

CS347

Lecture 2

April 9, 2001

©Prabhakar Raghavan

Today's topics

- Inverted index storage
 - Compressing dictionaries into memory
- Processing Boolean queries
 - Optimizing term processing
 - Skip list encoding
- Wild-card queries
- Positional/phrase queries
- Evaluating IR systems

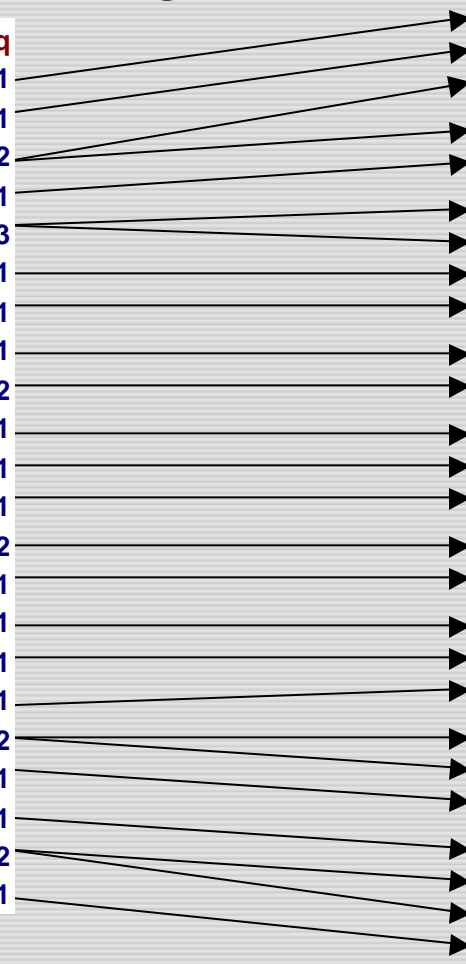
Recall dictionary and postings files

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1



Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

↑
In memory



Doc #	Freq
2	1
2	1
1	1
2	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	1
2	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1

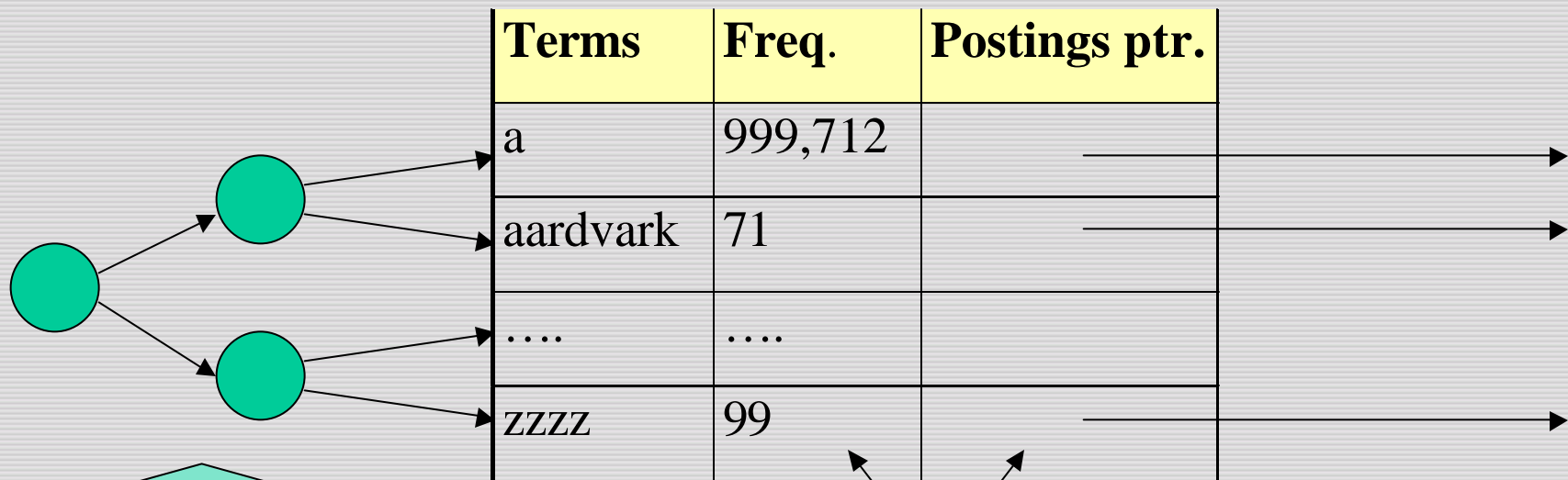
*Gap-encoded,
on disk* ↑

Inverted index storage

- Last time: Postings compression by gap encoding
- Now: Dictionary storage
 - Dictionary in main memory, postings on disk
- Tradeoffs between compression and query processing speed
 - Cascaded family of techniques

Dictionary storage - first cut

- Array of fixed-width entries
 - 28bytes/term = 14MB.



Allows for fast binary search into dictionary

20 bytes

4 bytes each

Exercise

- Is binary search really a good idea?
- What's a better alternative?

Fixed-width terms are wasteful

- Most of the bytes in the **Term** column are wasted - we allot 20 bytes even for 1-letter terms.
 - Still can't handle *supercalifragilisticexpialidocius*.
- Average word in English: ~8 characters.
 - *Written English averages ~4.5 characters: short words dominate usage.*
- Store dictionary as a string of characters:
 - Hope to save upto 60% of dictionary space.

Compressing the term list

....systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo....

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Binary search
these pointers

Total string length =
500KB x 8 = 4MB

Pointers resolve 4M
positions: $\log_2 4M =$
22bits = 3bytes

Total space for compressed list

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer } Now avg. 11
- Avg. 8 bytes per term in term string } bytes/term,
- 500K terms \Rightarrow 9.5MB } not 20.

Blocking

- Store pointers to every k th on term string.
- Need to store term lengths (1 extra byte)

....7systile9syzygetic8syzygial6syzygy11szaibelyite8szczecin9szomo....

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

} Save 9 bytes
} on 3
} pointers.

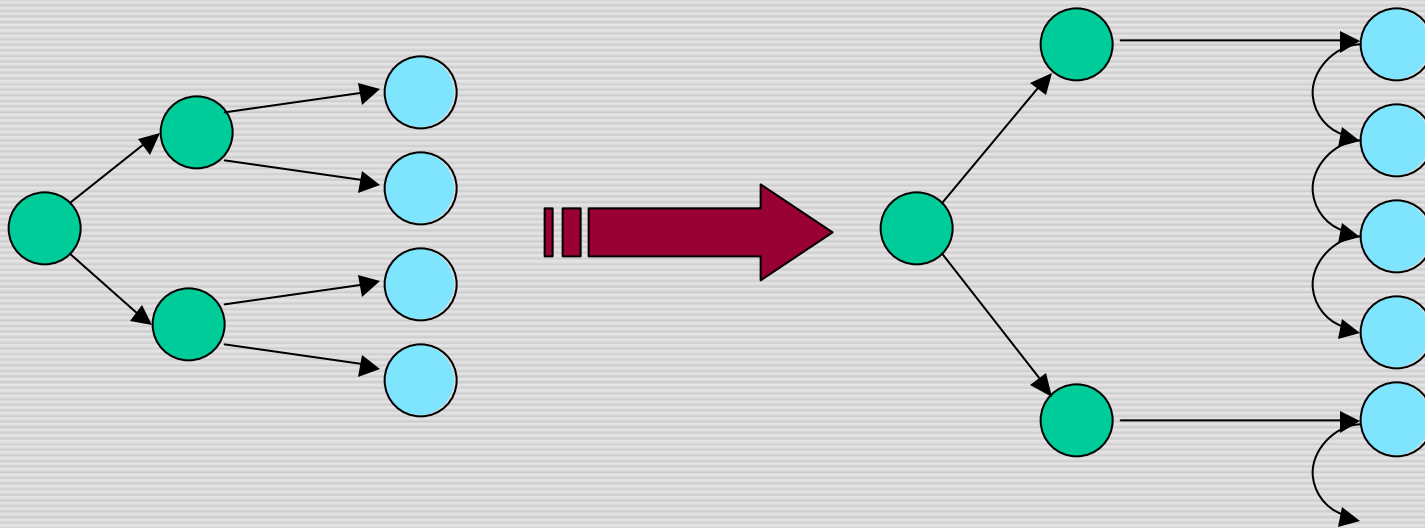
← Lose 4 bytes on term lengths.

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.
- Instead of chasing 2 pointers before, now chase 0/1/2/3 - avg. of 1.5.

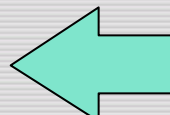


Extreme compression

- 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.

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.*



This is why we kept freq in dictionary.

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*)
- Get freq's for all terms.
- Estimate the size of each *OR* by the sum of its freq's.
- Process in increasing order of *OR* sizes.

Exercise

- Recommend a query processing order for

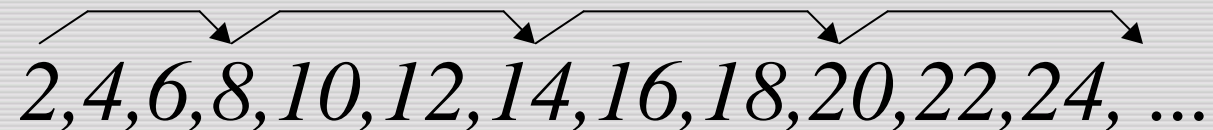
*(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)*

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

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 *AND*s)
- 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 $\sqrt{(\text{list length})}$

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”.
- Solution: index all *k*-grams occurring in any doc (any sequence of *k* chars).
- *e.g.*, from text “April is the cruelest month” we get the 2-grams (*bigrams*)
 - \$ is a special word boundary symbol

\$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\$

Processing wild-cards

- Query *mon** can now be run as
 - *\$m AND mo AND on*
- But we'd get a match on *moon*.
- Must post-filter these results against query.
- Exercise: Work out the details.

Further wild-card refinements

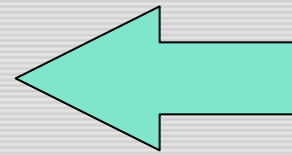
- Cut down on pointers by using blocks
- Wild-card queries tend to have few bigrams
 - keep postings on disk
- *Exercise: given a trigram index, how do you process an arbitrary wild-card query?*

Phrase search

- Search for “*to be or not to be*”
- No longer suffices to store only $\langle term:docs \rangle$ entries.
- Instead store, for each *term*, entries
 - \langle number of docs containing *term*;
 - *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, ...>



Which of these docs
could contain “*to be
or not to be*”?

Can compress position values/offsets as we did with docs in the last lecture.

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; ...

Evaluating an IR system

- 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

Standard relevance benchmarks

- TREC - National Institute of Standards and Testing (NIST)
- Reuters and other benchmark sets
- “Retrieval tasks” specified
 - sometimes as queries
- Human experts mark, for each query and for each doc, “Relevant” or “Not relevant”

Precision and recall

- Precision: fraction of retrieved docs that are relevant
- Recall: fraction of relevant docs that are retrieved
- Both can be measured as functions of the number of docs retrieved

Tradeoff

- 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
 - but precision usually decreases (in a good system)

Difficulties in precision/recall

- Should average over large corpus/query ensembles
- Need human relevance judgements
- Heavily skewed by corpus/authorship

Glimpse of what's ahead

- Building indices
- Term weighting and vector space queries
- Clustering documents
- Classifying documents
- Link analysis in hypertext
- Mining hypertext
- Global connectivity analysis on the web
- Recommendation systems and collaborative filtering
- Summarization
- Large enterprise issues and the real world

Resources for today's lecture

- *Managing Gigabytes*, Chapter 4.
- *Modern Information Retrieval*, Chapter 3.
- Princeton Wordnet
 - <http://www.cogsci.princeton.edu/~wn/>