Text Pre-processing and Faster Query Processing

David Kauchak

cs160

Fall 2009

adapted from:

http://www.stanford.edu/class/cs276/handouts/lecture2-Dictionary.ppt

Administrative

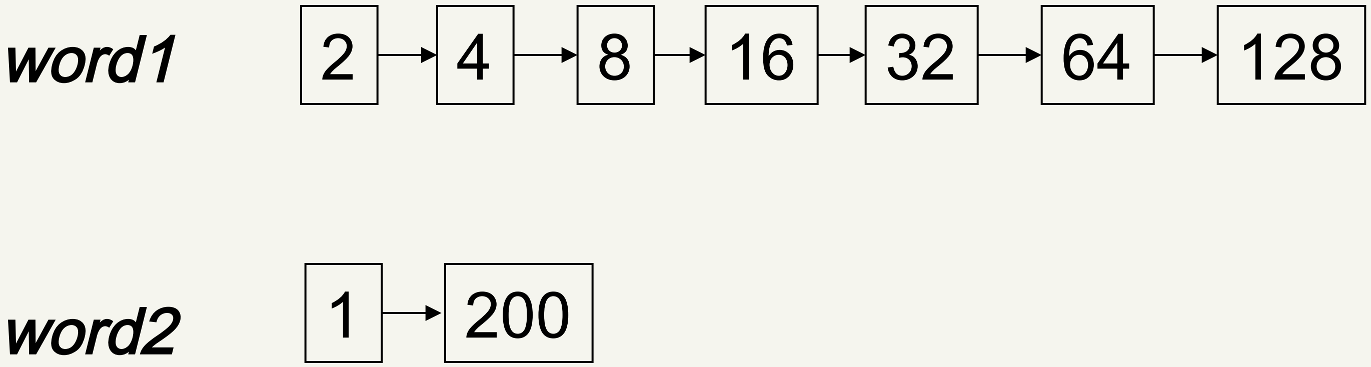
- Everyone have CS lab accounts/access?
- Homework 1
  - Page numbers
  - Due before class next Wed.
  - Popular media question
- Issues with assignment 1?
- Discussion board?
- CS lunch today

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries
- Text pre-processing
  - tokenizing
  - “all but the kitchen sink” - approaches to token normalization
- Regular expressions in Java (time permitting)

Recall the merge

- Walk through the two lists simultaneously



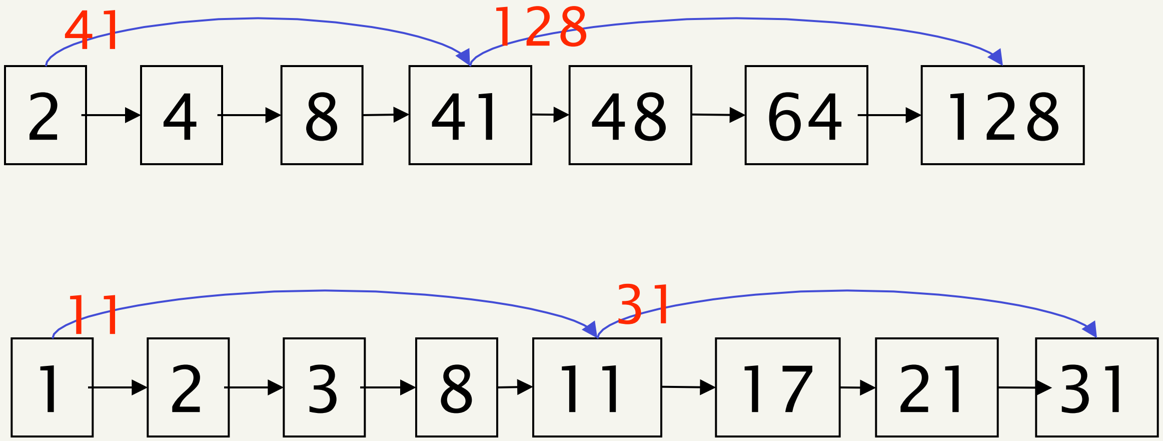
word1 2 → 4 → 8 → 16 → 32 → 64 → 128

word2 1 → 200

O(length1 + length2)

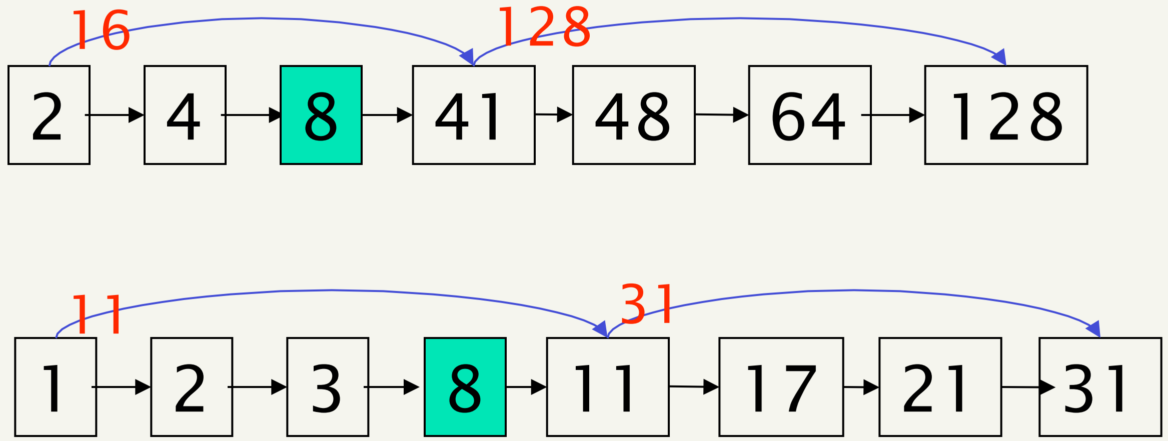
Can we do better?
Can we augment the data structure?

Augment postings with skip pointers (at indexing time)

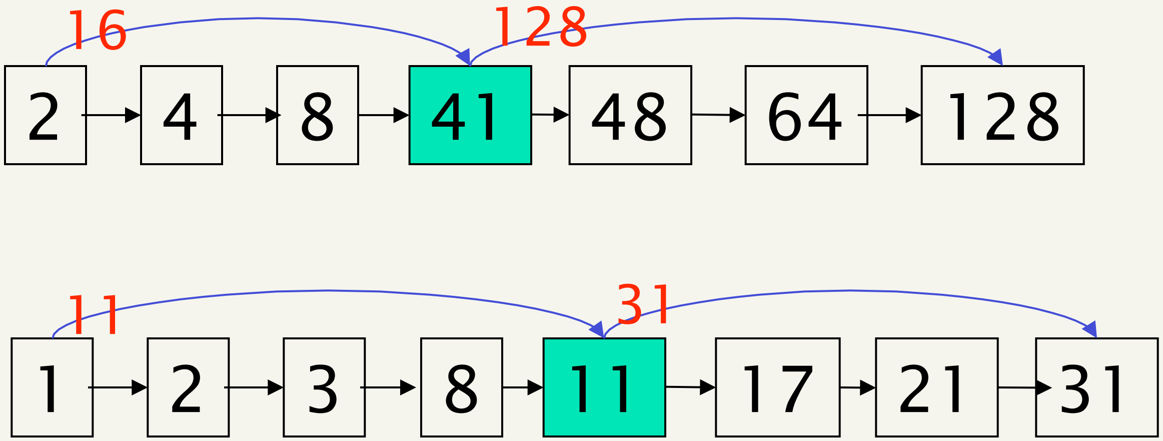


- How does this help?

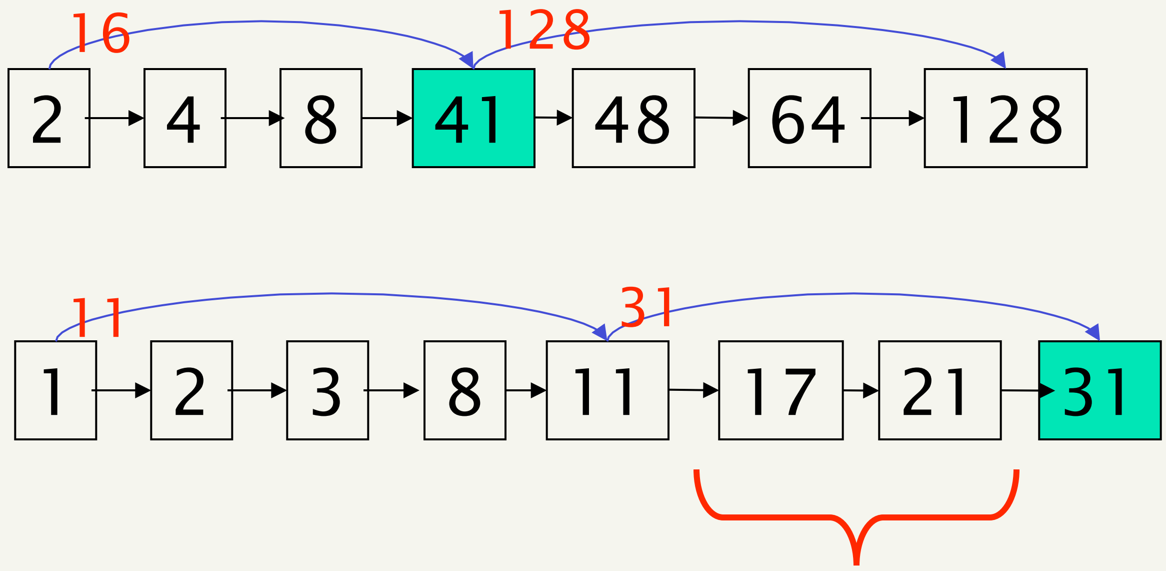
Query processing with skip pointers



Query processing with skip pointers



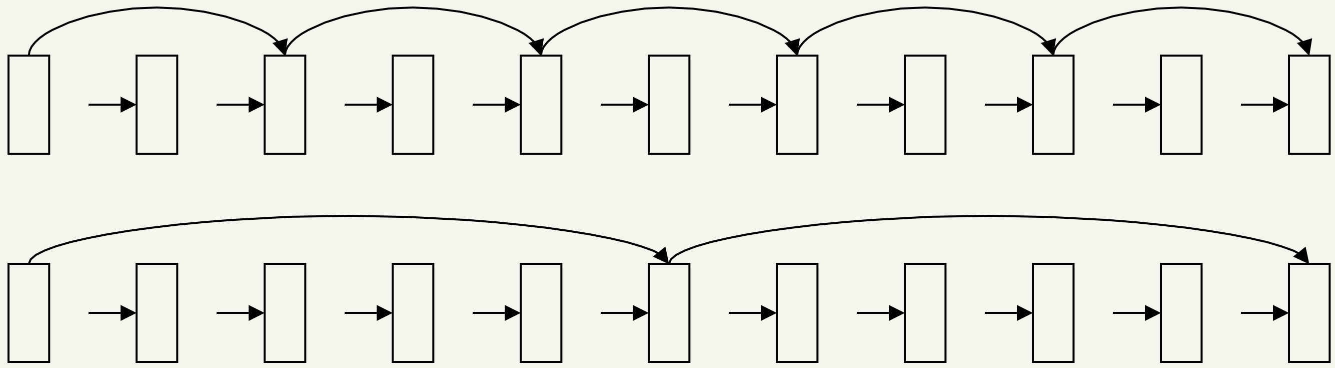
Query processing with skip pointers



we skip these entries

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers. More storage required.
  - 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.
    - ignores word distribution

- Are there any downsides to skip lists?
- The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging! (Bahle et al. 2002)
- A lot of what we'll see in the class are **options**. Depending on the situation some may help, some may not.

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries
- Text pre-processing
  - tokenizing
  - “all but the kitchen sink” - approaches to token normalization
- Regular expressions in Java (time permitting)

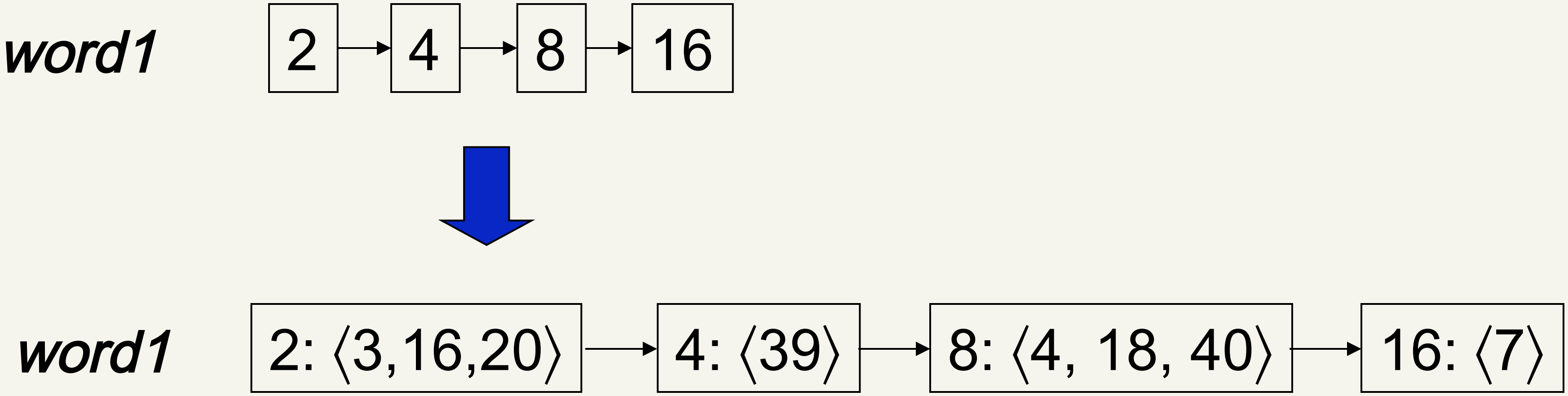
Phrase queries

- Want to be able to answer queries such as "***pomona college***"
- *"I went to a college in pomona"* would not a match
  - The concept of phrase queries has proven easily understood by users
  - Many more queries are *implicit phrase queries*

How can we modify our existing postings lists to support this?

Positional indexes

- In the postings, store a list of the positions in the document where the term occurred



word1 2 → 4 → 8 → 16

word1 2: ⟨3,16,20⟩ → 4: ⟨39⟩ → 8: ⟨4, 18, 40⟩ → 16: ⟨7⟩

docID: ⟨position1,position2,…⟩

Positional index example

be:

1: ⟨7,18,33,72,86,231⟩
2: ⟨3,149⟩
4: ⟨17,191,291,430,434⟩
5: ⟨363, 367⟩

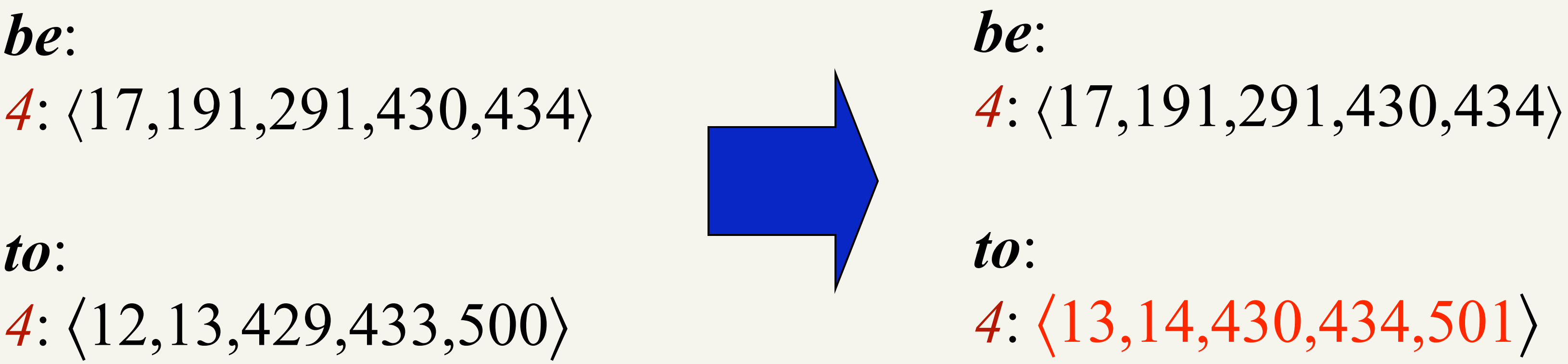
to:

1: ⟨4,17,32, 90⟩
2: ⟨5, 50⟩
4: ⟨12,13,429,433,500⟩
5: ⟨4,15,24,38,366⟩

1. Looking only at the "be" postings list, which document(s) could contain "***to be or not to be***"?
2. Using both postings list, which document(s) could contain "***to be or not to be***"?
3. Describe an algorithm that discovers the answer to question 2 (hint: think about our linear "merge" procedure)

Processing a phrase query: “to be”

- Find all documents that have have the terms using the “merge” procedure
- For each of these documents, “merge” the position lists with the positions offset depending on where in the query the word occurs



be:
4: ⟨17,191,291,430,434⟩

to:
4: ⟨12,13,429,433,500⟩

be:
4: ⟨17,191,291,430,434⟩

to:
4: ⟨13,14,430,434,501⟩

Processing a phrase query: “to be”

- Find all documents that have have the terms using the “merge” procedure
- For each of these documents, “merge” the position lists with the positions offset depending on where in the query the word occurs

be:

4: ⟨17,191,291,430,434⟩

to:

4: ⟨12,13,429,433,500⟩

be:

4: ⟨17,191,291,430,434⟩

to:

4: ⟨13,14,430,434,501⟩

What about proximity queries?

- Find “pomona” within k words of “college”
- Similar idea, but a bit more challenging

- Naïve algorithm for merging position lists
  - Assume we have access to a merge with offset exactly i procedure (similar to phrase query matching)
  - for i = 1 to k
    - if merge with offset i matches, return a match
    - if merge with offset -i matches, return a match

- Naïve algorithm is inefficient, but doing it efficiently is a bit tricky

Positional index size

- You can compress position values/offsets
- 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

- What does adding positional information do to the size of our index?
- Need an entry for each occurrence, not just once per document
- Posting size depends on the lengths of the documents

Positional index size

- 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	?	
100,000		

Positional index size

- 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	
100,000	?	

Positional index size

- 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	?
100,000	1	

Positional index size

- 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	?

Positional index size

- 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

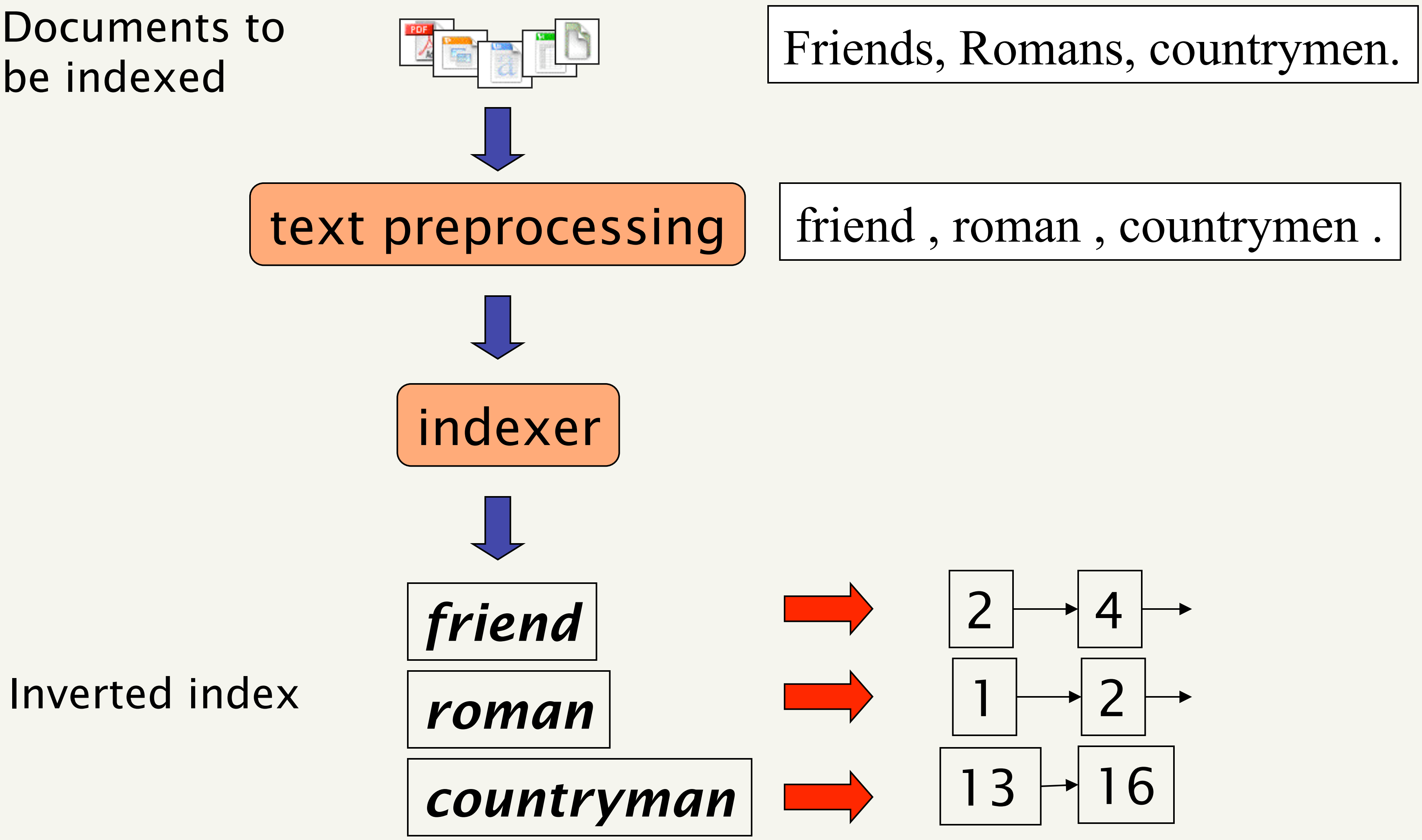
Popular phrases

- Is there a way we could speed up common/ popular phrase queries?
  - ***"Michael Jackson"***
  - ***"Britney Spears"***
  - ***"New York"***
- We can store the phrase as another *term* in our dictionary with it's own postings list
- This avoids having do do the "merge" operation for these frequent phrases

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries
- Text pre-processing
  - tokenizing
  - “all but the kitchen sink” - approaches to token normalization
- Regular expressions in Java (time permitting)

Inverted index construction



Documents to be indexed

Friends, Romans, countrymen.

text preprocessing

friend , roman , countrymen .

indexer

Inverted index

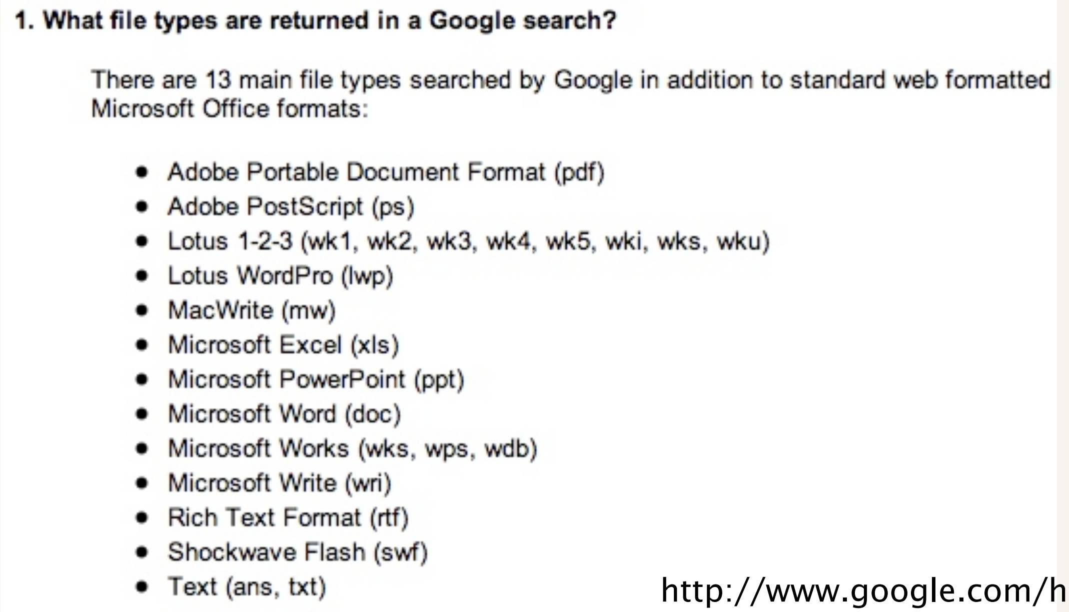
friend → 2 → 4 →

roman → 1 → 2 →

countryman → 13 → 16

What’s in a document?

- I give you a file I downloaded
- You know it has text in it
- What are the challenges in determining what characters are in the document?
    - File format:



1. What file types are returned in a Google search?

There are 13 main file types searched by Google in addition to standard web formatted Microsoft Office formats:

- Adobe Portable Document Format (pdf)
- Adobe PostScript (ps)
- Lotus 1-2-3 (wk1, wk2, wk3, wk4, wk5, wki, wks, wku)
- Lotus WordPro (lwp)
- MacWrite (mw)
- Microsoft Excel (xls)
- Microsoft PowerPoint (ppt)
- Microsoft Word (doc)
- Microsoft Works (wks, wps, wdb)
- Microsoft Write (wri)
- Rich Text Format (rtf)
- Shockwave Flash (swf)
- Text (ans, txt)

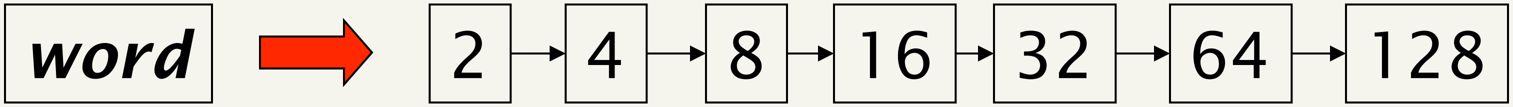
http://www.google.com/help/faq_filetypes.html

What’s in a document?

- I give you a file I downloaded
- You know it has text in it
- What are the challenges in determining what characters are in the document?
  - Language:
    - 莎, Δ, Tübingen, …
    - Sometimes, a document can contain multiple languages (like this one :)
  - Character set/encoding
    - UTF-8
    - How do we go from the binary to the characters?
  - Decoding
    - zipped/compressed file
    - character entities, e.g. ‘&nbsp;’

What is a “document”?

- A postings list is a list of documents

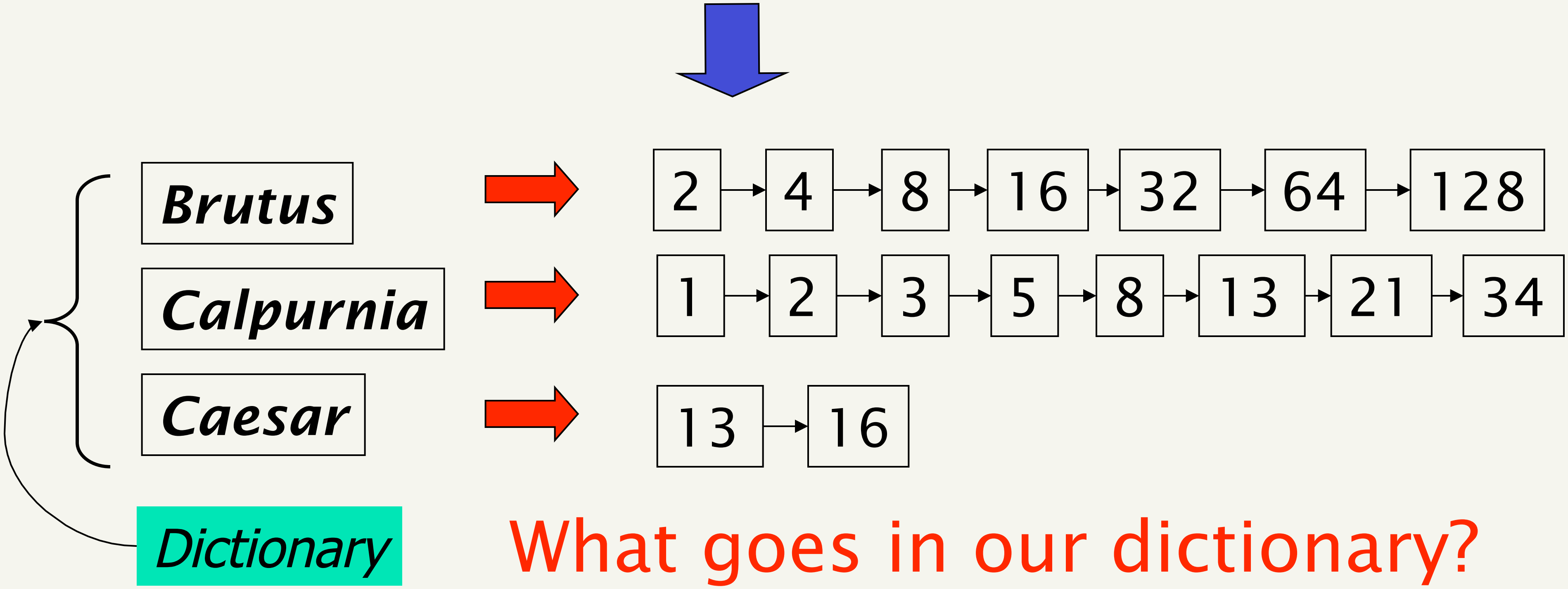


- What about:
  - a web page
  - a book
  - a report/article with multiple sections
  - an e-mail
  - an e-mail with attachments
  - a powerpoint file
  - an xml document
- What amount of text is considered a “document” for these lists?

Text pre-processing

- Assume we’ve figured all of this out and we now have a stream of characters that is our document

“***Friends, Romans, Countrymen*** …”



Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

Calpurnia → 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

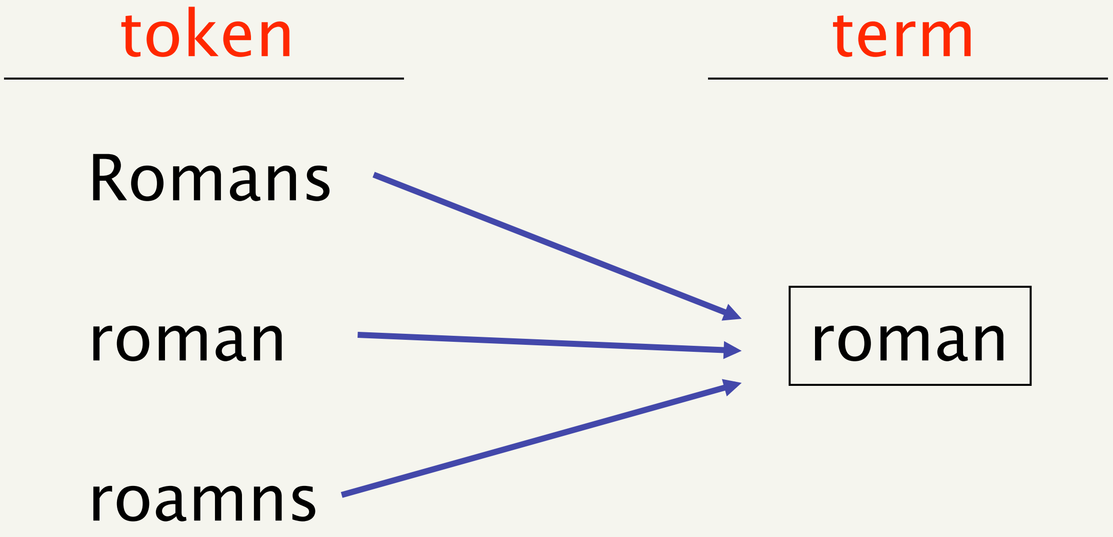
Caesar → 13 → 16

Dictionary

What goes in our dictionary?

Text pre-processing

- A *token* is a sequence of characters that are grouped together as a semantic unit
- A *term* is an entry in the dictionary
- Multiple tokens may map to the same term:



token

term

Romans

roman

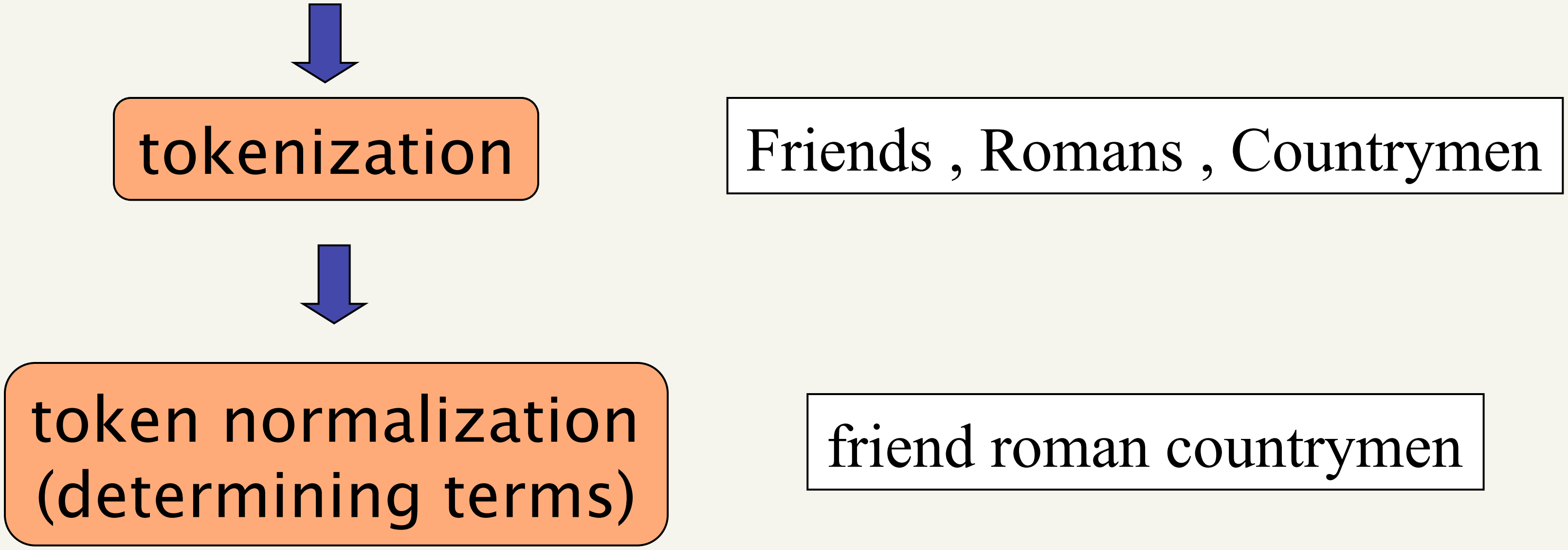
roamns

roman

Text pre-processing

- Determining the *tokens* and *terms* are the two major pre-processing steps

"***Friends, Romans and Countrymen** …*"



tokenization

Friends , Romans , Countrymen

token normalization (determining terms)

friend roman countrymen

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries
- Text pre-processing
  - tokenizing
  - "all but the kitchen sink" - approaches to token normalization
- Regular expressions in Java (time permitting)

Basic tokenization

- If I asked you to break a text into tokens, what might you try?

  - Split tokens on whitespace
  - Split or throw away punctuation characters

Tokenization issues: ‘

Finland’s capital…

?

Tokenization issues: ‘

***Finland’s** **capital**…*

Finland

Finland ‘ s

Finland ‘s

Finlands

Finland s

Finland’s

What are the benefits/drawbacks?

Tokenization issues: ‘

Aren’t we …

?

Tokenization issues: ‘

Aren’t we …

Aren’t

Arent

Are n’t

Aren t

Tokenization issues: hyphens

Hewlett-Packard

state-of-the-art

co-education

lower-case

?

Tokenization issues: hyphens

Hewlett-Packard ***state-of-the-art***

co-education ***lower-case***

- Keep as is
- merge together
  - HewlettPackard
  - stateoftheart
- Split on hyphen
  - lower case
  - co education

What are the benefits/drawbacks?

More tokenization issues

- Compound nouns: San Francisco, Los Angelos, …
  - One token or two?
- Numbers
  - Examples
    - Dates: 3/12/91
    - Model numbers: B-52
    - Domain specific numbers: PGP key - 324a3df234cb23e
    - Phone numbers: (800) 234-2333
    - Scientific notation: 1.456 e-10

Tokenization: language issues

Lebensversicherungsgesellschaftsangestellter

‘life insurance company employee’

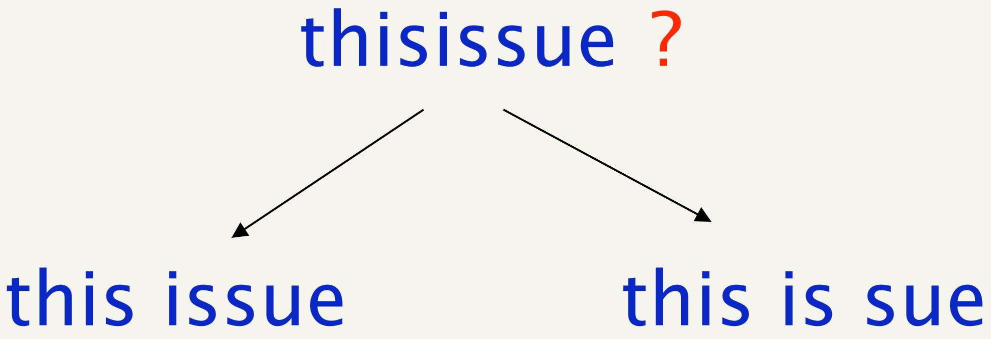
- Opposite problem we saw with English (San Francisco)
- German compound nouns are not segmented
- German retrieval systems frequently use a **compound splitter** module

Tokenization: language issues

莎拉波娃现在居住在美国东南部的佛罗里达。

Where are the words?

- Chinese and Japanese have no spaces between words
  - A word can be made up of one or more characters
  - There is ambiguity about the tokenization, i.e. more than one way to break the characters into words
  - Word segmentation problem



thisissue ?

this issue

this is sue

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries
- Text pre-processing
  - tokenizing
  - "all but the kitchen sink" - approaches to token normalization
- Regular expressions in Java (time permitting)

Token normalization/ Dictionary construction

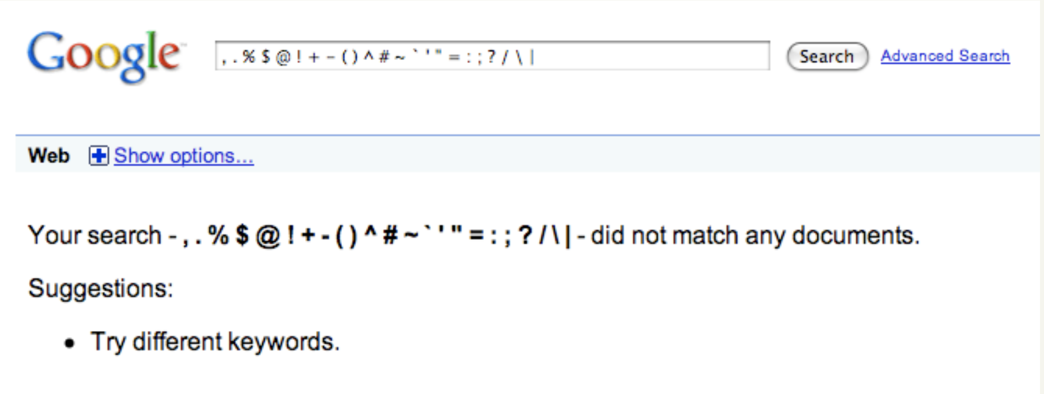
- We now have the documents as a stream of tokens

Friends , Romans , Countrymen

- We have two decisions to make:
  - Are we going to keep all of the tokens?
    - punctuation?
    - common words, “to”, “the”, “a”
  - What will be our *terms*, i.e. our dictionary entries
    - Determine a mapping from *tokens* to *terms*

Punctuation characters

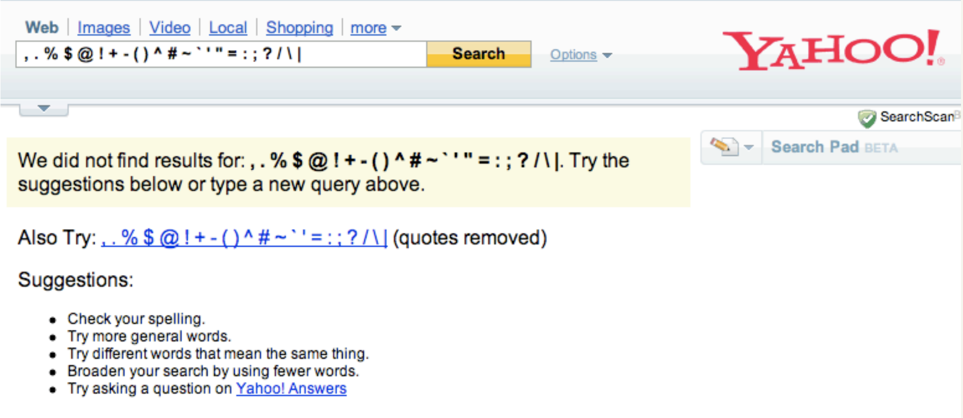
- Most search engines do not index most punctuation characters: , . % $ @ ! + - ( ) ^ # ~ ` ' " = : ; ? / \ |



Your search - , . % $ @ ! + - ( ) ^ # ~ ` ' " = : ; ? / \ | - did not match any documents.

Suggestions:

- Try different keywords.

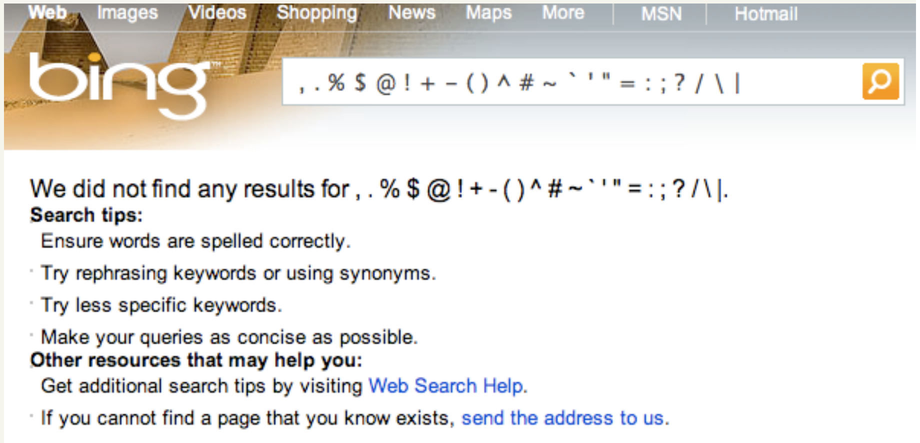


We did not find results for: , . % $ @ ! + - ( ) ^ # ~ ` ' " = : ; ? / \ |. Try the suggestions below or type a new query above.

Also Try: , . % $ @ ! + - ( ) ^ # ~ ` ' = : ; ? / \ | (quotes removed)

Suggestions:

- Check your spelling.
- Try more general words.
- Try different words that mean the same thing.
- Broaden your search by using fewer words.
- Try asking a question on Yahoo! Answers



We did not find any results for , . % $ @ ! + - ( ) ^ # ~ ` ' " = : ; ? / \ |.

Search tips:

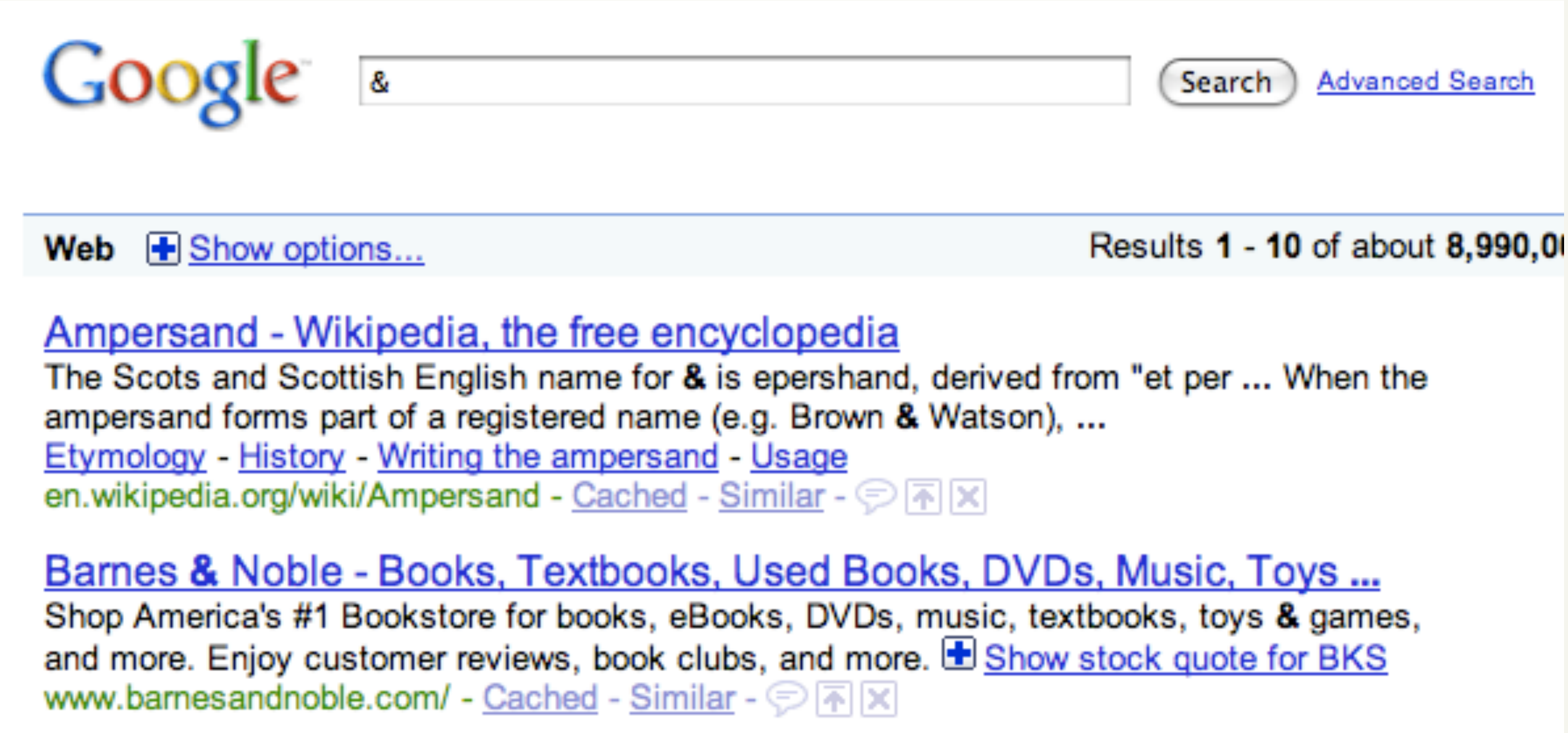
- Ensure words are spelled correctly.
- Try rephrasing keywords or using synonyms.
- Try less specific keywords.
- Make your queries as concise as possible.

Other resources that may help you:

- Get additional search tips by visiting Web Search Help.
- If you cannot find a page that you know exists, send the address to us.

Punctuation characters

- Although there are sometimes exceptions…



&

Results **1** - **10** of about **8,990,0**

Ampersand - Wikipedia, the free encyclopedia
The Scots and Scottish English name for **&** is epershand, derived from "et per ... When the ampersand forms part of a registered name (e.g. Brown **&** Watson), ...
Etymology - History - Writing the ampersand - Usage
en.wikipedia.org/wiki/Ampersand - Cached - Similar -

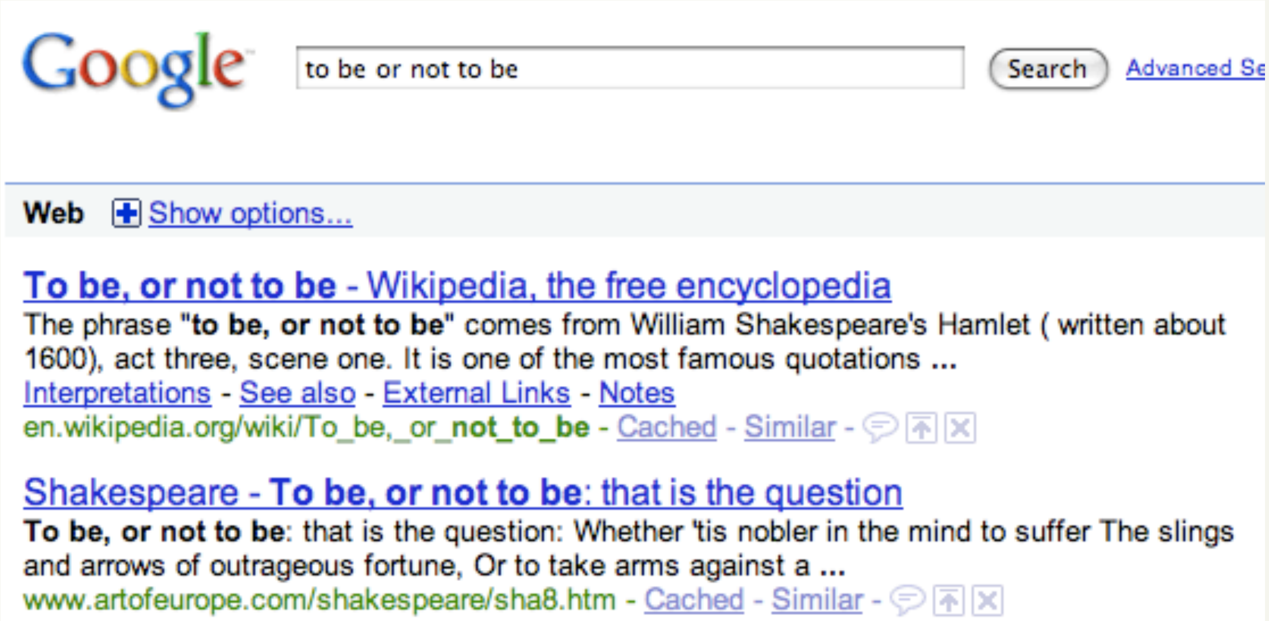
Barnes & Noble - Books, Textbooks, Used Books, DVDs, Music, Toys ...
Shop America's #1 Bookstore for books, eBooks, DVDs, music, textbooks, toys **&** games, and more. Enjoy customer reviews, book clubs, and more. Show stock quote for BKS
www.barnesandnoble.com/ - Cached - Similar -

Stop words

- With a stop list, you exclude from the index/ dictionary the most common words
- Pros:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
- Cons
  - Phrase queries: “King of Denmark”
  - Song titles, etc.: “Let it be”, “To be or not to be”
  - “Relational” queries: “flights to London”

Stop words

- The trend for search engines is to **not** use stop lists
  - Good compression techniques mean 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



To be, or not to be - Wikipedia, the free encyclopedia
The phrase "**to be, or not to be**" comes from William Shakespeare's Hamlet (written about 1600), act three, scene one. It is one of the most famous quotations ...
Interpretations - See also - External Links - Notes
en.wikipedia.org/wiki/To_be,_or_**not_to_be** - Cached - Similar -

Shakespeare - **To be, or not to be**: that is the question
To be, or not to be: that is the question: Whether 'tis nobler in the mind to suffer The slings and arrows of outrageous fortune, Or to take arms against a ...
www.artofeurope.com/shakespeare/sha8.htm - Cached - Similar -

Token normalization

- Want to find a many to one mapping from tokens to terms
- Pros
  - smaller dictionary size
  - increased recall (number of documents returned)
- Cons
  - decrease in specificity, e.g. can't differentiate between plural non-plural
  - exact quotes
  - decrease in precision (match documents that aren't relevant)

Two approaches to normalization

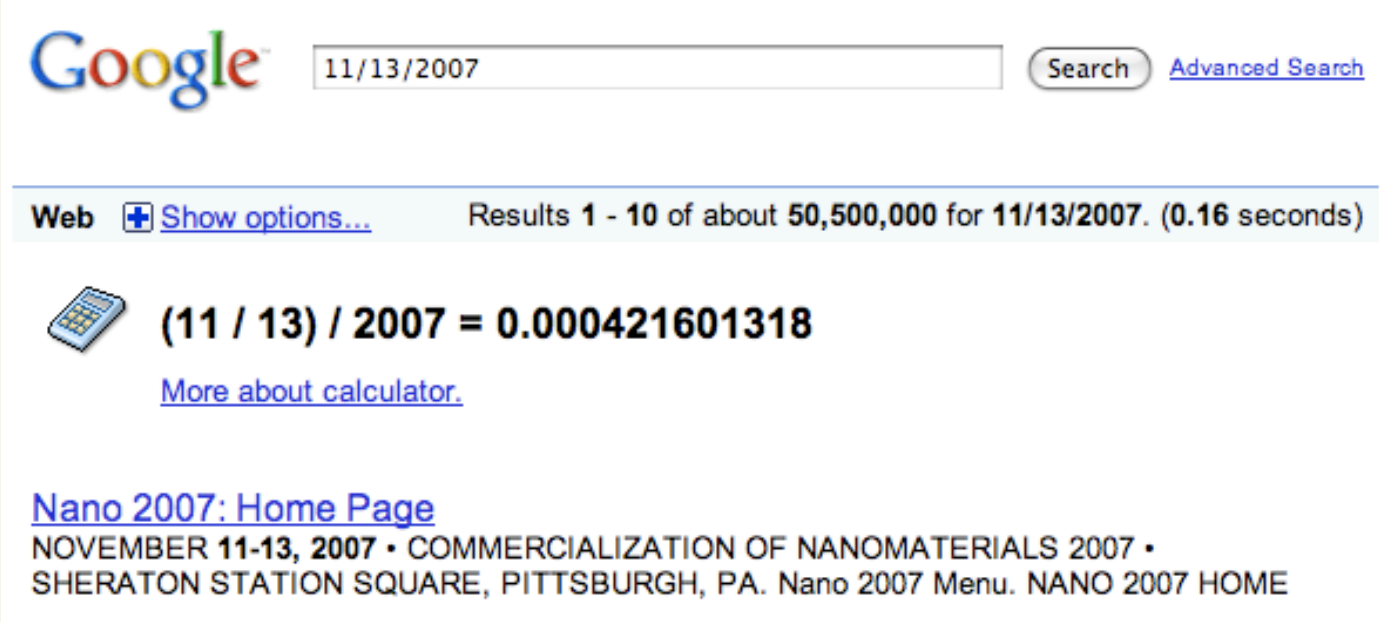
- Implicitly define equivalence classes of terms by performing operations on tokens
  - deleting periods in a term
  - removing trailing letters (e.g. ‘s’)
- Alternative is to do expansion. Start with a list of terms and expand to possible tokens
  - window → Window, Windows, window, windows
  - Potentially more powerful, but less efficient

Token normalization

- Abbreviations - remove periods
  - I.B.M. → IBM
  - N.S.A. → N.S.A
  - Aug 2005 Google example: C.A.T. → Cat Fanciers website *not* Caterpiller Inc.
- Numbers
  - Keep (try typing random numbers into a search engine)
  - Remove: can be very useful: think about things like looking up error codes/stacktraces on the web
  - Identify types, like date, IP, …
  - Flag as a generic “number”

Token normalization

- Dates
  - 11/13/2007
  - 13/11/2007
  - November 13, 2007
  - Nov. 13, 2007
  - Nov 13 '07



Results **1** - **10** of about **50,500,000** for **11/13/2007**. (**0.16** seconds)

(11 / 13) / 2007 = 0.000421601318

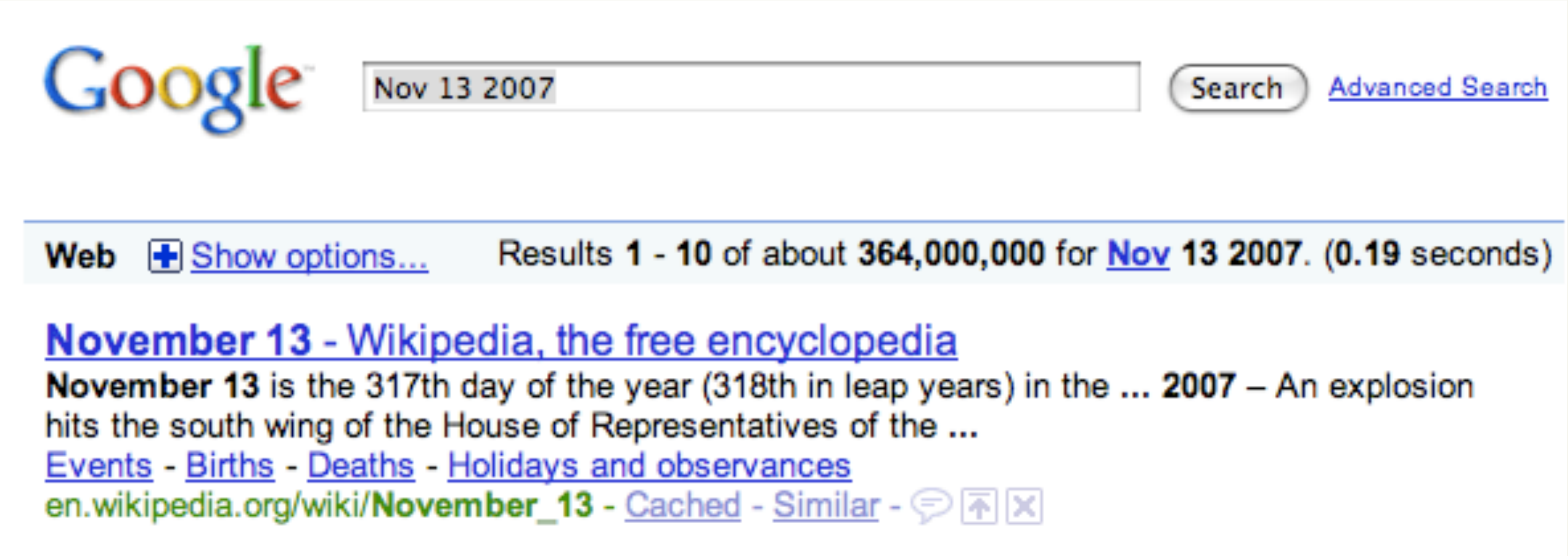
More about calculator.

Nano 2007: Home Page

NOVEMBER **11-13, 2007** • COMMERCIALIZATION OF NANOMATERIALS 2007 • SHERATON STATION SQUARE, PITTSBURGH, PA. Nano 2007 Menu. NANO 2007 HOME

Token normalization

- Dates
  - 11/13/2007
  - 13/11/2007
  - November 13, 2007
  - Nov. 13, 2007
  - Nov 13 ‘07



Results **1** - **10** of about **364,000,000** for Nov **13 2007**. (**0.19** seconds)

November 13 - Wikipedia, the free encyclopedia

November 13 is the 317th day of the year (318th in leap years) in the ... **2007** – An explosion hits the south wing of the House of Representatives of the ...

Events - Births - Deaths - Holidays and observances

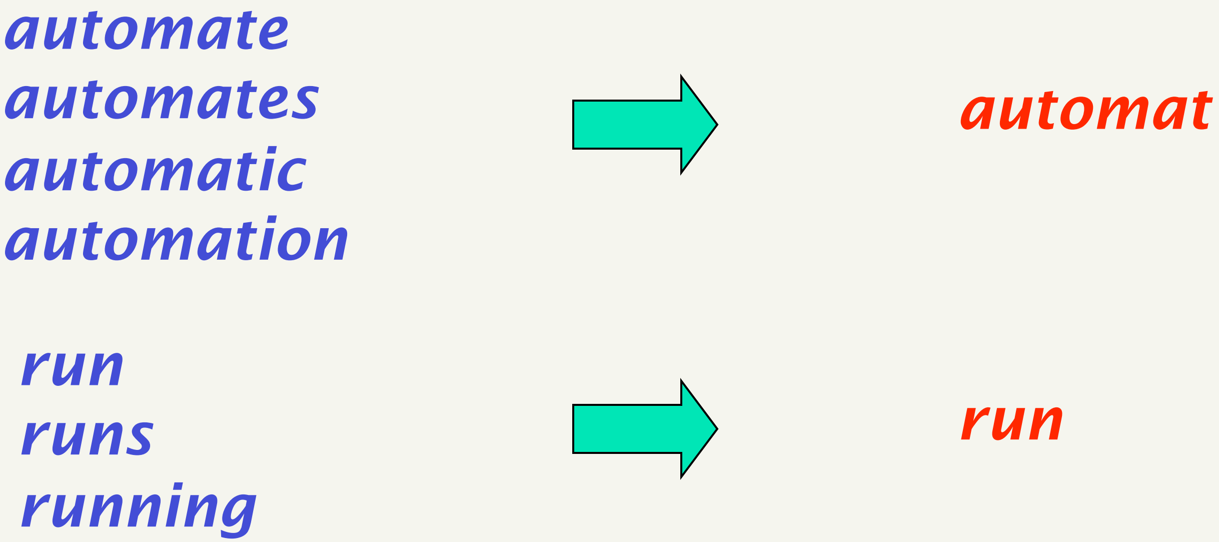
en.wikipedia.org/wiki/**November_13** - Cached - Similar -

Token normalization: lowercasing

- Reduce all letters to lowercase
  - "New policies in …" → "new policies in …"
- Any problems with this?
  - Can change the meaning
    - Sue vs. sue
    - Fed vs. fed
    - SAIL vs. sail
    - CAT vs. cat
- Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

Stemming

- Reduce terms to their "roots" before indexing
- The term "stemming" is used since it is accomplished mostly by chopping off part of the suffix of the word



automate
automates
automatic
automation
→ *automat*

run
runs
running
→ *run*

Stemming example

Taking a course in information retrieval is more exciting than most courses

`Take a cours in inform retriev is more excit than most cours`

http://maya.cs.depaul.edu/~classes/ds575/porter.html
or use the class from hw1 to try some examples out

Porter’s algorithm (1980)

- Most common algorithm for stemming English
  - Results suggest it’s at least as good as other stemming options
- Multiple sequential phases of reductions using rules, e.g.
  - sses → ss
  - ies → i
  - ational → ate
  - tional → tion
- http://tartarus.org/~martin/PorterStemmer/

Lemmatization

- Reduce inflectional/variant forms to base form
- Stemming is an *approximation* for lemmatization
- Lemmatization implies doing “proper” reduction to dictionary headword 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

What normalization techniques to use…

- What is the size of the corpus?
  - small corpora often require more normalization
- Depends on the users and the queries
- Query suggestion (i.e. “did you mean”) can often be used instead of normalization
- Most major search engines do little to normalize data except lowercasing and removing punctuation (and not even these always)

Outline for today

- Improvements to basic postings lists
  - Speeding up the merge operation
  - Adding phrase queries and proximity queries

- Text pre-processing
  - tokenizing
  - "all but the kitchen sink" - approaches to token normalization

- Regular expressions in Java (time permitting)

Regular expressions

- Regular expressions are a very powerful tool to do string matching and processing
- Allows you to do things like:
  - Tell me if a string starts with a lowercase letter, then is followed by 2 numbers and ends with “ing” or “ion”
  - Replace all occurrences of one or more spaces with a single space
  - Split up a string based on whitespace or periods or commas or …
  - Give me all parts of the string where a digit is proceeded by a letter and then the ‘#’ sign

A quick review of regex features

- Literals: we can put any string in regular expression
  - “this is a test”.matches(“test”)
  - “this is a test”.matches(“hmm”)
- Meta-characters
  - \w - word character (a-zA-Z_0-9)
  - \W - non word-character (i.e. everything else)
  - \d - digit (0-9)
  - \s - whitespace character (space, tab, endline, …)
  - \S - non-whitespace
  - . - matches any character

regex features

- Metacharacters
  - “The year was 1988”.matches(“19\d\d”)
  - “Therearenospaceshere”.matches(“\s”)
- Java and ‘\’ - annoyingly, need to escape the backslash
  - “The year was 1988”.matches(“19\\d\\d”)
  - “Therearenospaceshere”.matches(“\\s”)

more regex features

- Character classes
  - [aeiou] - matches any vowel
  - [^aeiou] - matches anything BUT the vowels
  - [a-z] - all lowercase letters
  - [0-46-9]
  - "The year was 1988".matches("[12]\d\d\d")
- Special characters
  - '^' matches the beginning of the string
    - "^\d"
    - "^The"

More regex features

- Special characters
  - ‘$’ matches the end of the string
    - “Problem 1 - 5 points:”. matches(“^Problem \d - \d points$”)
    - “Problem 1 - 8 points”. matches(“^Problem \d - \d points$”)
- Quantifiers
  - * - zero or more times
  - + - 1 or more times
  - ? - once or not at all
  - “^\d+”
  - “[A-Z][a-z]*”
  - “Runners?”

Regex in java

- java.util.regex.*
  - Patterns
  - Matcher
- For any string:
  - string.matches(regex) - returns true if the string matches the pattern (remember, if it doesn't have '^' or '$' than it can match **part** of the string)
  - string.split(regex) - split up the string where the delimiter is all matches of the expression
  - string.replaceAll(regex, replace) - replace all matches of "regex" with "replace"
- LOTS of resources out there!
  - http://java.sun.com/docs/books/tutorial/essential/regex/intro.html
  - http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html

Resources for today's lecture

- IIR 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.