Introduction to
Information Retrieval

CS276: Information Retrieval and Web Search

Pandu Nayak and Prabhakar Raghavan

Lecture 2: The term vocabulary and postings lists

Recap of the previous lecture

- Basic inverted indexes:
  - Structure: Dictionary and Postings

BRUTUS	$\longrightarrow$	1	2	4	11	31	45	173	174

CAESAR	$\longrightarrow$	1	2	4	5	6	16	57	132	. . .

CALPURNIA	$\longrightarrow$	2	31	54	101

  - Key step in construction: Sorting
- Boolean query processing
  - Intersection by linear time "merging"
  - Simple optimizations
- Overview of course topics

Plan for this lecture

Elaborate basic indexing

- Preprocessing to form the term vocabulary

  - Documents

  - Tokenization

  - What *terms* do we put in the index?

- Postings

  - Faster merges: skip lists

  - Positional postings and phrase queries

Recall the basic indexing pipeline

Documents to
be indexed.

Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends	Romans	Countrymen

Linguistic modules

Modified tokens.

friend	roman	countryman

Indexer

Inverted index.

friend	⟹	2 → 4 →
roman	⟹	1 → 2
countryman	⟹	13 → 16

4
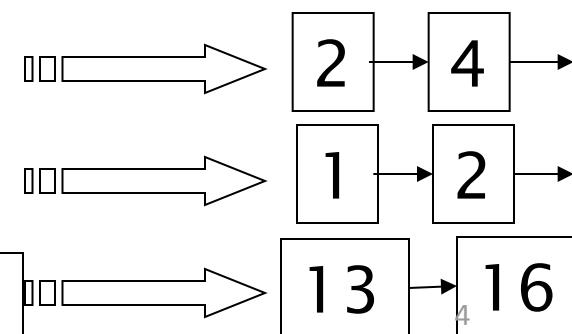
Parsing a document

- 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

- 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.
- <u>What is a unit document</u>?
  - 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

Tokenization

- <u>Input</u>: "***Friends, Romans, Countrymen***"
- <u>Output</u>: 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 <u>further processing</u>
  - 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*

- *My PGP key is 324a3df234cb23e*

- *(800) 234-2333*

  - Often have embedded spaces

  - 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*
      - Until at least 2003, it didn't on Google
        - 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

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 |
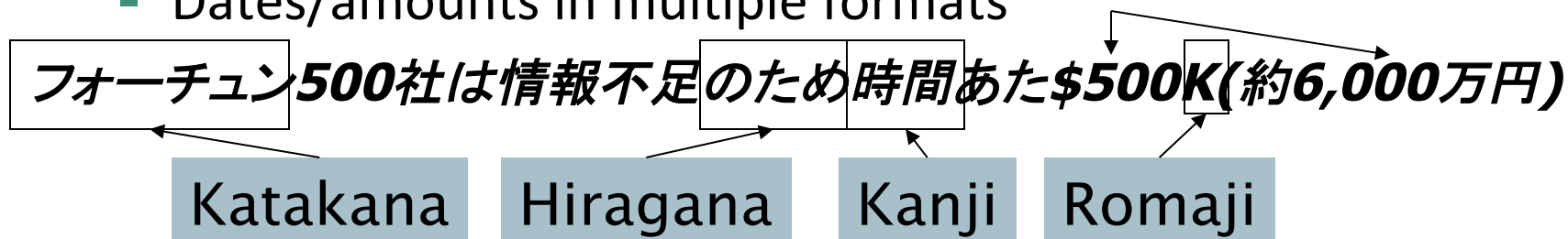
End-user can express query entirely in hiragana!

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.

- ← → ← →   ← 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

13

Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. 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 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 (lecture 7) 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"

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 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**
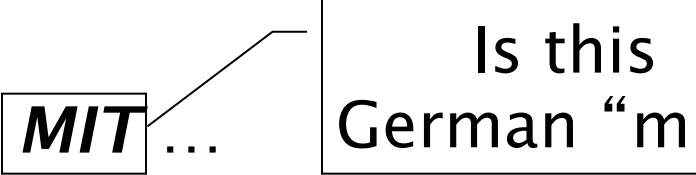
Normalization: other languages

- 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?

- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - ***Tuebingen, Tübingen, Tubingen ➜ Tubingen***

Normalization: other languages

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

- Tokenization and normalization may depend on the language and so 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...

- Google example:
  - Query **C.A.T.**
  - #1 result was for "cat" (well, Lolcats) *not* Caterpillar Inc.

I keepz ur beerz till I getz toona

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

Thesauri and soundex

- 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 document contains *automobile*, index it under *car-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 lectures 3 and 9

Lemmatization

- Reduce inflectional/variant forms to base form

- E.g.,

  - *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

- 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.*

Typical rules in Porter

- *sses → ss*

- *ies → i*

- *ational → ate*

- *tional → tion*

- Rules sensitive to the *measure* of words

- *(m>1) EMENT →*
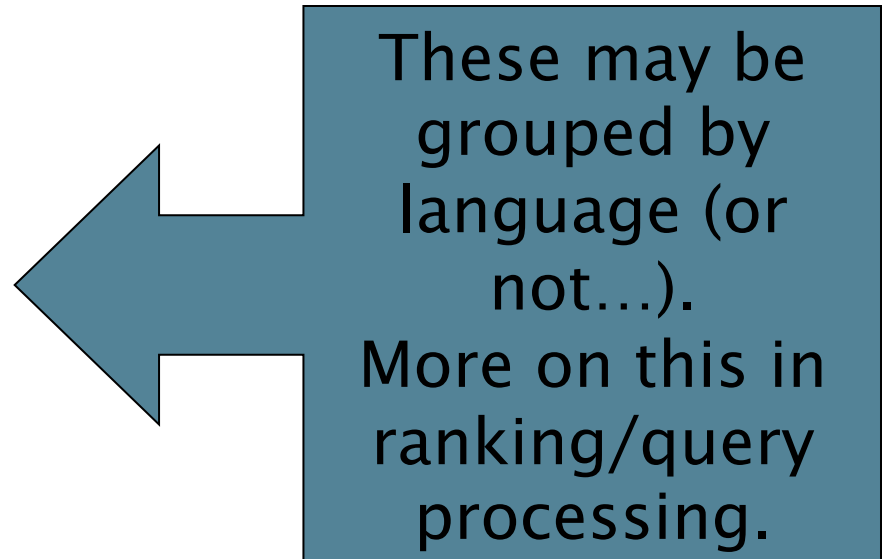   - *replacement → replac*
    - *cement → cement*

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!

Language-specificity

- Many of the above features embody transformations that are

  - Language-specific and

  - 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.

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
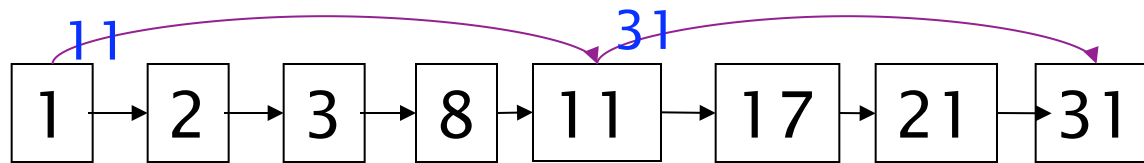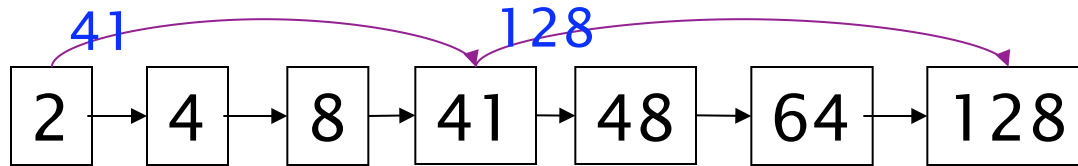
2	4	8	41	48	64	128	*Brutus*

2 → 8 ⬅

| 1 | 2 | 3 | 8 | 11 | 17 | 21 | 31 | *Caesar*
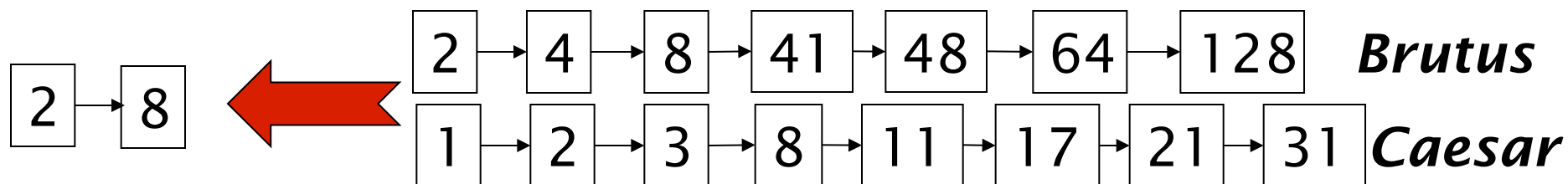
If the list lengths are m and n, the merge takes O($m+n$) operations.
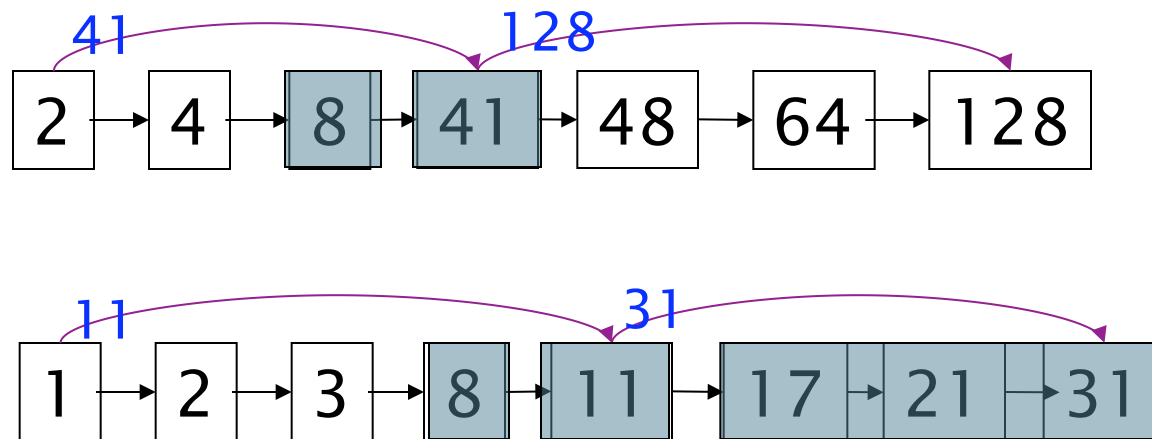
Can we do better?
Yes (if index isn't changing too fast).

Augment postings with skip pointers (at indexing time)

- Why?
- <u>To skip postings that will not figure in the search results.</u>
- 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 → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.
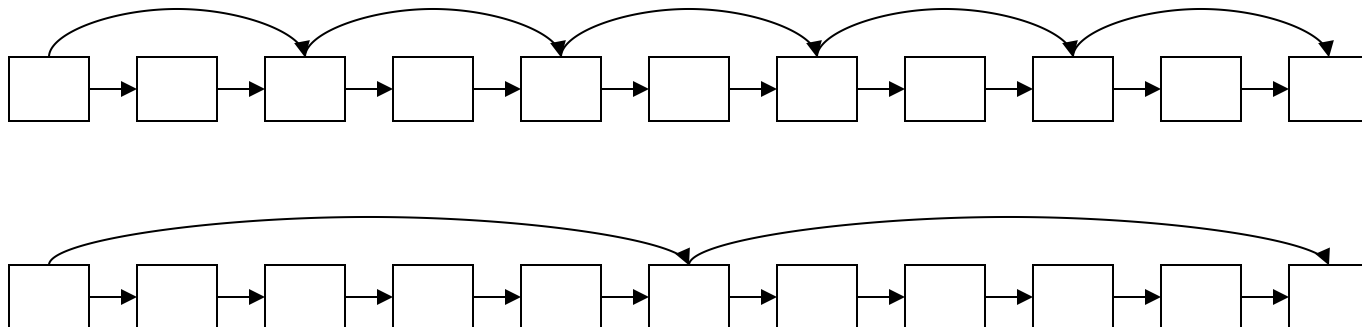
Placing skips

- Simple heuristic: for postings of length *L*, use $\sqrt{L}$ evenly-spaced skip pointers.

- This ignores the distribution of query terms.

- Easy if the index is relatively static; harder if *L* keeps changing because of updates.

- This definitely used to help; with modern hardware it may not (Bahle et al. 2002) unless you're memory-based

  - 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

- 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

  - Many more queries are *implicit phrase queries*

- For this, it 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, 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).
- Call any string of terms of the form NX*N 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
- Query processing: parse it into N's and X's
  - Segment query into enhanced biwords
  - Look up in index: ***catcher rye***

Issues for biword indexes

- False positives, as noted before

- Index blowup due to bigger dictionary

  - Infeasible for more than biwords, big even for them

- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Solution 2: Positional indexes

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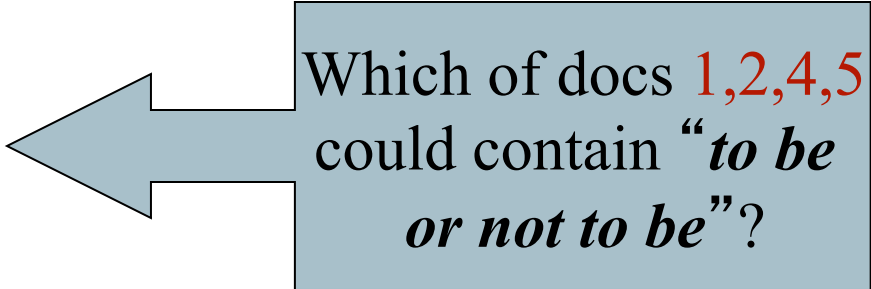
  <**term**, number of docs containing **term**;

  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  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*"?

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

- But we now need to deal with more than just equality

41

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**
  - 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*?
  - This is a little tricky to do correctly and efficiently
  - See Figure 2.12 of IIR
  - There's likely to be a problem on it!

Positional index size

- You can compress position values/offsets: we'll talk about that in lecture 5

- Nevertheless, a positional index expands postings storage *substantially*

- Nevertheless, a 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 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

- 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", "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

Resources for today's lecture

- IIR 2

- MG 3.6, 4.3; MIR 7.2

- 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://www.seg.rmit.edu.au/research/research.php?author=4

- D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.