Part 3: The term vocabulary, postings lists and tolerant retrieval

Francesco Ricci

Most of these slides comes from the course:

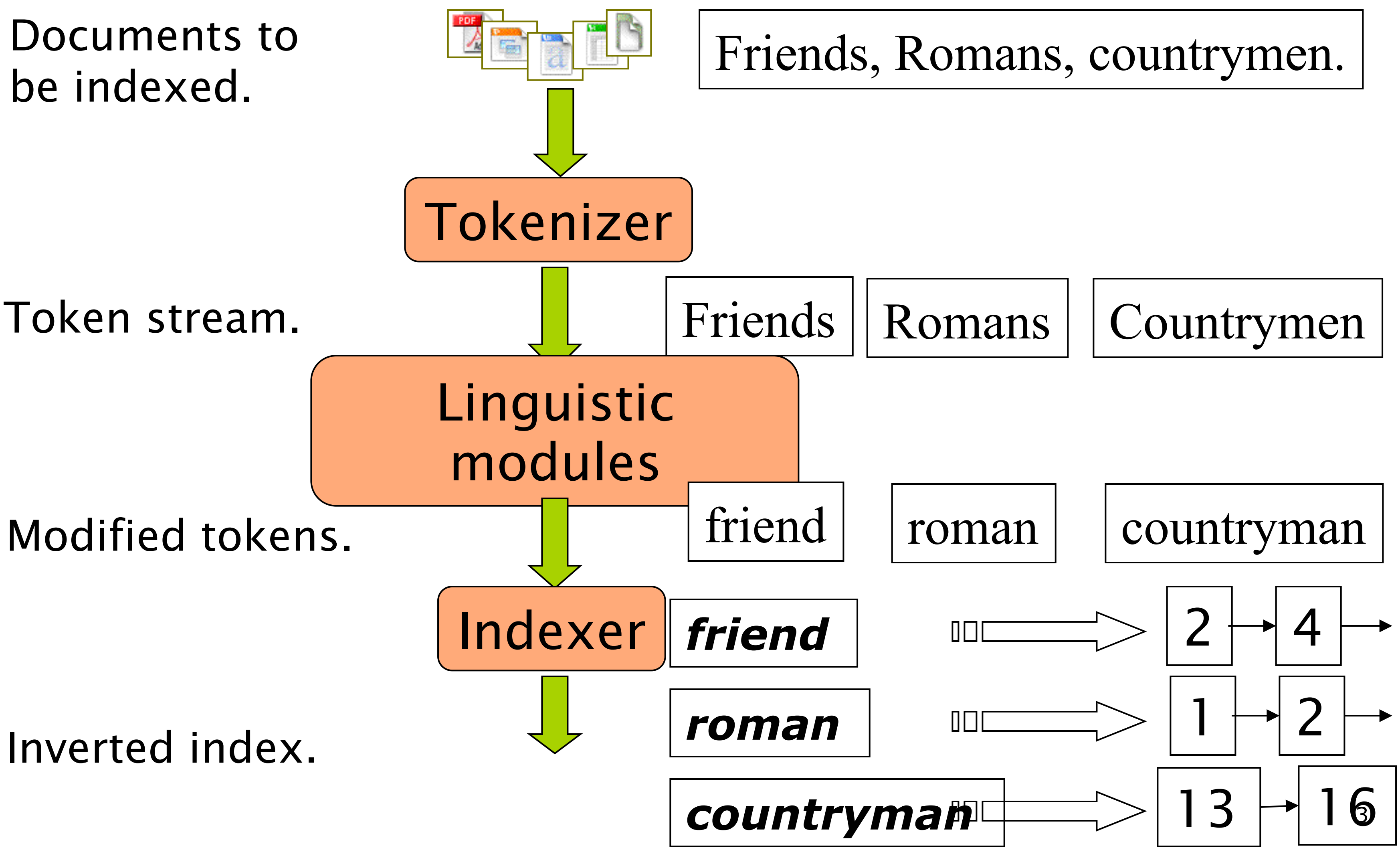
Information Retrieval and Web Search,

Christopher Manning and Prabhakar Raghavan

Content

- Elaborate basic indexing
- Preprocessing to form the term vocabulary
  - Documents
  - Tokenization
  - What *terms* do we put in the index?
- Postings
  - Phrase queries and positional postings
- "Tolerant" retrieval
  - Wild-card queries
  - Spelling correction
  - Soundex

Recall the basic indexing pipeline



Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set encoding 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**:
  - The classification is predicted with simple rules
  - Example: "if there are many `the' then it is English".

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 a mbox)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages).

TOKENS AND TERMS

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- *A token is an **instance** of a **sequence of characters***
- 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*** ?
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

General Idea

- If you consider 2 tokens (e.g. splitting words with hyphens) then queries containing only one of the two tokens will **match**
  - Ex1. Hewlett-Packard – a query for "packard" will retrieve documents about "Hewlett-Packard" *OK?*
  - Ex2. San Francisco – a query for "francisco" will match docs about "San Francisco" *OK?*
- If you consider 1 token then query containing only one of the two possible tokens will **not match**
  - Ex3. co-education – a query for "education" will not match docs about "co-education".

Numbers

- ***3/20/91*** ***Mar. 12, 1991*** ***20/3/91***
- ***55 B.C.***
- ***B-52***
- ***My PGP key is 324a3df234cb23e***
- ***(800) 234-2333***

- Often have embedded spaces (*but we should not split the token*)
- Older IR systems may not index numbers
  - But often very useful: think about things like looking up error codes/stacktraces on the web
- 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 now, 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.

Stop words

- With a stop list, you **exclude** from the dictionary entirely **the commonest words**:
  - **Little semantic content**: *the, a, and, to, be*
  - **Many of them**: ~30% of (positional) postings for top 30 words
- 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"

Reuters RCV-1

	(distinct) terms			nonpositional postings			tokens (= number of positi entries in postings)		
	number	Δ%	T%	number	Δ%	T%	number	Δ%	T%
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	−2	−2	100,680,242	−8	−8	179,158,204	−9	−9
case folding	391,523	−17	−19	96,969,056	−3	−12	179,158,204	−0	−9
30 stop words	391,493	−0	−19	83,390,443	−14	−24	121,857,825	−31	−38
150 stop words	391,373	−0	−19	67,001,847	−30	−39	94,516,599	−47	−52
stemming	322,383	−17	−33	63,812,300	−4	−42	94,516,599	−0	−52

- 800,000 Documents
- Average tokens per document: 247
- *If the documents are larger do you expect a bigger/ smaller reduction of nonpositional postings when eliminating stop words?*
- *Online text analysis: http://textalyser.net/*
- *Words frequency data http://www.wordfrequency.info*

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 a term: *a term is a (normalized) word type, which is an entry in our IR system dictionary*
- We define equivalence classes of terms by, e.g.,
  - deleting periods to form a term
    - ***U.S.A., USA*** $\in$ ***[USA]***
  - deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory*** $\in$ ***[antidiscriminatory]***

Equivalence class of a
$[a] = \{x \mid x \sim a\}$

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*** $\in$ ***[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.*

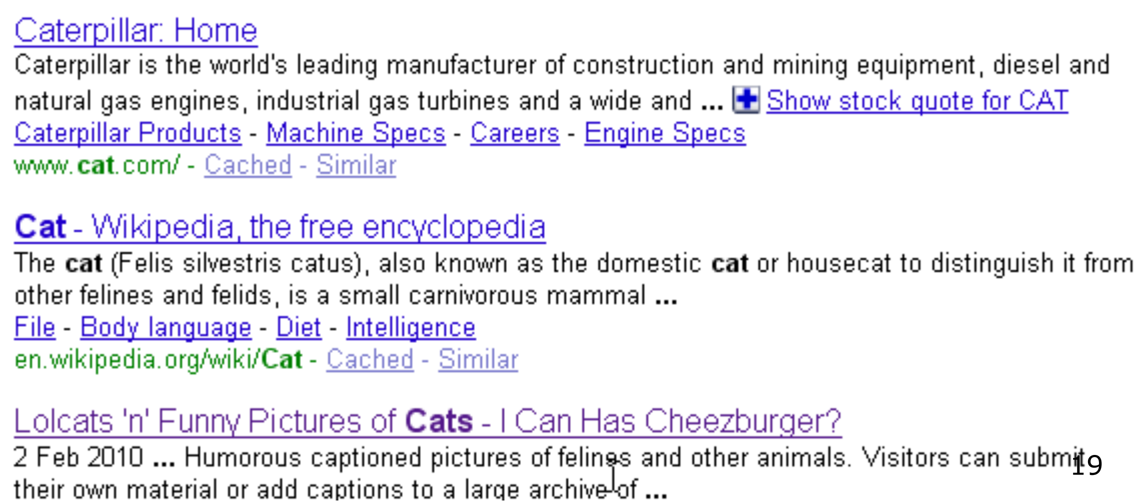
Case folding

- 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…

Federal reserve

Steel Authority of India

- Google example:
  - Query ***C.A.T.***
  - #1 result is for Caterpillar Inc., then "usual" cat



Normalization to terms

- An alternative to equivalence classing is to include in the dictionary many variants of a term and then do asymmetric expansion at query time

- An example of where this may be useful
  - User enters: ***window*** System searches: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient (Why?)

Thesauri and soundex

- Do we handle synonyms?
  - We can rewrite to form hand-constructed equivalence-class terms
    - ***Car* ~ *automobile*   *color* ~ *colour***
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or index the terms separately and expand at query time:
    - When the query contains ***automobile***, look under ***car*** as well *(but what expansions to consider?)*
- What about spelling mistakes?
  - One approach is soundex, which forms equivalence classes of words based on phonetic heuristics.
- And homonyms?

Lemmatization

- Reduce inflectional/variant forms to base form (the one that you search in your English dictionary)
- 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 is at least as good as other stemming options
- 5 phases of reductions and some conventions:
  - phases applied sequentially
  - each phase consists of a set of rules
  - sample convention: *of the rules in a group, select the one that applies to the **longest suffix**.*
- Girl**s**

"s" is the suffix

"Girl" is the stem

Typical rules in Porter

The longest suffix in these 4 rules

- *ational* → *ate*   (e.g., rational -> rate)
- *tional* → *tion*   (e.g., conventional -> convention)
- *sses* → *ss*   (e.g., guesses -> guess)
- *ies* → *i*   (e.g., dictionaries -> dictionari)

- Is the remaining word a stem? After the transformation the word should longer than a threshold (m = number of syllables)

  Rule: *(m>1) EMENT* →
  Examples:
  replacement → *replac*   (Yes)
  cement → *cement*   (No because "c" is not longer than 1 syllable)

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 for some queries but harms precision on others
    - E.g., operative (dentistry) ⇒ oper
  - Definitely useful for Spanish, German, Finnish, …
    - 30% performance gains for Finnish!

Examples

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

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.

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 to be 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

  1. More vocabulary's entries, OR
  2. The postings list structure must be expanded.

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
  - But, what about **three** words?

Longer phrase queries

- Longer phrases (more than 2) are processed as:
- ***"stanford university palo alto"*** can be broken into the Boolean query on biwords:
  - ***"stanford university"*** AND "***university palo***" AND "***palo alto***"

- BUT, without looking at 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 X* rye***
  - *But will also match docs containing "catcher with the 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, term-freq in Doc1: position1, position2 … ;
  Doc2, term-freq in Doc2: position1, position2 … ;
  etc.>

Positional index example

<***be***: 993427;
1, *6*: 7, 18, 33, 72, 86, 231;
2, *2*: 3, 149;
4, *5*: 17, 191, 291, 430, 434;
5, *9*: 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

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***:

  docId, term-freq in docId

  - *2, 5*: 1,17,74,222,551; *4, 5*: 8,16,190,429,433; *7, 3*: 13,23,191; ...
- ***be***:
  - *1, 2*: 17,19; *4, 5*: 17,191,291,430,440; *5, 3*: 14,19,101; ...
- Same general method for proximity searches

Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - /*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.

Positional Intersect

```
POSITIONALINTERSECT(p1, p2, k)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then l ← ⟨ ⟩
           pp1 ← positions(p1)
           pp2 ← positions(p2)
           while pp1 ≠ NIL
           do while pp2 ≠ NIL
                 do if |pos(pp1) − pos(pp2)| ≤ k
                         then ADD(l, pos(pp2))
                         else if pos(pp2) > pos(pp1)
                                 then break
                    pp2 ← next(pp2)
              while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
              do DELETE(l[0])
              for each ps ∈ l
              do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
              pp1 ← next(pp1)
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

New part to check proximity

Example k=2

- pp1=<1,3,5>, pp2 = <4,6,8> for DocID=77
- *L9* |1-4|<=2? No ; *L18* pp1=<3,5>
- *L9* |3-4|<=2? Yes; *L10* l=<4>; *L13* pp2=<6,8>
- *L9* |3-6|<=2? No;
- Check L14 |4-3|>2? No (so 4 is not deleted from l)
- *L17* Answer=<(77,3,4)>; *L18* pp1=<5>
- *L9* |5-6|<=2? *L10* Yes; l=<4,6>; *L13* pp2 =<8>
- *L9* |5-8|<=2? No
- Check L14 |4-5|>2? No (so 4 is not deleted from l)
- *L17* Answer=<(77,3,4) (77,5,4) (77,5,6)>

Positional index size

- You can compress position values/offsets (discussed in chapter 5 of IIR book)
- 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 (U.S. Securities and Exchange Commission), books, even some epic poems … can have easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Non pos. 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
- *Because we use position-ids and term-ids that are shorter than terms – otherwise positional index would even larger than original text*
- *Imagine what is the consequence for indexing the Web*
- 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"***
    - *(because the positional postings of these two very common terms will be very long)*
- *Use a biword index for certain queries and a positional index for others.*

WILD-CARD QUERIES

Wild-card queries: *

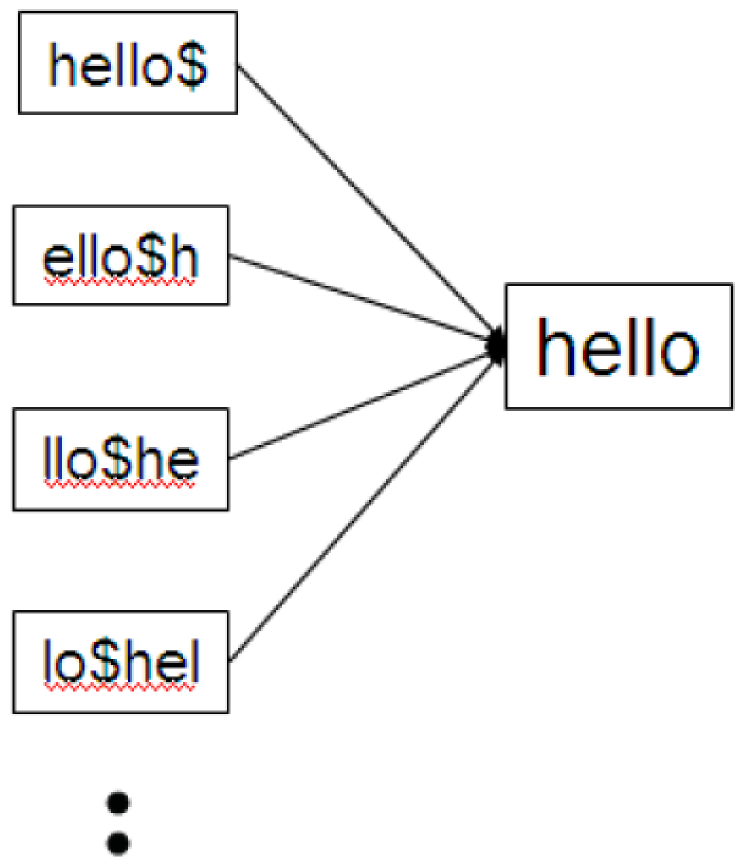
- ***mor*:*** find all docs containing any word beginning "mon"
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mor ≤ w < mos***
- ****mor:*** find words ending in "mon": *harder*
  - Maintain an additional B-tree for terms written *backwards*
  - So we can retrieve all words in range: ***rom ≤ w < ron***.

Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

Handling*'s in the middle

- How can we handle *'s in the middle of query term?
  - ***co*tion***
- We could look up ***co**** AND ****tion*** in a B-tree and intersect the two term sets
  - Expensive
- The solution: transform wild-card queries so that the *'s occur at the end
- This gives rise to the **Permuterm** Index.

Permuterm index example



▶ **Figure 3.3** A portion of a permuterm index.

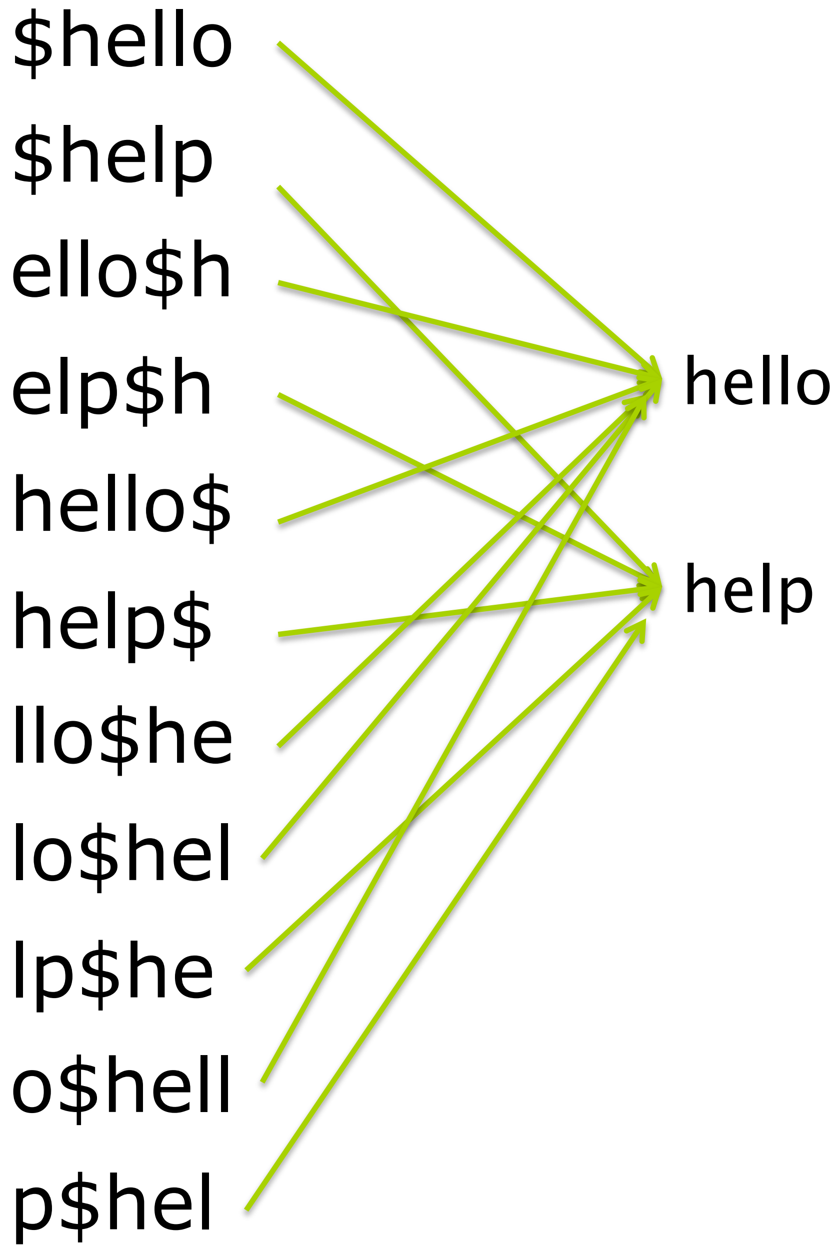
- From the permuterm you get the term and then from the standard index you get the documents containing the term.

Example

$hello
hello$
ello$h
llo$he → hello
lo$hel
o$hell

$help
help$
elp$h → help
lp$he
p$hel

Example


$hello
$help
ello$h
elp$h
hello$
help$
llo$he
lo$hel
lp$he
o$hell
p$hel
hello
help

Permuterm index

- For term **tech**, index the documents containing **tech** under multiple keys:
  - ***tech$, ech$t, chte, htec, $tech -*** where **$** is a special symbol
- Queries:
  - **tech** → lookup on **tech$** - will find only the key **tech** – and then retrieve the postings
  - **tech* →** lookup on **all** terms starting with **$tech** (**$tech***) – will find: **$tech, $technical, $technique, …** and then retrieve the postings of all these terms
  - ***tech →** lookup **tech$* - will find: tech$hi-, tech$air-, tech$**

Permuterm Index

- **X*Y** lookup on **Y$X***
  - Example: **m*n** → lookup on **n$m*** - will find **man**, **moron**, ecc
- The trick is:
  - Given a query with 1 wildcard, concatenate with $ (at the end) and the rotate the query until the wildcard is at the end

- The trick works also for this: ***tech*** → lookup on **tech*$* = tech*** - will find **tech$, tech$hi-, technical$, technical$hi-**

Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before
- Collect all the (permu)terms in the B-tree that are in the range specified by the wild-card (*first the permuterm and then the indexed terms*)
- Search in the inverted index all the documents indexed by these terms

- *Permuterm problem: ≈ quadruples lexicon size*

Empirical observation for English

Bigram (*k*-gram) indexes

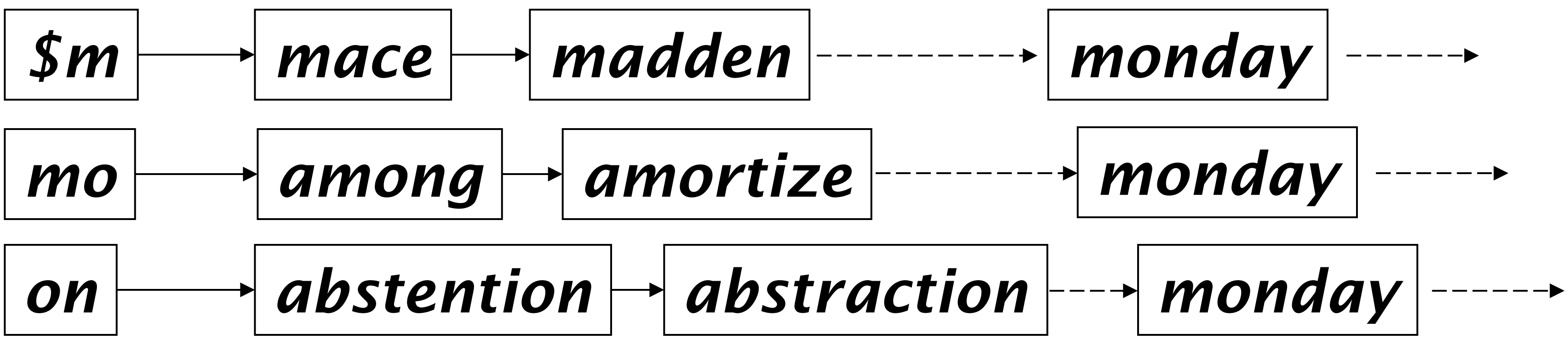
- Enumerate all *k*-grams (sequence of *k* chars) occurring in any term
- *e.g.*, from text "***April is the cruelest month***" we get the 2-grams (*bigrams*)

  > a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru, ue,el,le,es,st,t$, m,mo,on,nt,h

  - $ is a special word boundary symbol
- **Maintain a *<u>second</u>* inverted index *<u>from bigrams to dictionary terms</u>* that match each bigram.**

Bigram index example

- The *k*-gram index finds *terms* based on a query consisting of *k*-grams (here *k*=2)

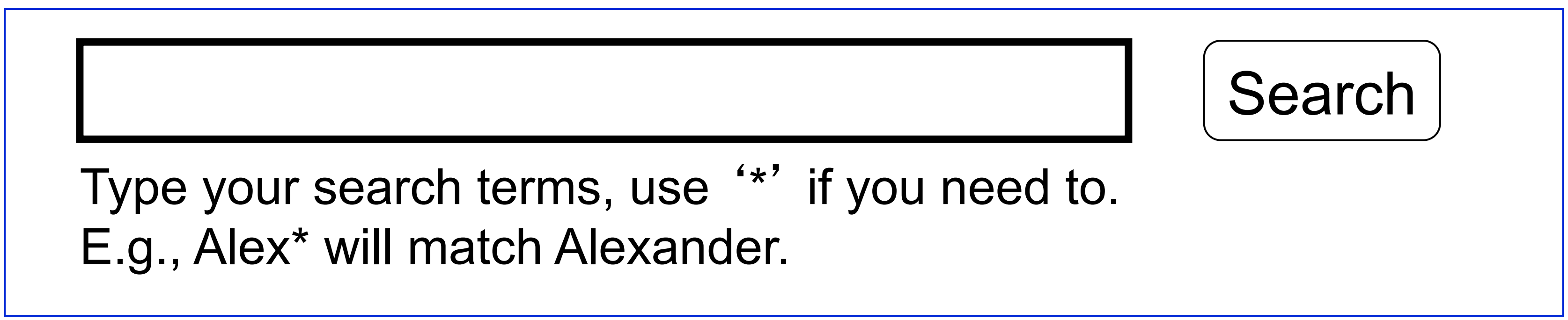


Processing wild-cards

- Query ***mon**** can now be run as
  - ***$m*** *AND* ***mo*** *AND* ***on***
- Gets terms that match all AND conditions - they satisfy our wildcard query (necessary condition)
- But we will get false positive:
  - Eg.: we'd retrieve ***moon*** *(false positive)*
- Must **post-filter** these terms against query
- Surviving enumerated terms are then looked up in the term-document inverted index
- Fast, space efficient (compared to permuterm).

Processing wild-card queries

- As before, we must execute a Boolean query for each enumerated, filtered term
- Wild-cards can result in expensive query execution (very large disjunctions…)
  - pyth* AND prog*
- If you encourage "laziness" people will respond!



- Which web search engines allow wildcard queries? (please double check)

SPELLING CORRECTION

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve “right” answers
- Two main flavors:
  - Isolated word
    - Check each word on its own for misspelling
    - But this will not catch typos resulting in correctly spelled words: e.g., ***from*** → ***form***
  - Context-sensitive
    - Look at surrounding words,
    - e.g., ***I flew form Heathrow to Narita.***

Query mis-spellings

- Our principal focus here
  - E.g., the query ***Alanis Morisett***
- We can either
  - Retrieve documents indexed by "the" correct spelling (Alanis Morisette), OR
  - Return several suggested alternative queries with the correct spelling
    - *Did you mean … ?*
  - One shot vs. Conversational

Isolated word correction

- Fundamental premise – there is a **lexicon** from which the correct spellings come
- Two basic choices for this
  1. A standard lexicon such as
     - Webster's English Dictionary
     - An "industry-specific" lexicon – hand-maintained
  2. The lexicon of the indexed corpus
     - E.g., all words on the web
     - All names, acronyms etc.
     - (Including the mis-spellings in the corpus)

Isolated word correction

- Given a lexicon and a character sequence Q, return the words in the lexicon **closest** to Q
- What's "closest"?
- There are several alternatives (see IIR book)
  - Edit distance (**Levenshtein** distance)
  - Weighted edit distance
  - *n*-gram overlap

Edit distance

- Given two strings *S* and T, the minimum number of operations to convert S (source) into T (target)
- Operations are typically character-level
  - Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from ***dof*** to ***dog*** is 1
  - From ***cat*** to ***act*** is 2 (Just 1 with transpose.)
  - from ***cat*** to ***dog*** is 3.
- Generally found by dynamic programming
- And also http://en.wikipedia.org/wiki/Levenshtein_distance

Weighted edit distance

- The weight of an operation is not constant and depends on the character(s) involved
  - Meant to capture OCR or keyboard errors, e.g. ***m*** more likely to be mis-typed as ***n*** than as ***q***
  - Therefore, replacing ***m*** by ***n*** is a smaller edit distance than by ***q***
  - This may be formulated as a probability model
- Requires weight matrix as input
- Modify dynamic programming to handle weights.

Using edit distances

- Given query:
  - EITHER: first enumerate all character sequences within a preset edit distance (e.g., 2) and then intersect this set with list of "correct" words found in the vocabulary
  - OR: search in the vocabulary the correct words within a preset distance to the query
- Show terms you found to user as suggestions
- Alternatively:
  1. We can look up all possible corrections in our inverted index and return all docs … slow
  2. We can run with a single most likely correction
- These last alternatives **disempower** the user, but save a round of interaction with the user.

Edit distance to all dictionary terms?

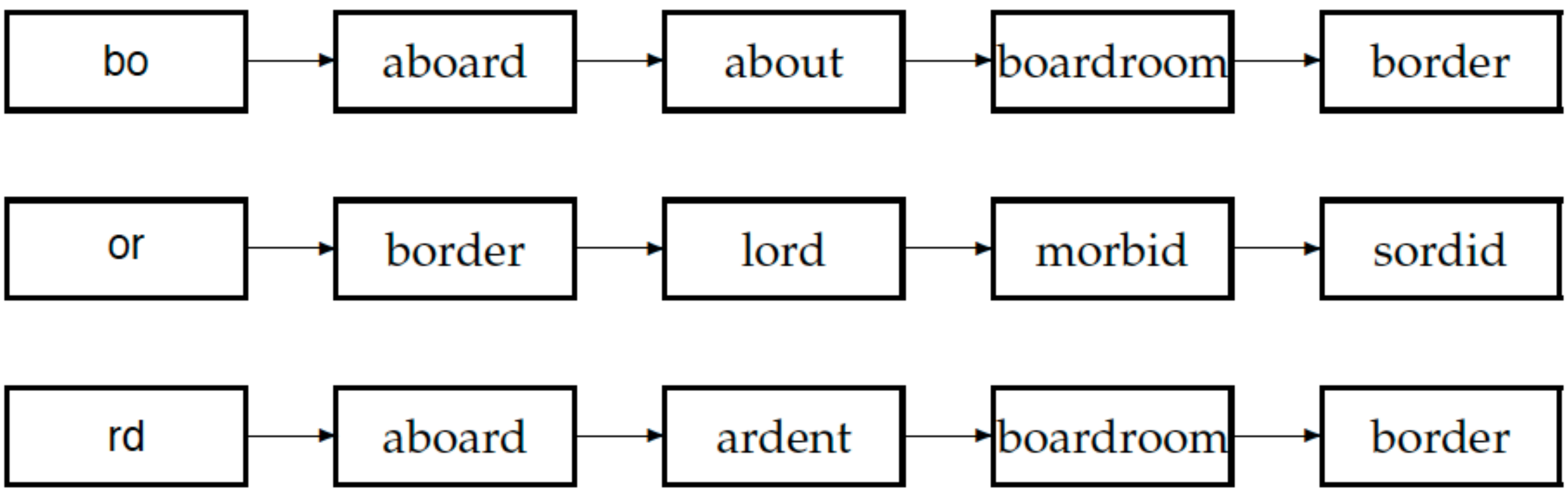
- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
  - Expensive and slow
  - Alternative?
- How do we cut the set of candidate dictionary terms? *Any idea?*
- One possibility is to use *n*-gram overlap for this – because it is faster (provided that you have the n-gram index)
- This can also be used by itself for spelling correction.

n-gram overlap

- Enumerate all the *n*-grams in the query string
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching **any** of the query *n*-grams (*why not* ***all****?*)
- Or consider a threshold by the number of matching *n*-grams (e.g., at least 2 n-grams)
  - Variants – weight by keyboard layout, etc.

Matching 2-grams

- Matching **at least two** 2-grams in the word "bord" will retrieve “aboard”, "border" and “boardroom”
- But “boardroom” is an unlikely correction – has a larger edit distance than "aboard"



► **Figure 3.7** Matching at least two of the three 2-grams in the query bord.

Example with trigrams

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***dicember***
  - Trigrams are *dic, ice, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$\left|X \cap Y\right| / \left|X \cup Y\right|$$

- Equals **1** when *X* and *Y* have the **same elements** and **0** when they are **disjoint**
- *X* and *Y* don't have to be of the same size
- Always assigns a number between 0 and 1
  - Now threshold to decide if you have a match
  - E.g., if J.C. > 0.8, declare a match

Reading Material

- Sections: 2.1, 2.2, 2.4
- Sections: 3.2, 3.3
- Advanced search functionalities in google http://www.google.com/support/websearch/bin/answer.py?answer=136861