CS3245

Information Retrieval

Lecture 3: Postings lists and Choosing terms

3

Last Time: Basic IR system structure

- Basic inverted indexes:
  - In memory dictionary and on disk postings

BRUTUS	→	1	2	4	11	31	45	173	174

CAESAR	→	1	2	4	5	6	16	57	132	. . .

CALPURNIA	→	2	31	54	101

  - Key characteristic: Sorted order for postings

- Boolean query processing
  - Intersection by linear time "merging"
  - Simple optimizations by expected size

Today: Zoom In on Postings Data Structure and How to Define Terms

- **Postings**
  - Faster merges: skip lists
  - Positional postings and phrase queries

- **Preprocessing to form the term vocabulary**
  - Documents
  - Tokenization
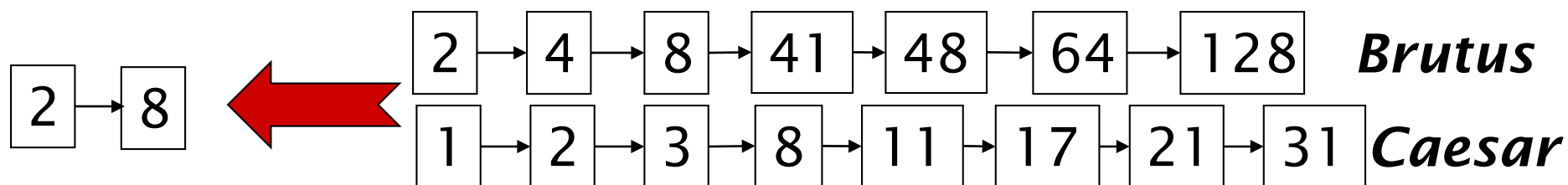 - What *terms* do we put in the index?

FASTER POSTINGS MERGES: SKIP POINTERS / SKIP LISTS

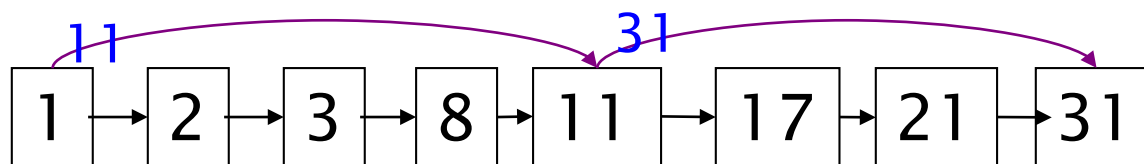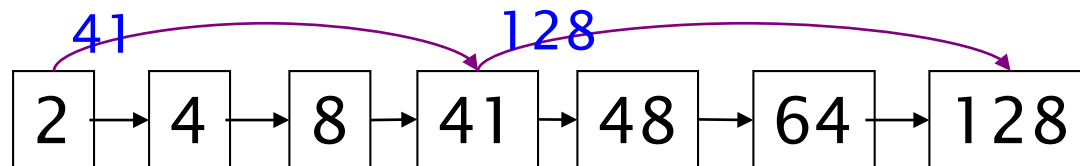*Blanks on slides, you may want to fill in*

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*** |

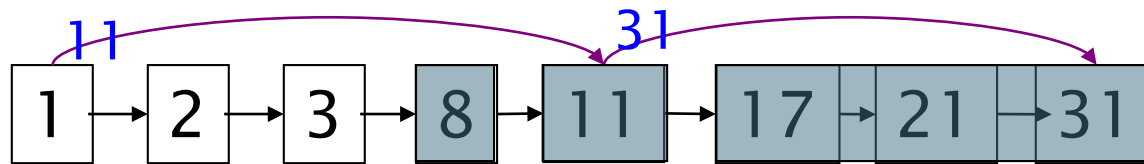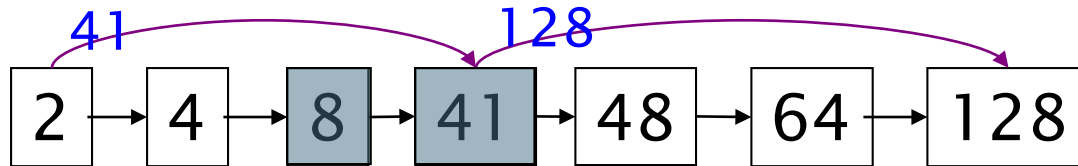If the list lengths are *m* and *n*, the merge takes O($m+n$) operations.

Can we do better?

Adding skip pointers to postings

- Done at indexing time.

- To skip postings that will not figure in the search results.

- How to do it? And 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.
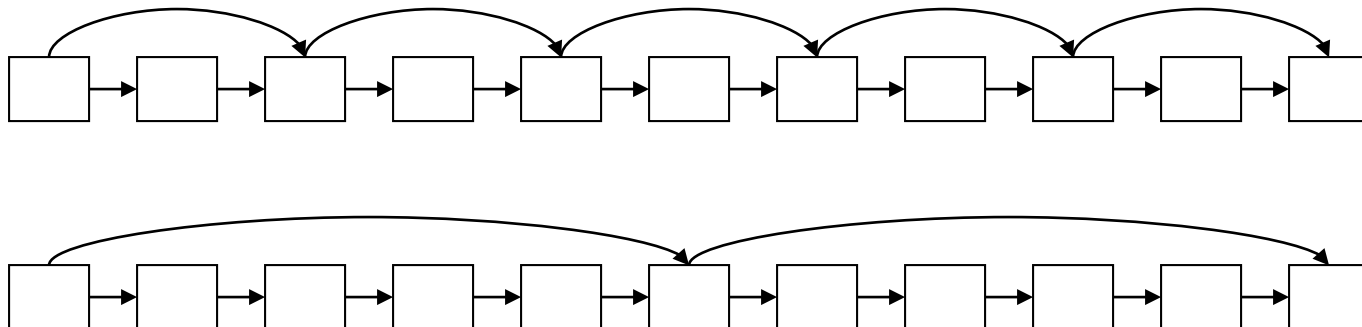
Placing skips

- Simple heuristic: for postings of length *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 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 INDICES

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 (for users; they "get it")

  - 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: bigram model using words

- 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 be processed as a Boolean query on biwords:

  stanford university palo alto →

  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, why?

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 extended biword.
  - Each extended biword is now 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 … ;
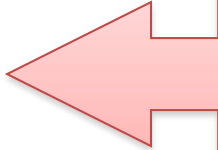
doc2: position1, position2 … ;

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, ...>

> Quick check:
> 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

- Now need to deal with more than just equality

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.

Positional index size

- We can compress position values/offsets, later in index compression

- 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%

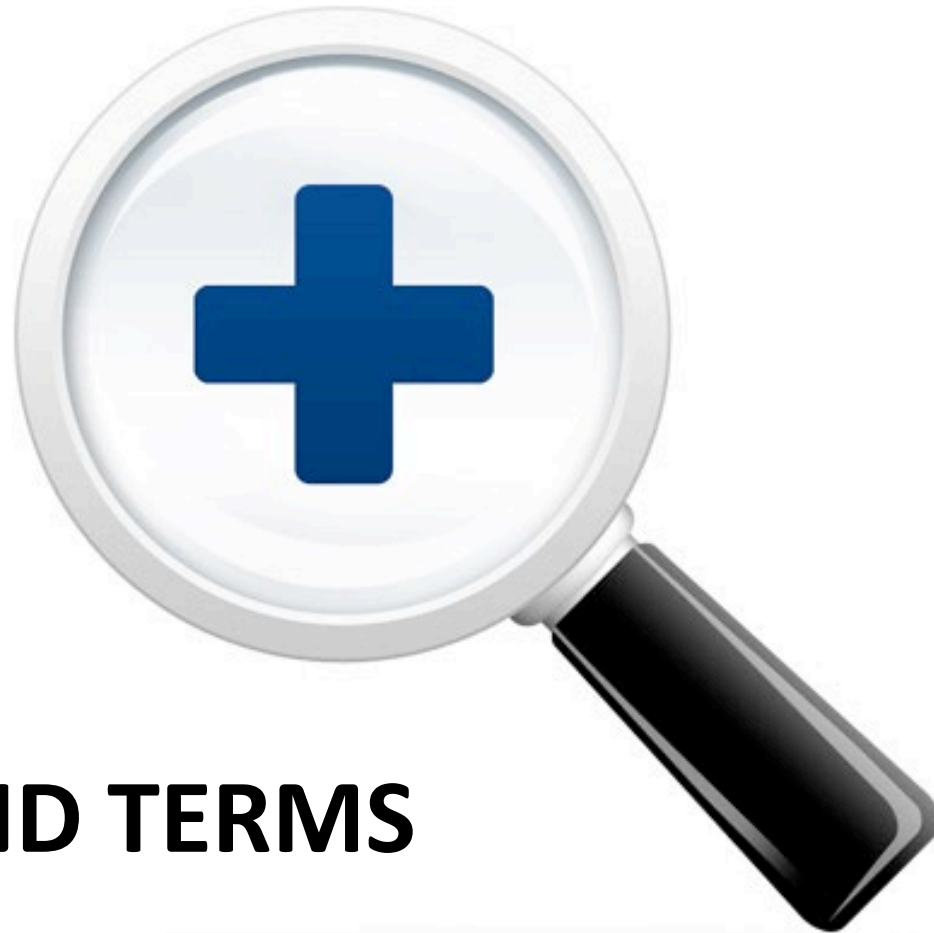| Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- A positional index is 2–4x larger as a non-positional index

- Positional index size is ~35–50% of the volume of original text

- Caveat: all of this holds for "English-like" languages

Combining biword and positional indices

- These two approaches can be profitably combined
  - For particular oft-queried 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
  - Added a next word index, recording words that follow a given word
  - A typical web query mixture was executed in ¼ of the time of using just a positional index
  - Required 26% more space over just having a positional index

TOKENS AND TERMS

Recap: Inverted index construction

Documents to be indexed.

Doc 1

Friends, Romans, countrymen.

Token stream.

Focus for today

Tokenizer

Friends | Romans | Countrymen

Linguistic modules

Modified tokens.

friend | roman | countryman

Indexer

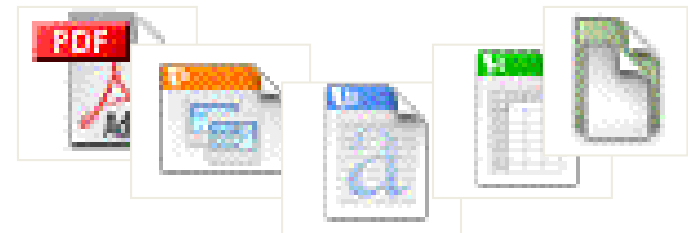Inverted index.

friend ⟹ 2 → 4

roman ⟹ 1 → 2

countryman ⟹ 1 → 16
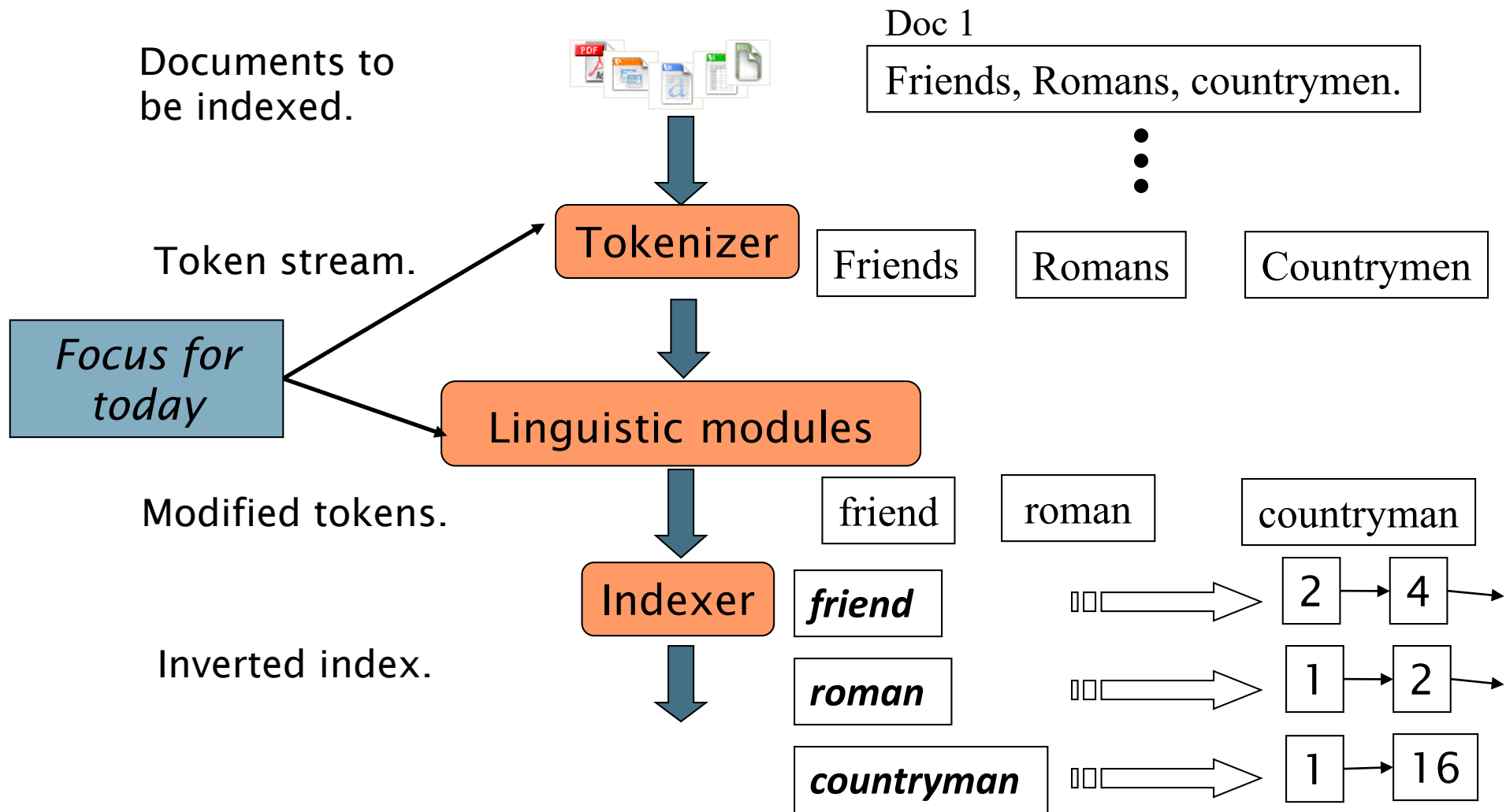
First step: Text extraction

- What format is it in?
  - PDF / Word / Excel / HTML?
- What language is used?
- What character set is used?

- Beyond the scope of this course, but most of the time are done heuristically, or assumed to be non-issues with help from vendor libraries

Complications: Format / Language

- Collection may have docs in different languages
  - A single index may have to contain terms of several languages.

- Even single documents may have multiple languages / formats
  - French email with a German PDF attachment
  - Crazy lecturer's homework assignment

Do not step beyond the Yellow Line until train stops
列 车 未 停 前 勿 跨 越 黄 线
Jangan melintas garisan kuning sebelum keretapi berhenti
ரயில் வண்டி நிற்கும் வரை, மஞ்சள்
கோட்டுக்கு மேலே போக வேண்டாம்

Photo Credits: Wikipedia commons

Granularity of indexing

- What should the unit document be?
  - A file?
  - An email? (Perhaps one of many in a mailbox / thread)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)

Collection? Set of documents? A document? Section of a document? Paragraph? Sentence? Word?

- Too coarse grained

- Too fine grained

Need to decide based on projected use of the IR engine

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is an instance of a sequence of characters grouped together as a useful semantic unit
- Each token is a candidate for an index entry, after further processing
- But what are valid tokens to emit?

(English) Tokenization: Issues in Handling Apostrophe, Hyphens and Spaces

- ***Finland's capital*** $\rightarrow$ ***Finland*?** ***Finlands*?** ***Finland's*?**

- ***Aren't*** $\rightarrow$ ***Aren*** and ***t*?** ***Are*** and ***n't*?** ***Are*** and ***not*?**

- ***Hewlett-Packard*** $\rightarrow$ ***Hewlett*** and ***Packard*?**
  - ***state-of-the-art***: break up hyphenated sequence.
  - ***co-education***
  - ***lowercase***, ***lower-case***, ***lower case***: all acceptable forms

- ***San Francisco***: one token or two?
  - How did you decide it is one token?

- What about ***Los Angeles-San Francisco*?**

Numbers, dates and other dangerous things

- ***3/20/13   Mar. 12, 2013   20/3/13***

- ***55 B.C.***

- ***B-52***

- ***My PGP key is 324a3df234cb23e***

- ***(800) 234-2333***

  - Often have embedded spaces, punctuation

  - Older IR systems may not index numbers

    - But often very useful: think about things like looking up error codes / product codes on the web

  - IR systems often opt to 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, which can give a 15% performance boost

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。

    Shā lā bō wá   xiànzài jūzhù   zài měiguó dōngnán bù de fóluólǐdá

  - Not always guaranteed a unique tokenization

- Japanese intermingles multiple writing systems

  - Dates / amounts in multiple formats

Fōchun   gohyaku-sha wa   jōhō fusoku no tame   jikan ata   gozyu man-doru (yaku rokusen man-en)

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 often express queries entirely in Hiragana!

Tokenization: language issues

- Arabic (or Hebrew) is written right to left, but certain items (e.g., numbers) are written left to right

- Words are separated, but letter forms within a word form complex ligatures

fi → fi	Example of a ligature in modern English
fl → fl	

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

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

← →   ← →   ← start

With Unicode, the surface presentation is complex (left to the renderer to solve), but the stored form is in linear order

Unicode in XKCD

Collaborative editing can quickly become a textual rap battle fought with increasingly convoluted invocations of U+202a to U+202e.   #geekout

http://xkcd.com/1137/

Stop word removal

- With a stop list, we exclude the most common words from the dictionary. Intuition:
  - They have little semantic content: *the, a, and, to, be*

- But the trend is away from doing this:
  - Good compression techniques 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"

Normalizing tokens 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

- Often, we 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***

Normalizing tokens 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 often less efficient

Removing accents and diacritics

- Accents: e.g., French ***résumé*** vs. ***resume.***

- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***

- Most important criterion:

  - How are your users like to write their queries for these words?

  - Even in languages that have accents, users often may not type them

  - Thus, often best to normalize to a de-accented term

    - ***Tuebingen, Tübingen, Tubingen ▸ Tubingen***

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 lowercase everything, since users' queries most often written this way

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

I keepz ur beerz till I getz toona

Other techniques

- Spelling / format variations?
  - by hand-crafted rules
    - *color = colour*
    - *3/12/91 = Mar. 12, 1991*

- Synonyms?
  - by thesaurus
    - *car ≈ automobile*

- Transliteration variations?
  - by Soundex (to be covered next week)
    - *Beijing = Peking*

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 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

- Most common algorithm for stemming English
    - Experiments 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*

Late phase rules in Porter check the length of the resulting word:

- *(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)

- Lemmatizer – Full morphological analysis to return (dictionary) base form of word
  - At most modest benefits for retrieval

- Do stemming and other normalizations help?
  - English: very mixed results. Helps recall for some queries but harms precision on others
    - E.g., operating system ⇒ oper sys
  - 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 them

- Shows the intertwining of NLP with IR
  PSA: take the NLP course to learn more!

Summary

Zoomed in on three issues:

1. Faster merging of posting lists: Skip pointers

2. Handling of phrase and proximity queries
   - Biword Indices
   - Positional Indices

3. Steps in choosing terms for the dictionary
   - Text extraction
   - Granularity of indexing
   - Tokenization
   - Stop word removal
   - Normalization
   - Lemmatization and stemming

Resources for today's lecture

- IIR 2
- MG 3.6, 4.3; MIR 7.2

- 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. 2002. Efficient phrase querying with an auxiliary index. SIGIR, pp. 215-221.
- Porter's stemmer: http://www.tartarus.org/~martin/PorterStemmer/
- Stemming and Lemmatization in NLTK