ich in MIT …

Is this German “mit”?

What terms do we index?

Cooper's concordance of Wordsworth was published in 1911. The applications of full-text retrieval are legion: they include résumé scanning, litigation support and searching published journals on-line.

Punctuation

- *Ne'er*: use language-specific, handcrafted "locale" to normalize.
  - Which language?
  - Most common: detect/apply language at a pre-determined granularity: doc/paragraph.
- *State-of-the-art*: break up hyphenated sequence. Phrase index?
- *U.S.A.* vs. *USA* - use locale.
- *a.out*

Numbers

- *3/12/91*
- *Mar. 12, 1991*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *100.2.86.144*
  - Generally, don't index as text.
  - Will often index "meta-data" separately
    - Creation date, format, etc.

Case folding

- Reduce all letters to lower case
  - exception: upper case (in mid-sentence?)
    - *e.g., General Motors*
    - *Fed* vs. *fed*
    - *SAIL* vs. *sail*

Thesauri and soundex

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., *car* = *automobile*
    - *your* ➔ *you're*
- Index such equivalences
  - When the document contains *automobile*, index it under *car* as well (usually, also vice-versa)
- Or expand query?
  - When the query contains *automobile*, look under *car* as well
- More on this later ...

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*

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. More on this in query processing.

Stemming

- Reduce terms to their “roots” before indexing
  - 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 compres and compres are both accept as equival to compres.

Porter's algorithm

- Commonest algorithm for stemming English
- 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

- $sses \rightarrow ss$
- $ies \rightarrow i$
- $ational \rightarrow ate$
- $tional \rightarrow tion$

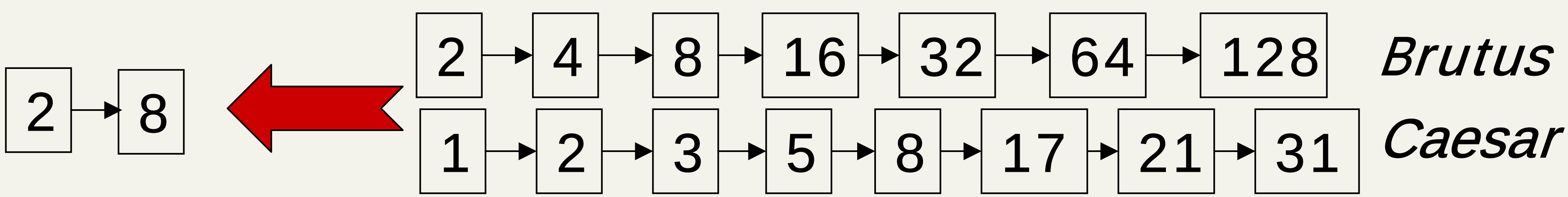
Other stemmers

- 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)
- Motivated by Linguistics as well as IR
- Full morphological analysis - modest benefits for retrieval

Faster postings merges: Skip pointers

Recall basic merge

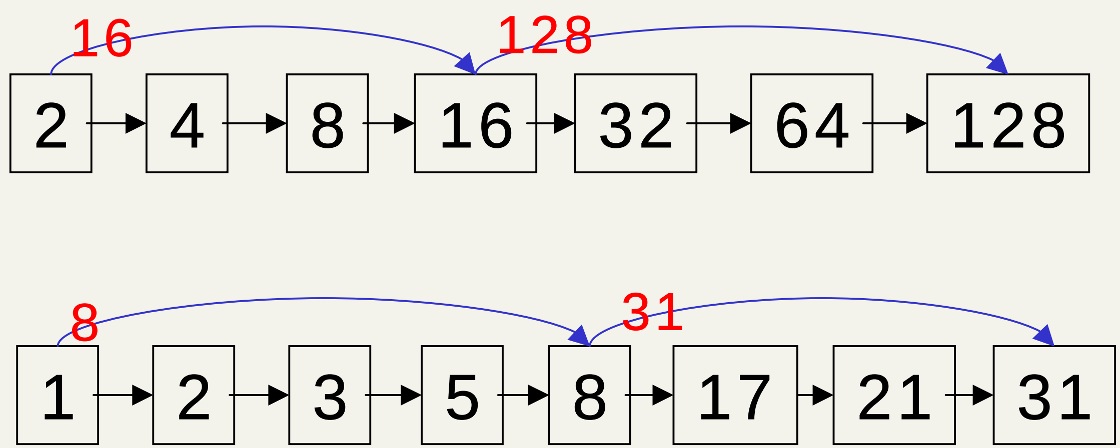
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

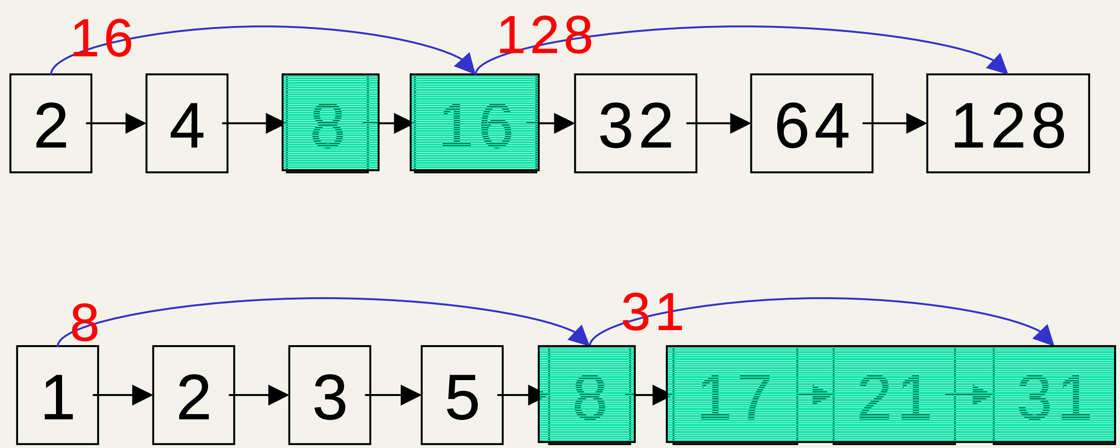
Can we do better?
Yes, if index isn't changing too fast.

Augment postings with skip pointers (at indexing time)



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

Query processing with skip pointers



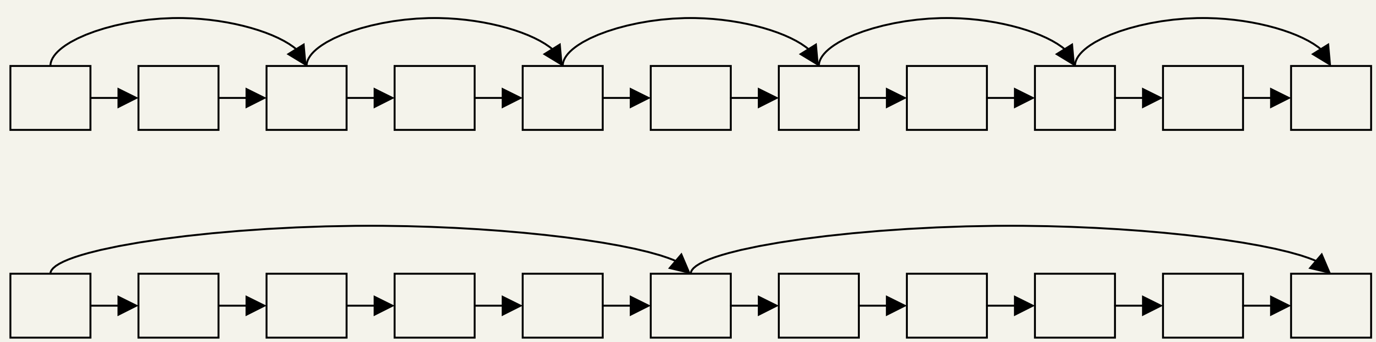
Suppose we've stepped through the lists until we process 8 on each list.

When we get to 16 on the top list, we see that its successor is 32.

But the skip successor of 8 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 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.

Phrase queries

Phrase queries

- Want to answer queries such as *stanford university* – as a phrase
- Thus the sentence “I went to university at Stanford” is not a match.
- No longer suffices to store only

< *term* : *docs*> entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans and Countrymen” would generate the biwords
  - *friends romans*
  - *romans and*
  - *and countrymen*
- Each of these 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

<u>Unlike</u> wild-cards, though, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Think about the difference.

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

- Given a query, parse it into N’s and X’s
  - Segment query into enhanced biwords
  - Look up index
- Issues
  - 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*

Other issues

- False positives, as noted before
- Index blowup due to bigger dictionary

Positional indexes

- Store, for each *term*, entries of the form:

  <number of docs containing *term*;

  doc1: position1, position2 ... ;

  doc2: position1, position2 ... ;

  etc.>

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***”?

- Can compress position values/offsets as we did with docs in the last lecture
- Nevertheless, this expands postings storage *substantially*

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; ...
- Same general method for proximity searches

Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  Here, /k means "within k words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?

Positional index size

- Can compress position values/offsets as we did with docs in the last lecture
- Nevertheless, this expands postings storage *substantially*

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size

Why?

  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems … easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- Positional index size factor of 2-4 over non-positional index
- Positional index size 35-50% of volume of original text
- Caveat: all of this holds for “English-like” languages

Resources for today's lecture

- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer: http//www.sims.berkeley.edu/~hearst/irbook/porter.html
- H.E. Williams, J. Zobel, and D. Bahle, "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems. http://www.seg.rmit.edu.au/research/research.php?author=4