

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

- Enhanced posting lists
 - Faster merges: skip lists
 - Positional postings and phrase queries
- Choosing terms for the dictionary
 - Document-level / Word-level pre-processing
 - What *terms* do we put in the index?



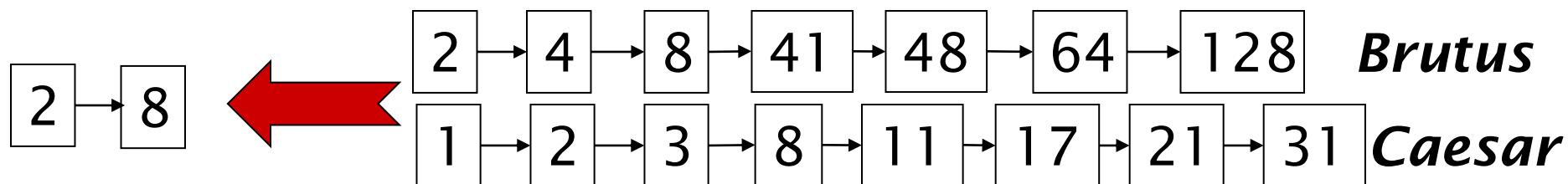
FASTER POSTINGS MERGES: SKIP POINTERS / SKIP LISTS

Blanks on slides, you may want to fill in



Recall basic merge

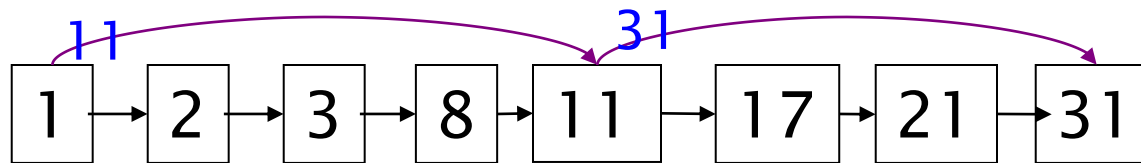
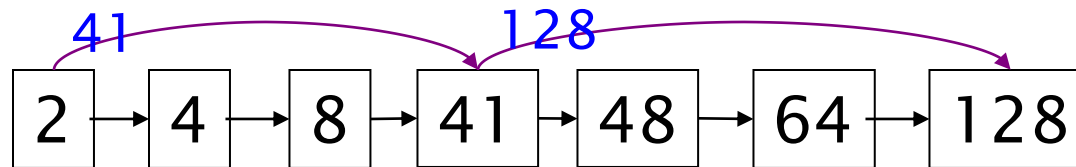
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

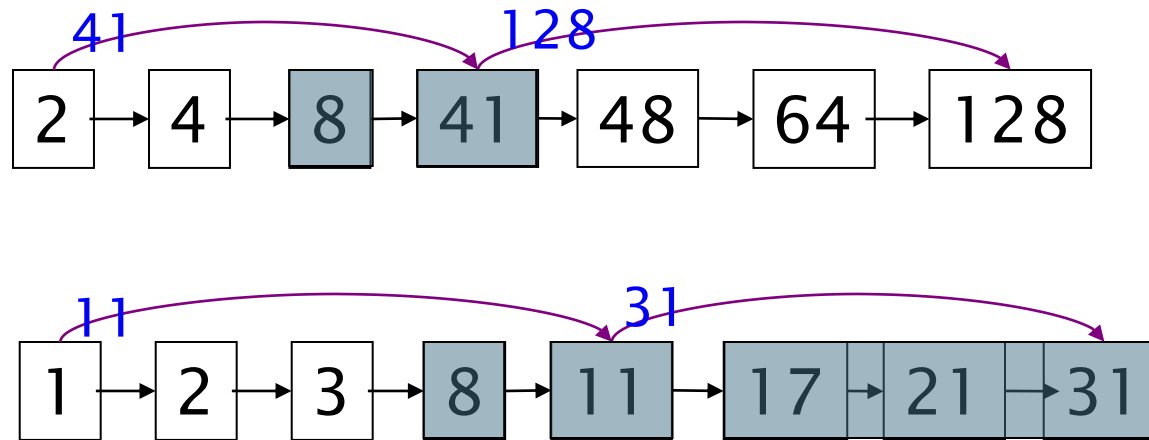
Can we do better?

Adding skip pointers to postings



- Done at indexing time.
- To skip postings that will not figure in the search results.
- How to do it (for intersection)?
- 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**. **11** is smaller.

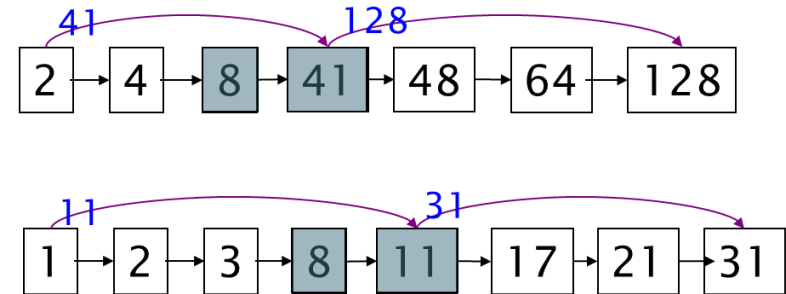
But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

Query processing with skip pointers

INTERSECTWITHSKIPS(p_1, p_2)

```

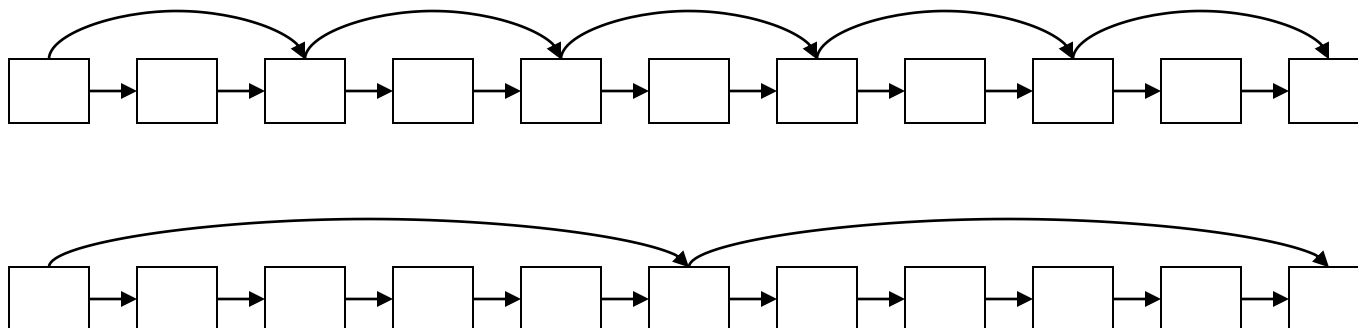
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9          then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12      else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13          then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14             do  $p_2 \leftarrow \text{skip}(p_2)$ 
15             else  $p_2 \leftarrow \text{next}(p_2)$ 
16  return answer
  
```



Where do we place skips?



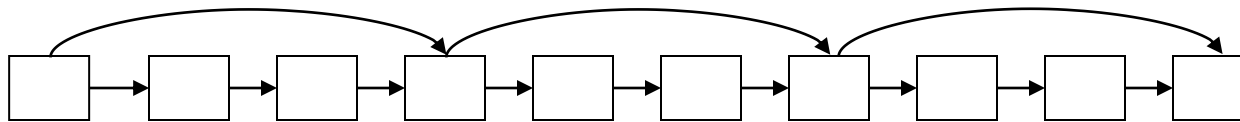
- Tradeoff:
 - More skips \rightarrow shorter skip spans \Rightarrow more likely to skip.
But lots of comparisons to skip pointers.
 - Fewer skips \rightarrow few pointer comparison, but then long skip spans \Rightarrow few successful skips.



Placing skips



- Simple heuristic: for postings of length L , use \sqrt{L} evenly-spaced skip pointers.
 - This ignores the distribution of query terms.



- 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



PHRASE QUERIES AND POSITIONAL INDICES

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.

stanford, 5 →

1	→	2	→	3	→	4	→	9
---	---	---	---	---	---	---	---	---

university, 7 →

5	→	6	→	7	→	8	→	9	→	13	→	21
---	---	---	---	---	---	---	---	---	---	----	---	----

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.

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
- 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*, document frequency;

doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>

Positional index example



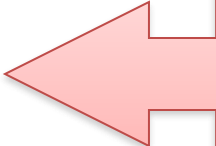
<**be**: 993427;

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"?

- 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

- 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
1 000	1	1
1 00,000	1	1 00

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

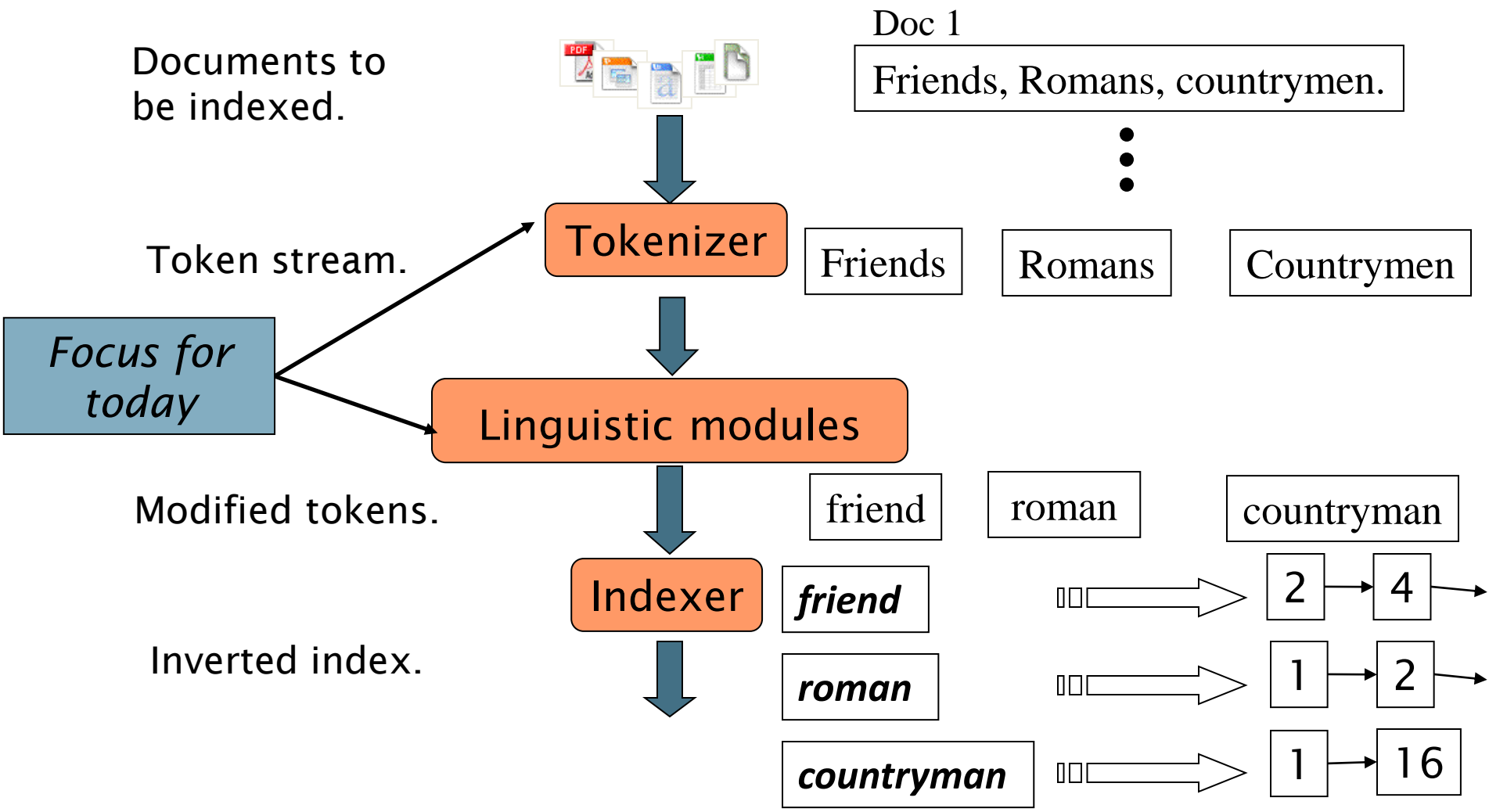


- Merging is slow in positional indices!
- Possible enhancement: Index popular bi-word from based on the query log
 - E.g., "Michael Jackson", "Britney Spears"
 - Retrieve the postings without merging (at the cost of some additional storage)



CHOOSING TERMS

Recap: Inverted index construction



First step: Text extraction



- Formats
 - PDF / Word / Excel / HTML?
- Languages
 - English / Chinese / Malay?
 - Or even a mix...
- Character sets

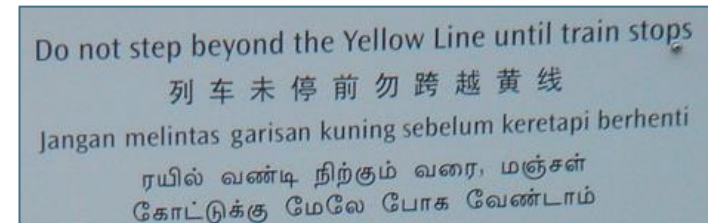


Photo Credits: Wikipedia commons

- 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

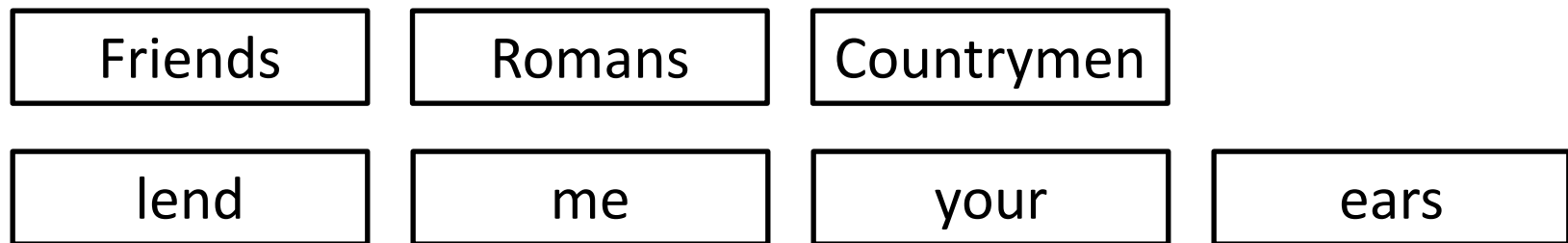
- What should the unit document be?
 - A book
 - A chapter?
 - A sentence?
 - A word?
- Too coarse grained:
- Too fine grained:

Need to decide based on projected use of the IR engine

Tokenization



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



- 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 fóluó lǐ dá

- Not always guaranteed a unique tokenization

- Japanese intermingles multiple writing systems

- Dates / amounts in multiple formats

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

- 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
fl → fl
Example of a ligature in modern English

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

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. 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

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.



I keepz ur beerz till I getz toona

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 example compress and compress are both accepted as equivalent 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:

- $(m > 1)$ *ELEMENT* → ""
 - *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 \Rightarrow 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* \approx *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