Today’s topics

- Inverted index storage (continued)
  - Compressing dictionaries in memory
- Processing Boolean queries
- Optimizing term processing
- Skip list encoding
- Wild-card queries
- Positional (phrase)/proximity queries
- Evaluating IR systems – Part I

Inverted index storage

- Last time: Postings compression by gap encoding
- This time: Dictionary storage
  - Dictionary in main memory, postings on disk
    - This is common, especially for something like a search engine where high throughput is essential, but can also store most of it on disk with small in-memory index
  - Tradeoffs between compression and query processing speed
  - Cascaded family of techniques

How big is the lexicon V?

- Grows (but more slowly) with corpus size
- Empirically okay model:
  \[ V = kN^{b} \]
- where \( b = 0.5 \), \( k \approx 30–100 \); \( N \) = # tokens
- For instance TREC disks 1 and 2 (2 Gb; 750,000 newswire articles): ~ 500,000 terms
- Number is decreased by case-folding, stemming
- Indexing all numbers could make it extremely large (so usually don’t)
- Spelling errors contribute a fair bit of size
Dictionary storage - first cut

- Array of fixed-width entries
  - 500,000 terms; 28 bytes/term = 14MB.

<table>
<thead>
<tr>
<th>Terms</th>
<th>Freq</th>
<th>Postings ptr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>999,712</td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>zzz</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

Allows for fast binary search into dictionary

Exercises

- Is binary search really a good idea?
- What are the alternatives?

Fixed-width terms are wasteful

- Most of the bytes in the Term column are wasted
  - we allot 20 bytes for 1 letter terms.
  - And still can’t handle supercalifragilisticexpialidocious.
- Written English averages ~4.5 characters.
  - Exercise: Why is/isn’t this the number to use for estimating the dictionary size?
  - Short words dominate token counts.
- Average word type in English: ~8 characters.
  - Store dictionary as a string of characters:
    - Pointer of next word shows end of last
    - Hope to save up to 60% of dictionary space.

Compressing the term list

- Binarize pointers
- Total string length = 500KB x 8 = 4MB
- Pointers resolve 4MB positions; avg AM = 22 bits = 2.8 bytes
- Save 9 bytes on 3 pointers.

Total space for compressed list

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer
- Avg. 8 bytes per term in term string
  - Now avg. 11 bytes/term, not 20.
- Total string length = 500KB x 8 = 4MB.

Blocking

- Store pointers to every kth on term string.
- Need to store term lengths (1 extra byte)
- Lose 4 bytes on term lengths.
Exercise

- Estimate the space usage (and savings compared to 9.5MB) with blocking, for block sizes of $k = 4$, $8$ and $16$.

Impact on search

- Binary search down to 4-term block;
- Then linear search through terms in block;
- 8 documents: binary tree ave. = 2.6 compares
- Blocks of 4 (binary tree), ave. = 3 compares

Impact on search (Continued)

- $= (1+2+2+1)/8 = (1+2+2+1)/8$

Extreme compression (see MG)

- Front-coding:
  - Sorted words commonly have long common prefix – store differences only (for 3 in 4)
  - Using perfect hashing to store terms "within" their pointers
  - Not good for vocabularies that change.
- Partition dictionary into pages
  - Use B-tree on first terms of pages
  - Pay a disk seek to grab each page
  - If we're paying 1 disk seek anyway to get the postings, "only" another seek/query term.

Compression: Two alternatives

- Lossless compression: all information is preserved, but we try to encode it compactly
  - What IR people mostly do
- Lossy compression: discard some information
  - Using a stoplist can be thought of in this way
  - Techniques such as Latent Semantic Indexing (17 Oct) can be viewed as lossy compression
  - One could prune from postings entries unlikely to turn up in the top $k$ list for query on word
    - Especially applicable to web search with huge numbers of documents but short queries
      - E.g., Carmel et al. SIGIR 2002

Boolean queries: Exact match

- An algebra of queries using AND, OR and NOT together with query words
  - What we used in examples in the first class
  - Uses “set of words” document representation
  - Precise: document matches condition or not
- Primary commercial retrieval tool for 3 decades
  - Researchers had long argued superiority of ranked IR systems, but not much used in practice until spread of web search engines
  - Professional searchers still like boolean queries: you know exactly what you're getting
    - C.f. Google's boolean AND criterion

Query optimization

- Consider a query that is an AND of $t$ terms.
- The idea: for each of the $t$ terms, get its term-doc incidence from the postings, then AND together.
- Process in order of increasing freq:
  - Start with smallest set, then keep cutting further.
Query processing exercises

- If the query is friends AND romans AND (NOT countrymen), how could we use the freq of countrymen?
- How can we perform the AND of two postings entries without explicitly building the 0/1 term-doc incidence vector?

General query optimization

- e.g., (madding OR crowd) AND (ignoble OR strife)
  - Can put any boolean query into CNF
  - Get freq’s for all terms.
  - Estimate the size of each OR by the sum of its freq’s (conservative).
  - Process in increasing order of OR sizes.

Exercise

- Recommend a query processing order for
  (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>eyes</td>
<td>213312</td>
</tr>
<tr>
<td>kaleidoscope</td>
<td>67009</td>
</tr>
<tr>
<td>marmalade</td>
<td>107913</td>
</tr>
<tr>
<td>skies</td>
<td>271658</td>
</tr>
<tr>
<td>tangerine</td>
<td>46653</td>
</tr>
<tr>
<td>trees</td>
<td>316812</td>
</tr>
</tbody>
</table>

Speeding up postings merges

- Insert skip pointers
- Say our current list of candidate docs for an AND query is 8,13,21.
  - (having done a bunch of ANDs)
- We want to AND with the following postings entry: 2,4,6,8,10,12,14,16,18,20,22
- Linear scan is slow.

Augment postings with skip pointers (at indexing time)

2,4,6,8,10,12,14,16,18,20,22,24, ...

- At query time:
  - As we walk the current candidate list, concurrently walk inverted file entry - can skip ahead
    - (e.g., 8,21).
  - Skip size: recommend about √(list length)

Caching

- If 25% of your users are searching for Britney Spears then you probably do not need spelling correction, but you don’t need to keep on intersecting those two postings lists
- Web query distribution is extremely skewed, and you can usefully cache results for common queries
Query vs. index expansion

- Recall, from lecture 1:
  - thesauri for term equivalents
  - soundex for homonyms
- How do we use these?
  - Can "expand" query to include equivalences
    - Query: car tyres → car tyres automobile tires
  - Can expand index
    - Index docs containing car under automobile, as well

Query expansion

- Usually do query expansion
- No index blowup
- Query processing slowed down
  - Docs frequently contain equivalences
  - May retrieve more junk
    - puma → jaguar
    - Carefully controlled wordnets

Wild-card queries: *

- **mon**: find all docs containing any word beginning "mon".
- Easy with binary tree (or B-Tree) lexicon: retrieve all words in range: mon < w < moo
- "mon": find words ending in "mon": harder
  - Permuterm index: for word hello index under:
    - hello$, ello$h, llo$he, lo$hel, o$hell
- Queries:
  - X lookup on X$ X* lookup on X*$
  - X lookup on X$* "X" lookup on X*
  - X*Y lookup on Y$X* X*Y*Z
- ??? Exercise!

Wild-card queries

- Permuterm problem: = quadruples lexicon size
- Another way: index all k-grams occurring in any word (any sequence of k chars)
  - e.g., from text "April is the cruellest month" we get the 2-grams (bigrams)
    - lamp, point, pis, sip, tho, she, ser, unr,
      - see, loves, sth, sm, moon, nth
    - $ is a special word boundary symbol

Processing n-gram wild-cards

- Query **mon** can now be run as
  - $m AND mo AND on
- Fast, space efficient
- But we'd get a match on moon.
- Must post-filter these results against query.
- Further wildcard refinements
  - Cut down on pointers by using blocks
  - Wild-card queries tend to have few bigrams
  - Keep postings on disk
  - Exercise: given a trigram index, how do you process an arbitrary wildcard query?

Phrase search

- Search for "to be or not to be"
- No longer suffices to store only <term:docs> entries
- But could just do this anyway, and then post-filter (i.e., grep) for phrase matches
  - Viable if phrase matches are uncommon
- Alternatively, store, for each term, entries
  - <number of docs containing term;
    - doc1: position1, position2 ... ;
    - doc2: position1, position2 ... ;
    - etc.>
Positional index example

- Can compress position values/offsets as we did with docs in the last lecture
- Nevertheless, this expands postings list in size substantially

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 where “to be or not to be” begins.
  - to:
    - 2:1,17,74,222,551; 4:8,27,101,429,433; 7:13,23,191; ...
  - be:
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Same general method for proximity searches

Example: WestLaw  http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- About 7 terabytes of data; 700,000 users
- Majority of users still use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
    - LIMIT /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- Long, precise queries; proximity operators; incrementally developed; not like web search

Evaluating an IR system – Part I

- What are some measures for evaluating an IR system’s performance?
  - Speed of indexing
  - Index/corpus size ratio
  - Speed of query processing
  - "Relevance" of results
- Note: information need is translated into a boolean query
- Relevance is assessed relative to the information need not the query

Standard relevance benchmarks

- TREC - National Institute of Standards and Testing (NIST) has run large IR testbed for many years
- Reuters and other benchmark sets used
- "Retrieval tasks" specified
  - sometimes as queries
  - Human experts mark, for each query and for each doc, "Relevant" or "Not relevant"
  - or at least for subset that some system returned

Precision and recall

- Precision: fraction of retrieved docs that are relevant = P(relevant/retrieved)
- Recall: fraction of relevant docs that are retrieved = P(retrieved/relevant)

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Not Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>1p</td>
<td>1p</td>
</tr>
<tr>
<td>Not Retrieved</td>
<td>fn</td>
<td>fn</td>
</tr>
</tbody>
</table>

Precision $P = \frac{tp}{tp + fp}$
Recall $R = \frac{tp}{tp + fn}$
Why not just use accuracy?

- How to build a 99.9999% accurate search engine on a low budget...
  - People doing information retrieval want to find something and have a certain tolerance for junk

Precision/Recall

- Can get high recall (but low precision) by retrieving all docs for all queries!
- Recall is a non-decreasing function of the number of docs retrieved
  - Precision usually decreases (in a good system)
- Difficulties in using precision/recall
  - Should average over large corpus/query ensembles
  - Need human relevance judgements
  - Heavily skewed by corpus/authorship

A combined measure: F

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):
  \[ F = \frac{1}{\frac{1}{P} \frac{1}{R} + (1 - \alpha) \frac{1}{P} \frac{1}{R} + \beta} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \]
- People usually use balanced F1 measure
  - i.e., with \( \beta = 1 \) or \( \alpha = \frac{1}{2} \)
- Harmonic mean is conservative average
  - See C.J. van Rijsbergen, Information Retrieval

F1 and other averages

- Combined Measures
- Minimum
- Maximum
- Arithmetic
- Geometric
- Harmonic

Resources for today’s lecture

- Managing Gigabytes, Chapter 4
  - Sections 4.0 – 4.3 and 4.5.
- Modern Information Retrieval, Chapter 3.
- Princeton Wordnet
  - http://www.cogsci.princeton.edu/~wn/

Glimpse of what’s ahead

- Building indices
- Term weighting and vector space queries
- Probabilistic IR
- User interfaces and visualization
- Link analysis in hypertext
- Web search
- Global connectivity analysis on the web
- XML data
- Large enterprise issues