Introduction to

Information Retrieval

Lecture 3

Laura Dietz

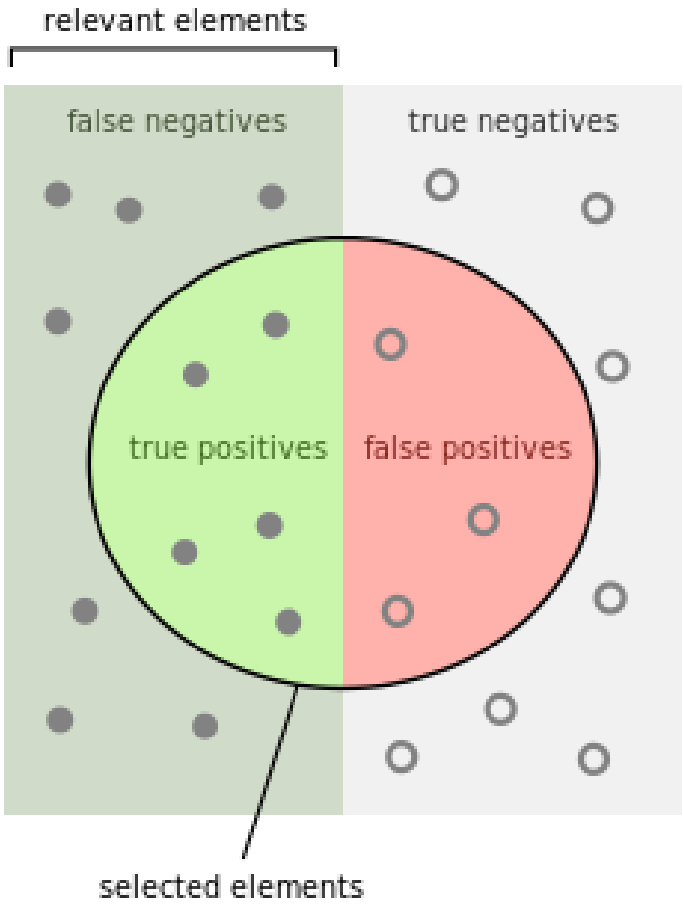
Information Retrieval and Web Search

Chapter 2: term vocabulary and postings lists

(based on slides from: Pandu Nayak and Prabhakar Raghavan)

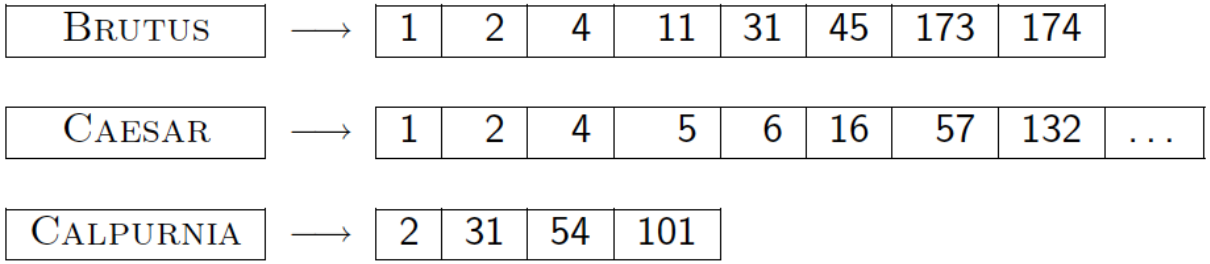
Recap of lecture 02

- Eval Metrics: Precision, Recall, F1, Accuracy
  - Other measures: R-Precision, NDCG, Mean-Avg Precision (MAP)
- Test collections
- Gold standard
- Inter-annotator agreement
- Tasks: Vertical Search, Entity Search, Recommender Systems



Recap of lecture 01

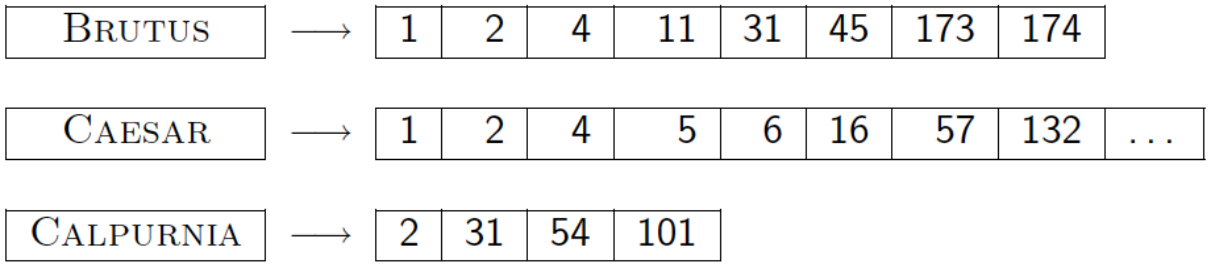
- Basic inverted indexes:
  - Structure: Dictionary and Postings



  - Key step in construction: Sorting
- Boolean query processing
  - Intersection by linear time “merging”
  - Simple optimizations

Explain to your neighbor:

- What is an inverted index and a postings list?
- How to use an inverted index to answer:
  - **(Brutus** AND **Caesar)** AND NOT **Calpurnia**

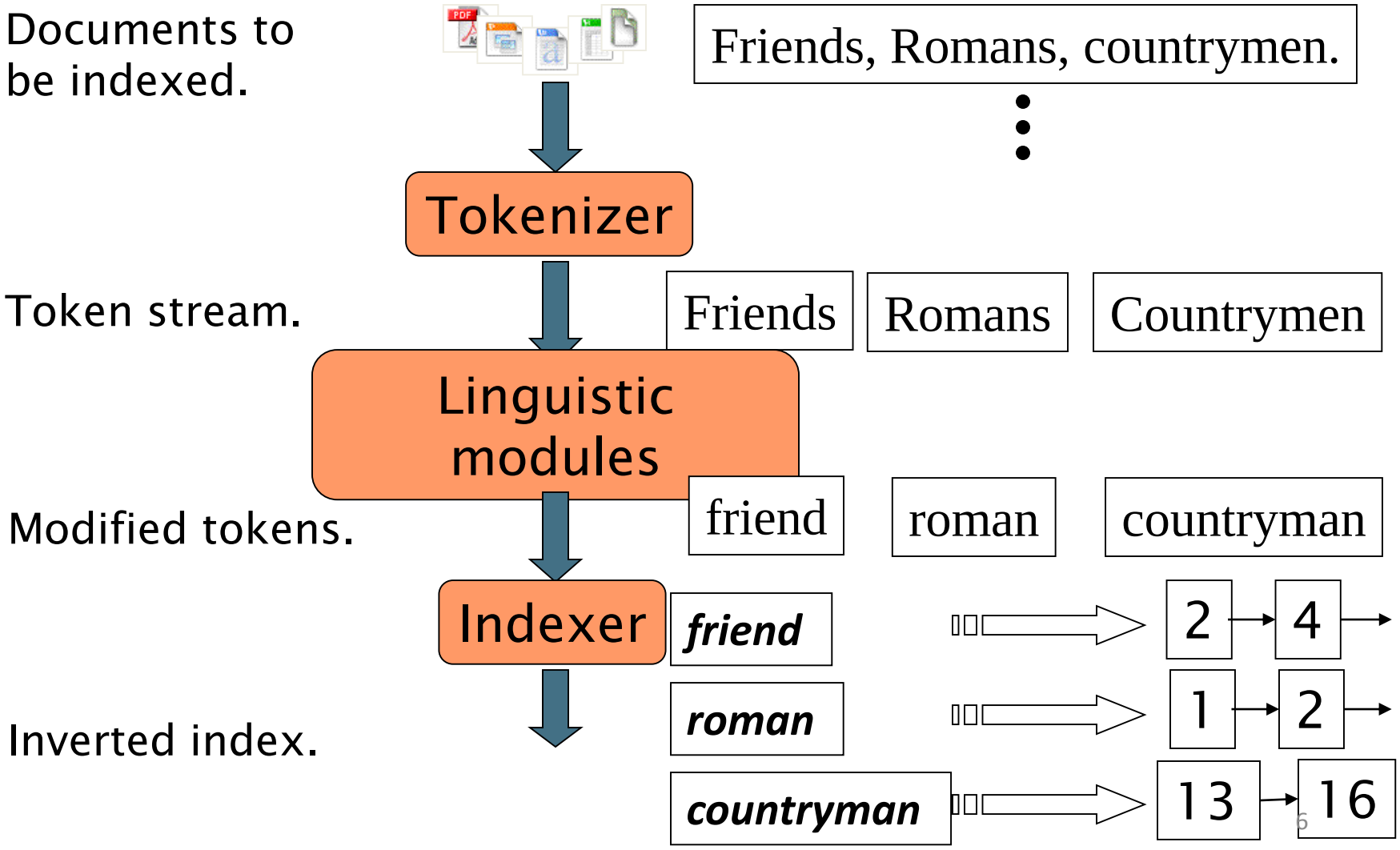


Plan for this lecture

Elaborate basic indexing

- Preprocessing to form the term vocabulary
  - Formats
  - Tokenization
  - Language Issues
  - Stemming: What *terms* do we put in the index?
- Postings
  - Faster merges: skip lists
  - Proximity: Positional index and phrase queries

Recall the basic indexing pipeline



Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- Wrappers: Home page, News, WikiHow, Twitter
- What language is it in?
- What character set is in use?

Each of these is a classification problem

But often determined 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 document?
  - 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)

TOKENS AND TERMS

Some definitions

- **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.
- **Word Type** – The same as a term in most cases: an equivalence class of tokens.

Tokenization

- Input: "***Friends, Romans, Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is a sequence of characters in a document
- 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*** → ***Finland? Finlands? Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
    - 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?

Numbers

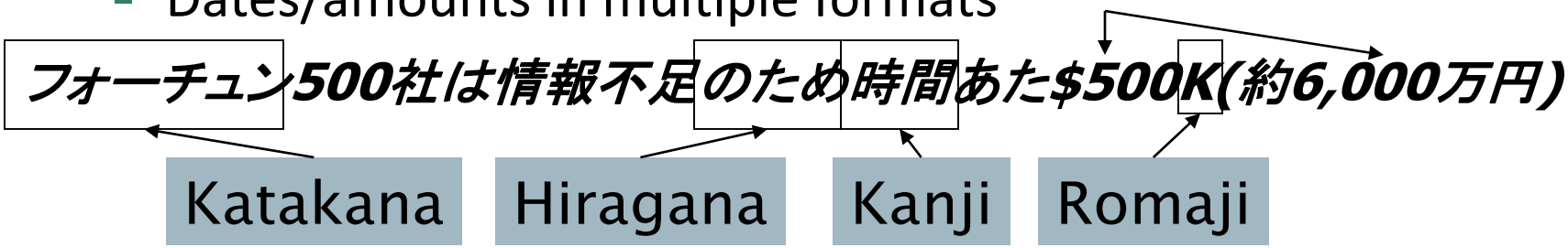
- ***3/12/91 Mar. 12, 1991 12.3.91***
- ***55 B.C.***
- ***B-52***
- ***PGP key 324a3df234cb23e***
- ***+49 621 4723-537***
  - Often spaces and symbols have meaning
  - Older IR systems may not index numbers
    - But often very useful: think about things like looking up error codes/stacktraces on the web
    - (One answer is using n-grams: Lecture 3)
  - 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***
      - Internationalization!
- 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



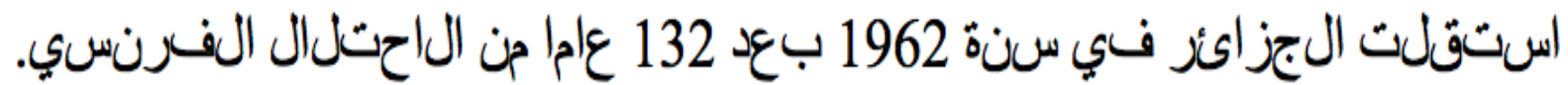
End-user can express query entirely in hiragana!

Tokenization: language issues

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

Tokenization: language issues

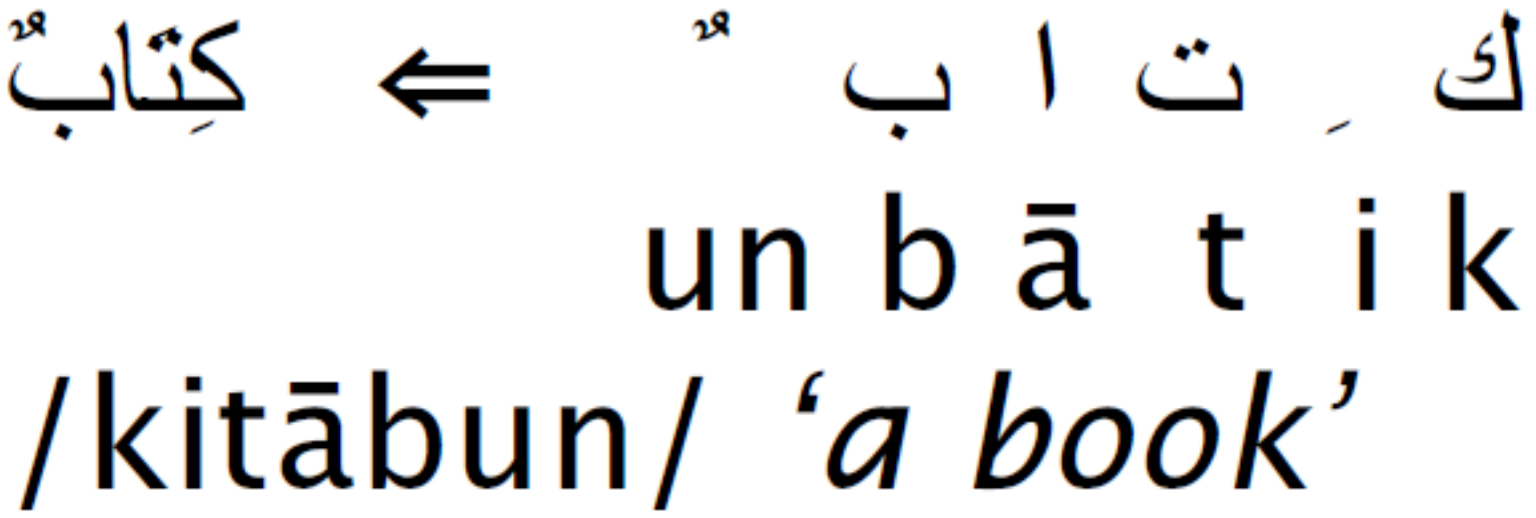
- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures



← → ← → ← start

- '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

Tokenization: language issues



- Vowels (and other grammatical markers) appear as diacritics above and below the consonants. Day-to-day text is unvocalized or only partially vocalized.

Arabizi: Arabic Chat Language

Moroccan	كيف داير في القراية؟
Arabizi	kayf dayer fee l9raya?
English	How are you doing with your studies?

Letters	Arabic chat alphabet	IPA
ء أ ؤ إ ئ آ	2	ʔ
ا	a e è [1]	æ(ː) a(ː) ɑ(ː) ɛ(ː) ɐ
ب	b p	b p
ت	t	t t̪ t͡s
ث	s th	s θ
ج	j dj g	ʒ d͡ʒ ɟ ɟʝ ɡ
ح	7	ħ ʜ
خ	kh 7' 5	x χ

Stop words

- With a stop list, you exclude the most common words from the dictionary entirely. Intuition:
  - 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 goes away from stopping:
  - Good compression techniques (chapter 5) means only little space is required for stopwords.
  - Good query optimization techniques (chapter 7) mean you pay little at query time for including stop words.
  - You need stopwords 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

- We need to "normalize" words in indexed text as well as query words into the same form
  - We want to match ***U.S.A.*** and ***USA***
- Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly equivalence classes of words defined by, e.g.,
  - deleting periods to form a term
    - ***U.S.A., USA => USA***
  - deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory => antidiscriminatory***

Normalization: other languages

- Accents: e.g., French ***résumé*** vs. ***resume*.**
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent
- Most important criterion:
  - How do your users like to write their queries for these words?

- Even in languages that have accents, users often may not type them
  - Often best to normalize to a normal form
    - ***Tuebingen, Tübingen, Tubingen => Tubingen or Tuebingen***

Normalization: other languages

- Normalization of things like date forms
  - ***30.7. vs. 7/30***
  - ***Japanese use of kana vs. Chinese characters***

- Tokenization and normalization may depend on the language and thereby is intertwined with language detection

Morgen will ich in MIT …

Is this German "mit"?

- Crucial: Need to "normalize" indexed text as well as query terms into the same form

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 in queries.
- Google example:
  - Query ***C.A.T.*** (intention: Caterpillar Inc.)
  - #1 result was an animal

Normalization to terms

- An alternative to normalized forms is asymmetric expansion of the query
- 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 (needs to look up and merge more posting lists)

Thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalent normalized terms
    - When the document contains ***automobile***, index it under ***car*** and ***automobile*** (and vice-versa)
  - Or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well
- What about spelling mistakes?
  - One approach is soundex, which forms equivalence classes of words based on phonetic heuristics
- More in chapters 3 and 9

Lemmatization

- Reduce conjugated word forms to base form, e.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are of different colors* →
  the boy car be of different color
- Lemmatization implies doing "proper" reduction to *dictionary headword form* (the **lemma**)
- Syntax: Inflectional morphology (cutting → cut)
- Change semantics: Derivational morphology (destruction → destroy)

Stemming

- Reduce terms to their word stem (root) before indexing
- "Stemming" removes suffixes with heuristics
  - 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

- Most common algorithm for English stemming
  - Results suggest it is at least as good as other stemming options
- Uses conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - If multiple rules apply, chose one with the longest suffix
- Details about Porter Algorithm:
  - http://snowball.tartarus.org/algorithms/porter/stemmer.html

Typical rules in Porter

- *Rules*
  - *ing* →
  - *ly* →
  - *sses* → *ss*
  - *ational* → *ate*
  - *tional* → *tion*
- Rules sensitive to the *measure* of words
  - Consider sequences of consonants or vowels [C](VC)^m[V].
- *(m>1) ement* →
  - *replacement* → *replac*
  - *cement* → *cement*

Issues: Over-/Understemming

- Overstemming:
  Different words map to the same stem
  - universal, university, and universe → univers
- May help recall but harms precision

- Understemming:
  Very similar words mapped to different stems
  - alumnus → alumnu
  - alumni → alumni
  - alumna / alumnae → alumna

Other stemmers

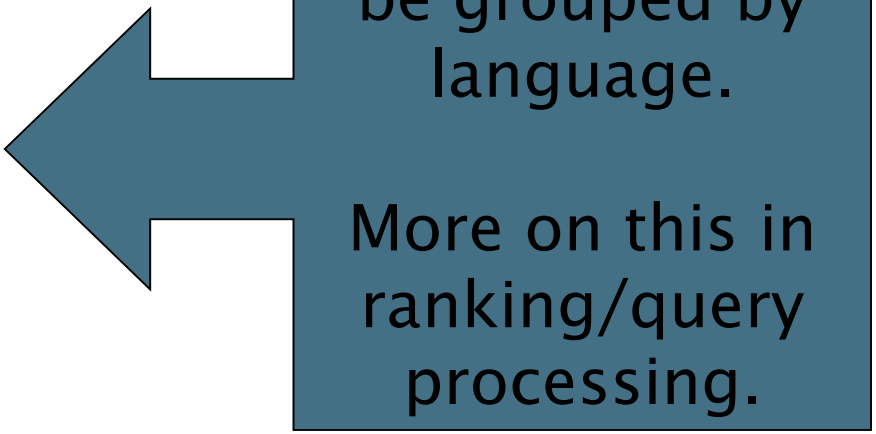
- Lovins stemmer
  - Single-pass, longest suffix removal (about 250 rules)
  - http://snowball.tartarus.org/algorithms/lovins/stemmer.html
- Krovetz Stemmer
  - Checks whether result of rule is contained in dictionary
  - http://lexicalresearch.com/sigir-1993.pdf
- Full morphological analysis – at most modest benefits for retrieval

Language-specificity

- Many of the above features embody transformations that are
  - Language-specific and
  - (Often) application-specific
- Benefits differ between languages
  - Good for English and German
  - 30% performance gains for Finnish!
  - Only 2% for Hebrew
- Stemmers are “plug-ins” to the indexing process
- Both open source and commercial stemmer plug-ins are available

Mark Language in Dictionary

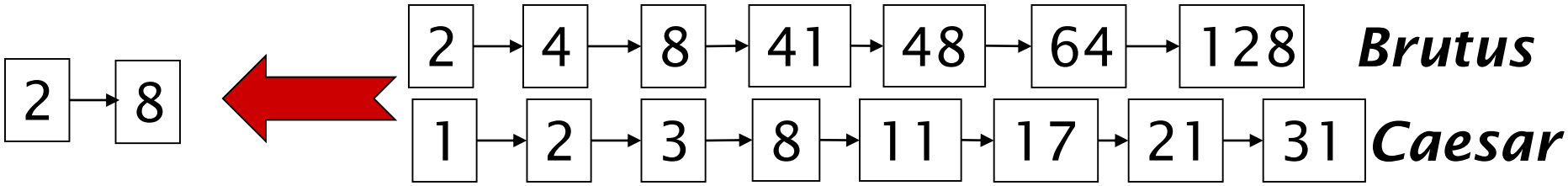
ensemble.french
時間.japanese
MIT.english
mit.german
guaranteed.english
entries.english
sometimes.english
tokenization.english



FASTER POSTINGS MERGES: SKIP POINTERS/SKIP LISTS

Recall basic merge

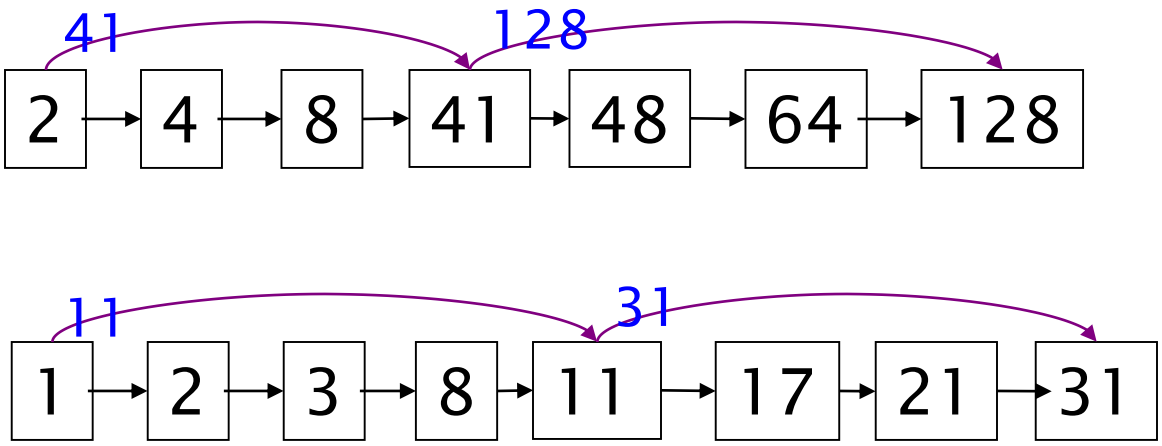
- Walk through the two postings simultaneously, in time linear in the total number of postings entries
- Query: Brutus AND Caesar



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

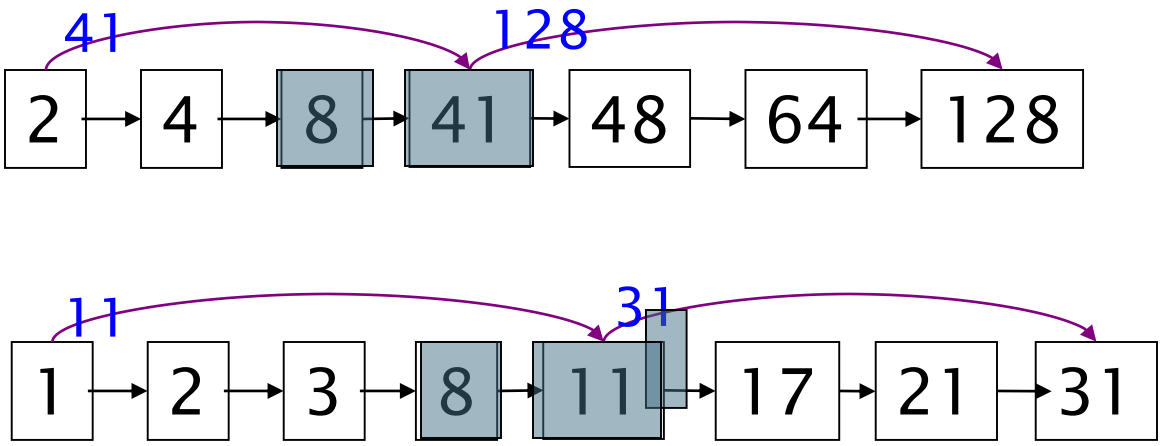
Can we do better?
Yes (for read-only indexes).

Augment postings with skip pointers (at indexing time)



- Why?
  - To skip batches of postings that lead to empty 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. We match it and advance.

We then have **41** and **11**. **11** is smaller.

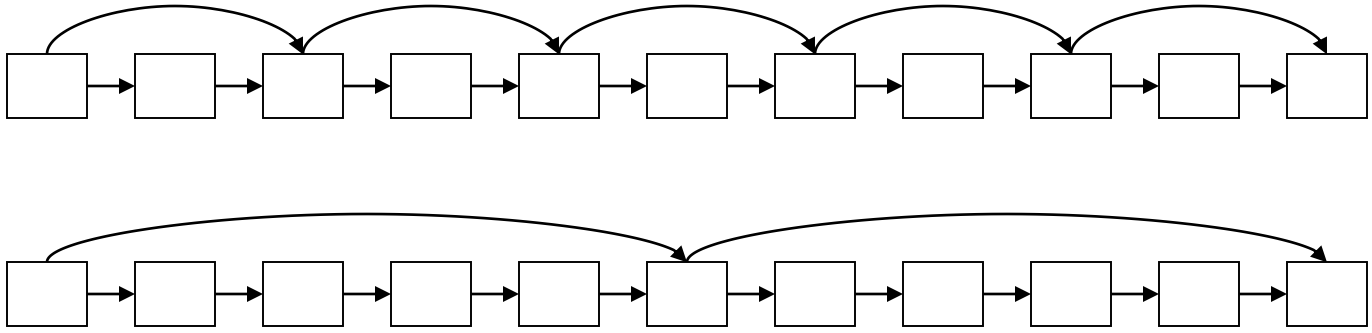
Skip successor of **11** is **31**, which is still less than 41. So we can skip ahead to 31 without missing postings.

Intersecting with skip pointers

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip.
    But lots of comparisons to skip pointers.
    Also: More data to store.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

- Simple heuristic: for postings of length L, use $\sqrt{L}$ evenly-spaced skip pointers.
- Easy to implement if the index is read-only.
- Requires maintenance when updating index, as L keeps changing.

- This ignores the distribution of query terms.
- Alternative: random insertion of skips.

PHRASE QUERIES AND POSITIONAL INDEXES

Phrase queries

- Want to be able to answer queries such as "***stanford university***" – as a phrase
- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works well in practice
  - Many queries are *implicit phrase queries*
- For this, it no longer suffices to only store

  term -> posting lists

A first attempt: Biword indexes

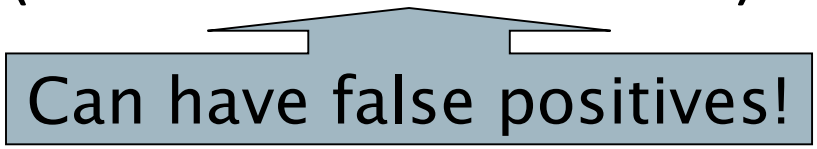
- (Additionally) index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Turn query into biwords for lookup and merging.

Query processing of phrase queries

- Merge biwords the same way as merging words.
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

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

- This may also match documents that does not contain the exact phrase (check with doc content).



- Phrase query might not be what the user intended (**university** NEAR **palo**)

Extended biwords

- Idea: only use bi-nouns (skipping prepositions, etc.)
- Parse the indexed text and perform part-of-speech-tagging
- For example, divide the terms into Nouns (N) and articles/prepositions (X).
- Call any string of terms of the form NX*N an extended biword.
  - Each such extended biword now becomes a term in the dictionary.
- Example: ***catcher in the rye***

  N X X N
- Query processing: parse it into N's and X's
  - Split query into extended biwords
  - Look up in index: ***catcher rye***

Issues for biword indexes

- Can have false positives (check with doc content)
- How about single term queries?
- Index blowup due to bigger dictionary
  - Infeasible already for tri-words,
  - Very large for biwords

- Alternative: only index biwords for very common phrases, example:
  - **Stanford university** but not **university palo**
- Can be part of a hybrid strategy with unigrams/words

Solution 2: Positional indexes

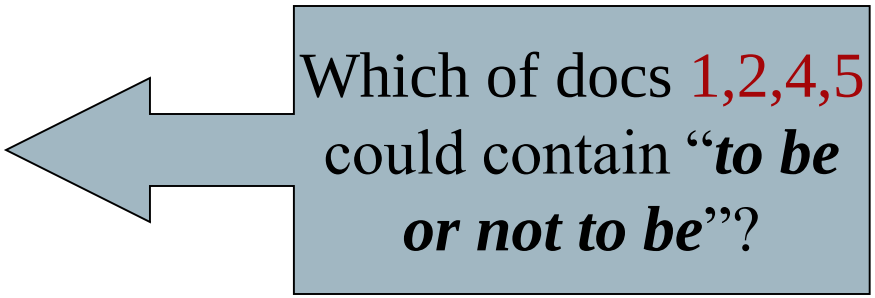
- In the postings, store for each ***term*** the position(s) in which tokens of it appear:

<***term***: number of docs containing ***term***;
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, …>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>



- For phrase queries, we use a merge algorithm recursively at the document level
- Needs adaptation to phrase and proximity

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; ...
  - ...
- General method for proximity searches

Proximity queries

- STATUTE /3 FEDERAL /2 TORT
  - Again, 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*?

"Proximity" intersection

```
POSITIONALINTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 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(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
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
```

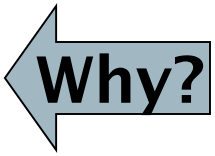
Positional index size

- You can compress position values/offsets: chapter 5
- Nevertheless, a positional index expands postings storage *substantially*
  - *Before: store term frequency = 1 integer*
  - *Now: store position per term occurrence = n integers*

- Benefits outweigh the costs:
- Phrase and proximity queries are very useful for both explicit and implicit phrases
  - **“Stanford university” “palo alto”**

Positional index size

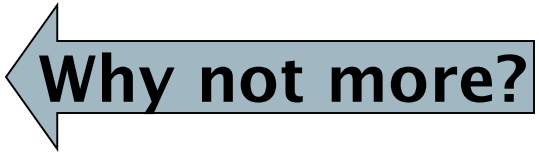
- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - Books, articles, documentation easily 100,000 terms
- Consider a term with frequency 0.1%



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

Rules of thumb

- 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



Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (***"Michael Jackson", "Stanford University"***) it is inefficient to combine terms via merging positional postings lists
    - Even more so for phrases like ***"The Who"***
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - Indexing common phrases
  - "next-word" index

Resources for today's lecture

- IIR Chapter 2
- Porter's stemmer:
  http://www.tartarus.org/~martin/PorterStemmer/
- Skip Lists theory: Pugh (1990)
  - Multilevel skip lists give same O(log *n*) efficiency as trees
- H.E. Williams, J. Zobel, and D. Bahle. 2004. "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.
  http://dl.acm.org/citation.cfm?id=1028102

Test your knowledge

- What are the issues with multiple languages?
- What are effective term normalization methods?
  - What are their respective issues?
- What are the pros/cons of skip lists?
  - Does it matter whether index is stored in memory or on conventional hard drives?
- What indexing techniques support phrase and proximity queries?
  - When to use which?