Logistics/Updates

• Project 1 details posted. For now: Due Feb 12, may push back a few days.
• Please use Piazza!
Lecture Plan

- Building an Index (continued)
  - Document processing
  - Tokenization
  - Term selection → Dictionary
  - Phrase queries
  - If time: One basic index optimization
Recap: The Search Process

1. Source Selection
2. Query Formulation
3. Search
4. Selection
5. Examination
6. Delivery
First Part of Course: the IR “Black Box”

- Study the IR black box in isolation
  - Input: Query
  - Output: Ranked list of documents
  - Goal: optimize the list of output documents
The IR Black Box

Query

Documents

Results
Inside The IR Black Box

Query

representation function

query representation

comparison function

hits

Documents

representation function

document representation

Index

1/20/2016 7CS 572: Information Retrieval. Spring 2016
Basic Inverted Indexes, Boolean Queries

• Basic inverted indexes:
  – Structure: Dictionary and Postings

\[
\begin{array}{c}
\text{Brutus} & \rightarrow & 1 & 2 & 4 & 11 & 31 & 45 & 173 & 174 \\
\text{Caesar} & \rightarrow & 1 & 2 & 4 & 5 & 6 & 16 & 57 & 132 & \ldots \\
\text{Calpurnia} & \rightarrow & 2 & 31 & 54 & 101 \\
\end{array}
\]

  – Key step in construction: Sorting

• Boolean query processing
  – Intersection by linear time “merging”
  – Simple optimizations
This lecture: What Goes Into the Index?

- Processing to form the term vocabulary
  - Documents
  - Tokenization
  - What *terms* do we put in the index?
- Dictionary Implementation
- Postings (basic optimization)
  - If time: Faster merges: skip lists
  - Positional postings, phrase queries
Recall the basic indexing pipeline

Documents to be indexed.

Token stream.

Modified tokens \(\rightarrow\) Terms

Inverted index.
Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem → later in the course.

Often done heuristically for important special cases

What about multimedia files (docs w/ images or video?)
Complications: Format and 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 an inbox.)
  – An email with 5 attachments?
  – A group of files (PPT or LaTeX as HTML pages)
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*?
    • It can be effective to get the user to put in possible hyphens
  – *San Francisco*: one token or two?
    • How do you decide if it should be one token?
Numbers

- 55 B.C.
- B-52
- My PGP key is 324a3df234cb23e
- (800) 234-2333
  - Often have embedded spaces
  - Older IR systems may not index numbers
    - But often very useful: think about things like looking up error codes/stacktraces on the web
    - (One answer is using n-grams: next lecture)
  - 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 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
    – Can give a 15% performance boost for German
Tokenization: language issues

• Chinese and Japanese languages 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

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

• ‘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
Strings and Segments

• Remember: user wants search for **concepts**
  – But what we actually search for are **character strings**

• What strings best represent concepts?
  – In English, words are often a good choice
    • Well-chosen phrases might also be helpful
  – In German, compounds may need to be split
    • Otherwise queries using constituent words would fail
  – In Chinese, word boundaries are not marked
    • This segmentation problem is similar to that of speech
Segmentation Details

• Words (from linguistics):
  – Morphemes are the units of meaning
  – Combined to make words
    • Anti (disestablishmentarian) ism

• Tokens (Java String Tokenizer version):
  – Eugene’s teaching ir - class.
Longest Substring Segmentation

• Greedy algorithm based on a lexicon

• Start with a list of every possible term

• For each unsegmented string
  – Remove the longest single substring in the list
  – Repeat until no substrings are found in the list

• Can be extended to explore alternatives
Longest Substring Example

- Possible German compound term:
  - washington

- List of German words:
  - ach, hin, hing, sei, ton, was, wasch

- Longest substring segmentation
  - was-hing-ton
  - Roughly translates as “What tone is attached?”
Probabilistic Segmentation

- For an input word $c_1 c_2 c_3 ... c_n$
- Try all possible partitions into $w_1 w_2 w_3 ...$
  - $c_1 c_2 c_3 ... c_n$
  - $c_1 c_2 c_3 c_3 ... c_n$
  - $c_1 c_2 c_3 ... c_n$ etc.
- Choose the highest probability partition
  - E.g., compute $Pr(w_1 w_2 w_3)$. How? Details soon.
- Challenges: lookup, probability estimation
Alternative: N-gram Indexing

• Consider a Chinese document  $c_1 c_2 c_3 \ldots c_n$

• **Don’t segment** (you could be wrong!)

• Instead, treat *every character bigram* as a term

  $c_1 c_2, c_2 c_3, c_3 c_4, \ldots, c_{n-1} c_n$

• Break up queries the same way

More on this later. Think why else might be useful or harmful.
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 terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary

- We most commonly implicitly define equivalence classes of terms by, e.g.,
  - deleting periods to form a term
    - **U.S.A.**, **USA** \(\rightarrow\) **USA**
  - deleting hyphens to form a term
    - **anti-discriminatory**, **antidiscriminatory** \(\rightarrow\) **antidiscriminatory**
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 normally have accents, users often may not type them
  – Often best to normalize to a de-accented term
    • Tuebingen, Tübingen, Tubingen \( \rightarrow \) 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

• 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...

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

• Inflectional morphology
  – Preserves part of speech
  – Destructions = Destruction+PLURAL
  – Destroyed = Destroy+PAST

• Derivational morphology
  – Relates parts of speech
  – Destructor = AGENTIVE(destroy)
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 headword form
Stemming

• Conflates words, usually preserving meaning
  – Rule-based suffix-stripping helps for English
    • \{destroy, destroyed, destruction\}: \textit{destr}
  – Prefix-stripping is needed in some languages
    • Arabic: \{alselam\}: \textit{selam} [Root: SLM (peace)]

• Imperfect: goal is to \underline{usually} be helpful
  – Overstemming
    • \{centennial, century, center\}: \textit{cent}
  – Understamming:
    • \{acquire, acquiring, acquired\}: \textit{acquir}
    • \{acquisition\}: \textit{acquis}
Porter’s algorithm

- Commonest algorithm for stemming English
  - Results 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**

- **Weight of word sensitive rules**
  - \((m>1)\) **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)

• 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, Russian…
    • 30% performance gains for Finnish!
Relating Words and Concepts

- **Homonymy:** *bank* (river) vs. *bank* (financial)
  - *Different* words are written the same way
  - We’d like to work with word **senses** rather than words

- **Polysemy:** *fly* (pilot) vs. *fly* (passenger)
  - A word can have different “shades of meaning”
  - Not bad for IR: often helps more than it hurts

- **Synonymy:** *class* vs. *course*
  - Causes search failures ... well address this next week!
Thesauri and soundex

• Do we handle synonyms and homonyms?
  – E.g., by hand-constructed equivalence classes
    • \textit{car} = \textit{automobile} \quad \textit{color} = \textit{colour}
  – We can rewrite to form equivalence-class terms
    • When the document contains \textit{automobile}, index it under \textit{car-automobile} (and vice-versa)
  – Or we can expand a query
    • When the query contains \textit{automobile}, look under \textit{car} as well

• What about spelling mistakes?
  – One approach is soundex, which forms equivalence classes of words based on phonetic heuristics

• More in next lecture
Word Sense Disambiguation

• Context provides clues to word meaning
  – “The doctor removed the appendix.”
• For each occurrence, note surrounding words
  – e.g., +/- 5 non-stopwords
• Group similar contexts into clusters
  – Based on overlaps in the words that they contain
• Separate clusters represent different senses
Disambiguation Example

- Consider four example sentences
  - The doctor removed the appendix
  - The appendix was incomprehensible
  - The doctor examined the appendix
  - The appendix was removed

- What clues can you find from nearby words?
  - Can you find enough word senses this way?
  - Might you find too many word senses?
  - What will you do when you aren’t sure?
Disambiguation Can Hurt

- Disambiguation tries to reduce incorrect matches
  - But errors can also reduce correct matches

- Ranked retrieval techniques already disambiguate
  - When more query terms are present, documents rank higher
  - Essentially, queries give each term a context
Phrases

• Phrases can yield more precise queries
  – “University of Maryland”, “solar eclipse”

• Automated phrase detection can be harmful
  – Bad choices result in missed matches
  – Therefore, we never index only phrases
    • Better to index phrases and their constituent words
  – IR systems are good at evidence combination
    • Better evidence combination ⇒ less help from phrases

• Parsing is still relatively slow and brittle
  – But Google is parsing (much) of text Web...
Lexical Phrases

• Same idea as longest substring match
  – But look for word (not character) sequences
• Compile a term list that includes phrases
  – Technical terminology can be very helpful
• Index any phrase that occurs in the list
• Most effective in a limited domain
  – Otherwise hard to capture most useful phrases
Syntactic Phrases

- Automatically construct “sentence diagrams”
  - Fairly good parsers are available
- Index the noun phrases
  - Might work for queries that focus on objects

The quick brown fox jumped over the lazy dog’s back
“Named Entity” Tagging

• Automatically assign “types” to words or phrases
  – Person, organization, location, date, money, ...

• More rapid and robust than parsing

• Best algorithms use “supervised learning”
  – Annotate a corpus identifying entities and types
  – Train a probabilistic model
  – Apply the model to new text
Stop words

• With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  – They have little semantic content: *the, a, and, to, be*
  – There are a lot of them: ~30% of 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”
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
Summary: “Term” is Whatever You Index

- Word sense
- Token
- Word
- Stem
- Character n-gram
- Phrase

Modified tokens $\rightarrow$ Terms

Flowchart:
1. Tokenizer
2. Linguistic modules
3. Indexer
Summary (Continued)

• The key is to index the right kind of terms
• Start by finding fundamental features
  – So far all we have talked about are character codes
  – Same ideas apply to handwriting, OCR, and speech
• Combine them into easily recognized units
  – Words where possible, character n-grams otherwise
• Apply further processing to optimize the system
  – Stemming is the most commonly used technique
  – Some “good ideas” don’t pan out that way
Dictionary entries – first cut

These may be grouped by language (or not…). More on this in ranking/query processing.

<table>
<thead>
<tr>
<th>English</th>
<th>Japanese</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ensemble.french</strong></td>
<td>時間.japanese</td>
</tr>
<tr>
<td><strong>MIT.english</strong></td>
<td><strong>mit.german</strong></td>
</tr>
<tr>
<td><strong>guaranteed.english</strong></td>
<td><strong>entries.english</strong></td>
</tr>
<tr>
<td><strong>sometimes.english</strong></td>
<td><strong>tokenization.english</strong></td>
</tr>
</tbody>
</table>
PHRASE QUERIES AND POSITIONAL INDEXES
Phrase queries

• Want to answer queries such as “emory university” – as a phrase
• Thus the sentence “I went to university at Emory” is not a match.
  – The concept of phrase queries has proven easily understood by users; about 10% of web queries are phrase queries
• No longer suffices to store only <term : docs> entries
Solution 1: 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.
Longer phrase queries

• Longer phrases are processed as we did with wild-cards:

• *emory university atlanta ga* can be broken into the Boolean query on biwords:

  \[
  \text{emory university AND university atlanta AND atlanta ga}
  \]

Without 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 extended biword.
  – 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 rye
Issues for biword indexes

• False positives, as noted before
• Index blowup due to bigger dictionary
  – Infeasible for more than biwords (e.g., triwords),
    → Too big even for biwords!

• Biword indexes are not the standard solution (for all biwords) but can be part of a combination strategy
Solution 2: Positional indexes

- Store, for each **term**, entries of the form:

  `<number of docs containing term; doc1: position1, position2 ... ; doc2: position1, position2 ... ; etc.>`
Positional index example

• Can compress position values/offsets
• Nevertheless, this expands postings storage substantially

Which of docs 1, 2, 4, 5 could contain “to be or not to be”? 

<be: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>
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** Here, \( /k \) 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 \( k \)?
Positional index size

- You can compress position values/offsets

- A positional index expands postings storage substantially

- Current standard in web search 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
  – Average web page has <1000 terms
  – SEC filings, books, even some epic poems ... easily 100,000 terms

• Consider a term with frequency 0.1%

<table>
<thead>
<tr>
<th>Document size</th>
<th>Postings</th>
<th>Positional postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100,000</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
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
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"

• Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  – A typical web query mixture was executed in ¼ of the time of using just a positional index
  – It required 26% more space than having a positional index alone
BASIC OPTIMIZATION:

FASTER POSTINGS MERGES WITH SKIP POINTERS A.K.A. SKIP LISTS
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?
Yes (if index isn’t changing too fast).
Augment postings with skip pointers (at indexing time)

- Why?
- To skip postings that will not figure in the search results.
- How?
- 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 → 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 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 you’re memory-based
  – The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!
Resources for today’s lecture

- MRS Chapters 1, 2
- BCC Ch 1, Sec 2.1 (can also read ahead, rest of 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
