CS3245

Information Retrieval

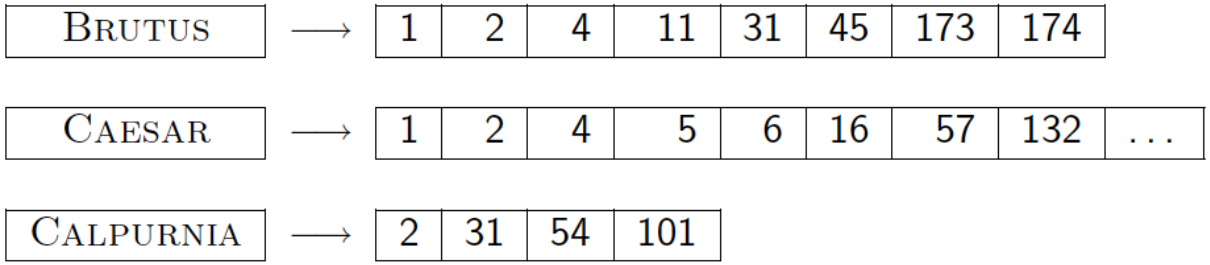
3

Lecture 3: Postings lists and Choosing terms

Live Q&A
https://pollev.com/jin

Last Time: Basic IR system structure

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



  - Key characteristic: Sorted order for postings

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

NUS National University of Singapore

Today

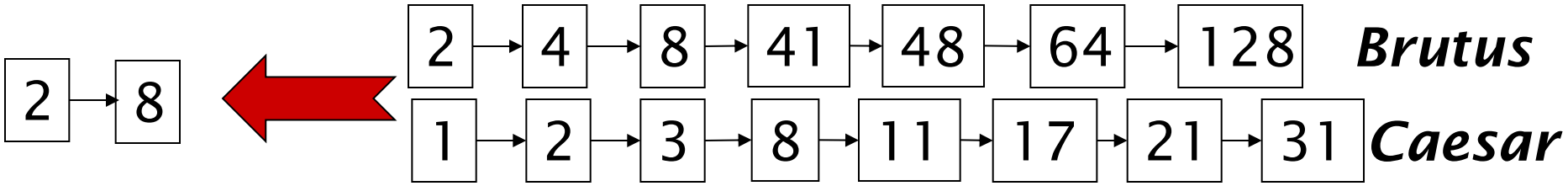
- Enhanced posting lists
  - Faster merges: skip pointers
  - Handling phrase queries: Biword index and Positional index
- Choosing terms for the dictionary
  - Document-level processing
  - Word-level processing

FASTER MERGES: SKIP POINTERS

Blanks on slides, you may want to fill in

Recall basic merge

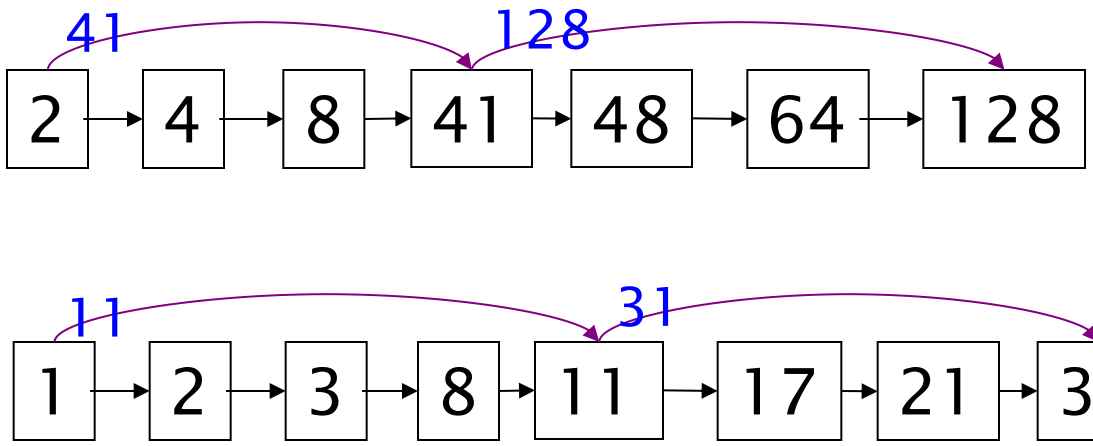
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

Can we do better?

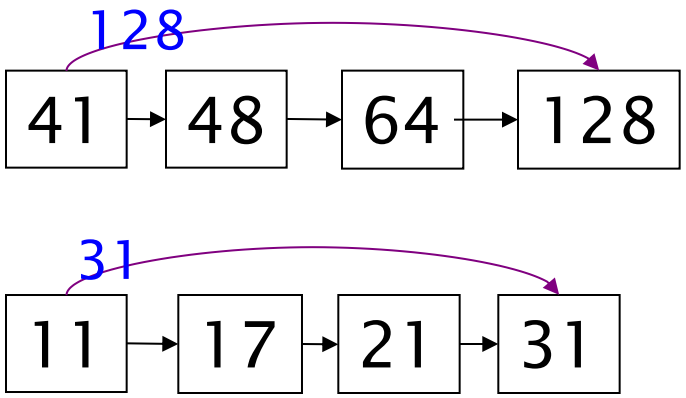
Adding skip pointers to postings



- Used to **skip postings** that are not part of the results
  - How to use them?

- Added during **indexing time**
  - Where to place them?

Query processing with skip pointers

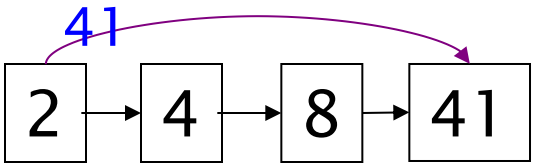


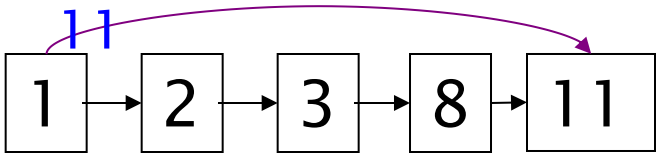
Suppose we are looking at **41** (upper) and **11** (lower).

11 is smaller and the skip successor of **11** on the lower list is **31.**

It is safe to skip to **31** directly. (Why?)

Query processing with skip pointers





Suppose we are looking at **2** (upper) and **1** (lower)

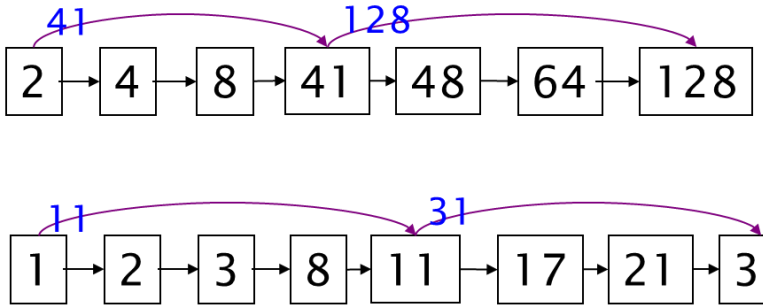
The skip successor of **1** on the lower list is **11**.

It is NOT safe to skip! (Why?)

We advance to 2 in lower list…

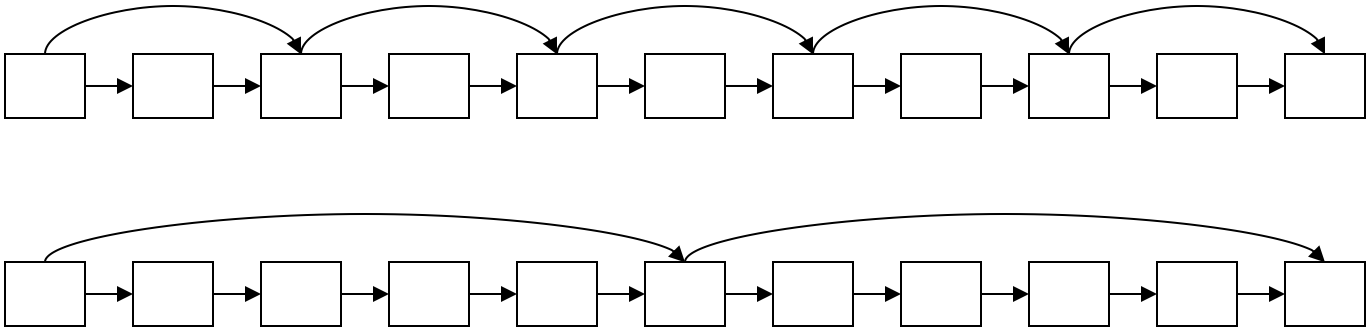
Query processing with skip pointers

```
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
```



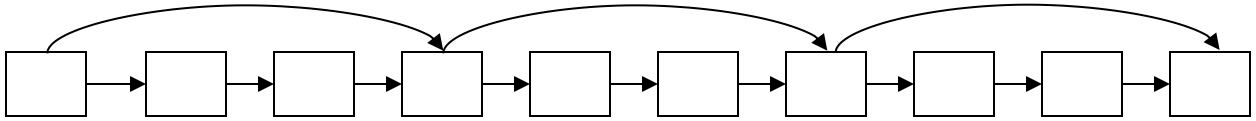
Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - 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 docIDs.



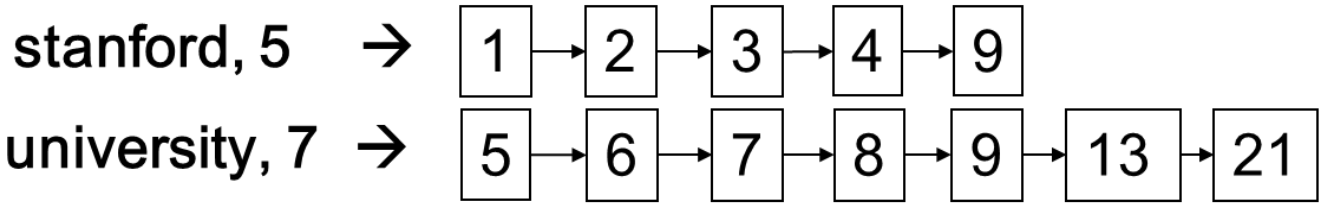
- This definitely used to help; but we need to be aware of the cost!
  - Pointer comparison
  - Disk space and I/O time for storing and loading a bigger list
  - Updating of pointers in a dynamic list

HANDLING PHRASE QUERIES

Phrase queries

- Want to be able to answer queries such as "***stanford university***" – as a phrase
  - Not the same as stanford AND university
  - Popular and easy to understand
  - E.g., *"I went to Stanford University"* is a match, but *"I went to university at Stanford"* is not.

- Not suffice to store individual terms with the docIDs.



A first attempt: Biword indexes

- **Index** every consecutive pair of terms in the text
- E.g., "I went to Stanford University"
  - 4 biwords: ***I went, went to, to Stanford, Stanford University***

    stanford university, 1 → 9
- **Process** the **two-word** phrase queries by looking up the biwords directly.
- How about **longer** phrase queries?

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***

- There could be false positives… **Why?**

Extended biwords

- **Index** all extended biwords
  - In the form NX*N, where N = Noun, X = Articles / Prepositions (Part-of-speech-tagging required)
- E.g., ***catcher in the rye***

  N X X N
  - 1 extended biword: ***catcher rye***
- **Process** phrase queries by extracting and looking up the extended biwords
- There could be false positives, too. **Why?**

Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- Not the standard solution 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***, document frequency;
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***"?

Processing a phrase query

- Query: “to be or not to be”

- Retrieve the lists for each term
- Merge at the document level and then position level.
  - ***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; ...
- Need to deal with more than just equality / intersection…

- But *proximity* queries can be handled as well…

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

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

Positional index size

- A positional index expands the storage *substantially*
  - 2-4x larger as a non-positional index
  - ~35-50% of the volume of original text
  - But we can compress position values/offsets, later in index compression

For "English-like" languages

- It 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.

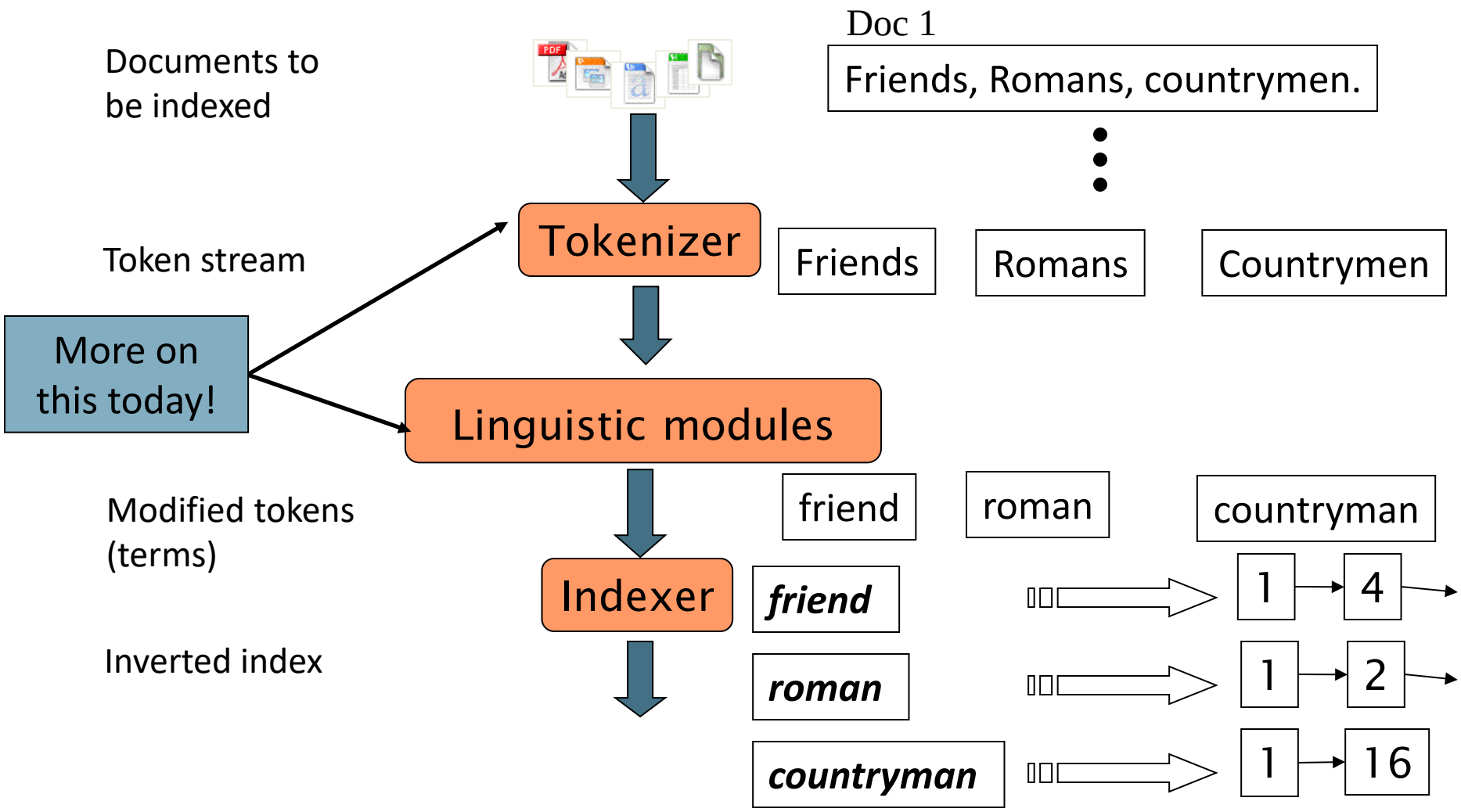
Combining biword and positional indices

- Merging is slow in positional indices!

- Possible enhancement: Index popular bi-word from based on the query log
  - E.g., "Michael Jackson", "Britney Spears"
  - Retrieve the postings without merging (at the cost of some additional storage)

CHOOSING TERMS

Recap: Inverted index construction



First step: Text extraction

- Formats
  - TXT / HTML / WORD / PDF / JPG?
- Languages
  - English / Chinese / Malay?
  - Or even a mix…
- Character sets
  - ASCII / UTF-8 / ISO-8859-1?

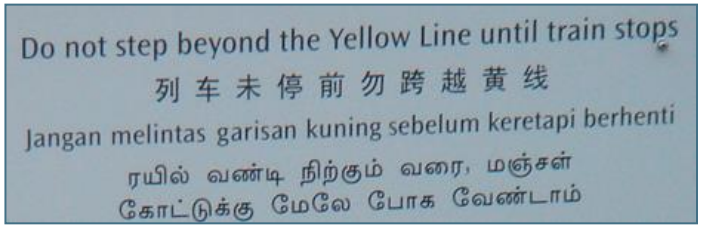


Photo Credits: Wikipedia commons

- *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

Blanks on slides, you may want to fill in

Granularity of indexing

- What should the unit document be?
  - A book
  - A chapter?
  - A sentence?
  - A word?
- Too coarse grained:
- Too fine grained:

Need to decide based on projected use of the IR engine

Tokenization

- Input: "***Friends, Romans, Countrymen, lend me your ears;***"
- Output: Tokens

Friends	Romans	Countrymen	
lend	me	your	ears

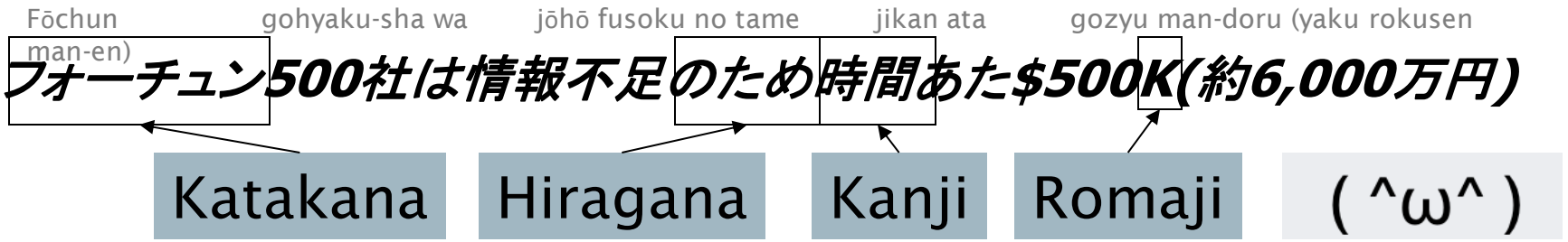
- A token is an instance of a sequence of characters grouped together as a useful semantic unit
- Each token is a candidate for an index entry (i.e., a term), after further processing
- But what are **valid** tokens to emit?

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

- ***Finland's capital*** → ***Finland***? ***Finlands***? ***Finland's***?
- ***Aren't*** → ***Aren*** and ***t***? ***Are*** and ***n't***? ***Are*** and ***not***?
- ***Hewlett-Packard*** → ***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***?

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



  - End-user often express queries entirely in Hiragana!

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.

Stop word removal

- With a stop list, we exclude the most common words from the dictionary.
  - They have little semantic content: *the, a, and, to, be*
  - Yet they take up a lot of space **Why?**
- But the trend is away from doing this:
  - Good compression techniques reduces the space needed for storage
  - Useful in for many queries
    - Phrase queries: “Prince of Denmark", "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

Normalizing tokens to terms

- A simple approach: Dropping some punctuations
  - deleting periods
    - ***U.S.A., USA ▸ USA***
  - deleting hyphens
    - ***anti-discriminatory, antidiscriminatory ▸ antidiscriminatory***
  - deleting accents
    - ***Tuebingen, Tübingen, Tubingen ▸ Tubingen***

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

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.

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 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!

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***

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

- 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