Indexing Models

CISC489/689-010, Lecture #4

Wednesday, Feb. 18

Ben Carterette

Last Time

- Text processing:
  - – Parsing – identifying important parts of document.
  - – Tokenizing – separating text into words.
  - – Stopping – removing very common words.
  - – Stemming – reducing words to common stems.
- Purpose: get text documents into a state ready for indexing.

Indexes

- *Indexes* are data structures designed to make search faster
- Text search has unique requirements, which leads to unique data structures
- Most common data structure is *inverted index*
  - general name for a class of structures
  - "inverted" because documents are associated with words, rather than words with documents
    - similar to a *concordance*

Indexes and Ranking

- Indexes are designed to support *search*
  - faster response time, supports updates
- Text search engines use a particular form of search: *ranking*
  - documents are retrieved in sorted order according to a score computed using the document representation, the query, and a *ranking algorithm*
- What is a reasonable abstract model for ranking?
  - enables discussion of indexes without details of retrieval model

Matching Documents and Queries

- What are the important properties of a document that indicate that it might be relevant to a query?
  – Query terms are in the document
  – Terms related to query terms are in the document
  – Contains query terms that are rare in collection
  – Lots of pages that contain query terms link to it
  – Links to other pages that contain query terms
  – Query terms close together in document
  – Date of last document update
  – Popularity of document

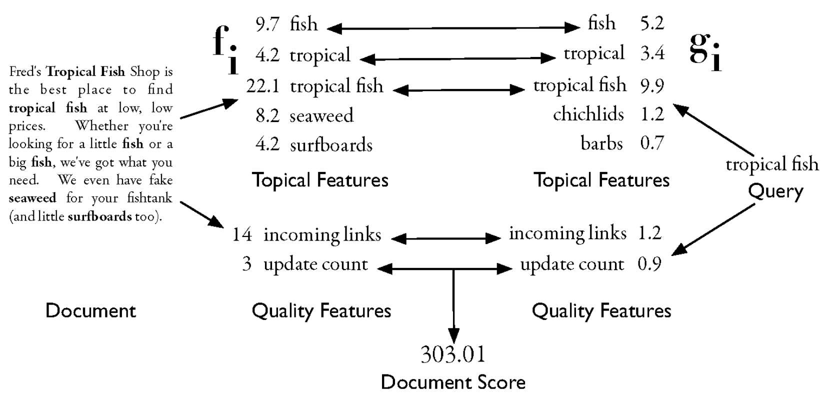
Scoring Documents

- Given those features, how would you score a document for a query?
  – Does it need to contain all the terms? Some?
  – Which features are more important?
  – How do you express their importance?
  – Add features? Multiply?

Concrete Model

$$R(Q,D) = \sum_i g_i(Q) f_i(D)$$

f_i is a document feature function
g_i is a query feature function



Storing Information

What information does the index need to contain to make the scoring happen quickly?

- Important features:
  - Terms are in document.
  - Query terms are rare.
  - Query terms are close together.
- Feature importance:
  - Number of times term appears.
  - Length of document.
  - Rarity of query term.
  - Number of words between query terms.
  - Number of times they occur together versus apart.

Number of times each term appears in each document (term frequency).
Number of documents each term appears in (document frequency).
Length of documents.

Example “Collection”

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Simple Index Model

	tropic	fish	include	found	environ	around	world	both	freshwat	salt	water	speci	...	length
S_1	2	2	2	1	1	1	1	1	1	1	1	1	0	15
S_2	2	3	0	0	0	0	0	0	0	0	1	0	12	18
S_3	1	2	0	0	0	0	0	0	0	0	0	0	7	10
S_4	0	2	0	0	0	0	0	0	1	1	1	0	8	13
doc. freq.	3	4	1	1	1	1	1	1	2	2	3	1	...	56

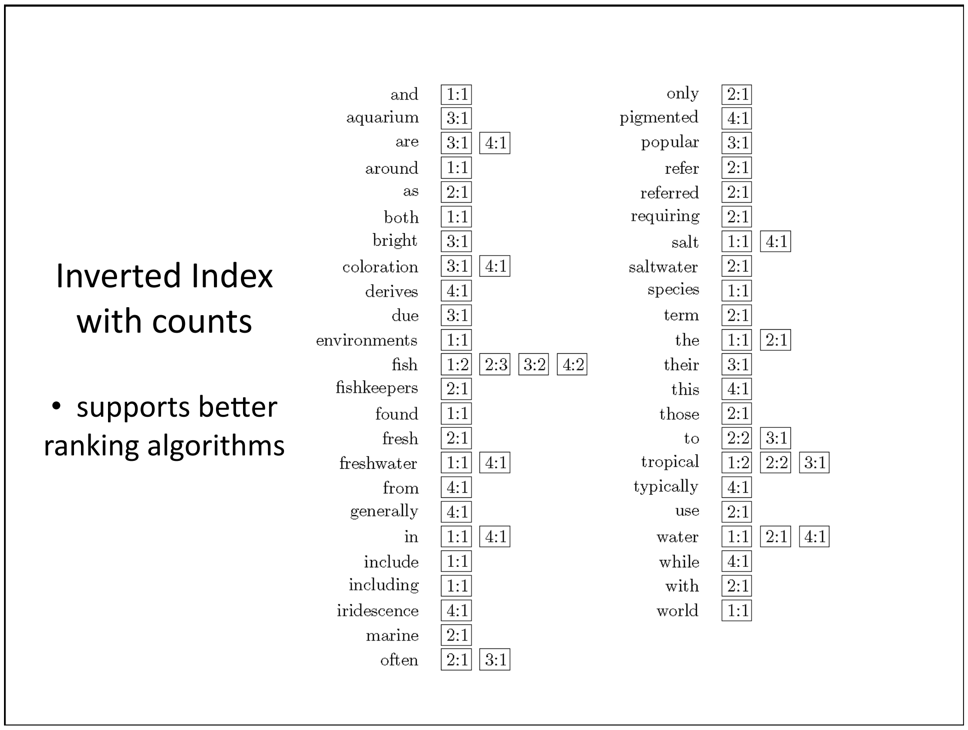
- Stores term frequencies in documents, document frequencies of terms, document lengths.
- What is the problem?
  - Suppose integers require 4 bytes of storage.
  - As collection size increases, space requirements grow rapidly.
    - 1M docs with 2M terms = 7451 Gb of disk space!
  - Do we really need to store all those 0s?

Inverted Index

- Each index term is associated with an *inverted list*
  - Contains lists of documents, or lists of word occurrences in documents, and other information
  - Each entry is called a *posting*
  - The part of the posting that refers to a specific document or location is called a *pointer*
  - Each document in the collection is given a unique number
  - Lists are usually *document-ordered* (sorted by document number)

Simple Inverted Index

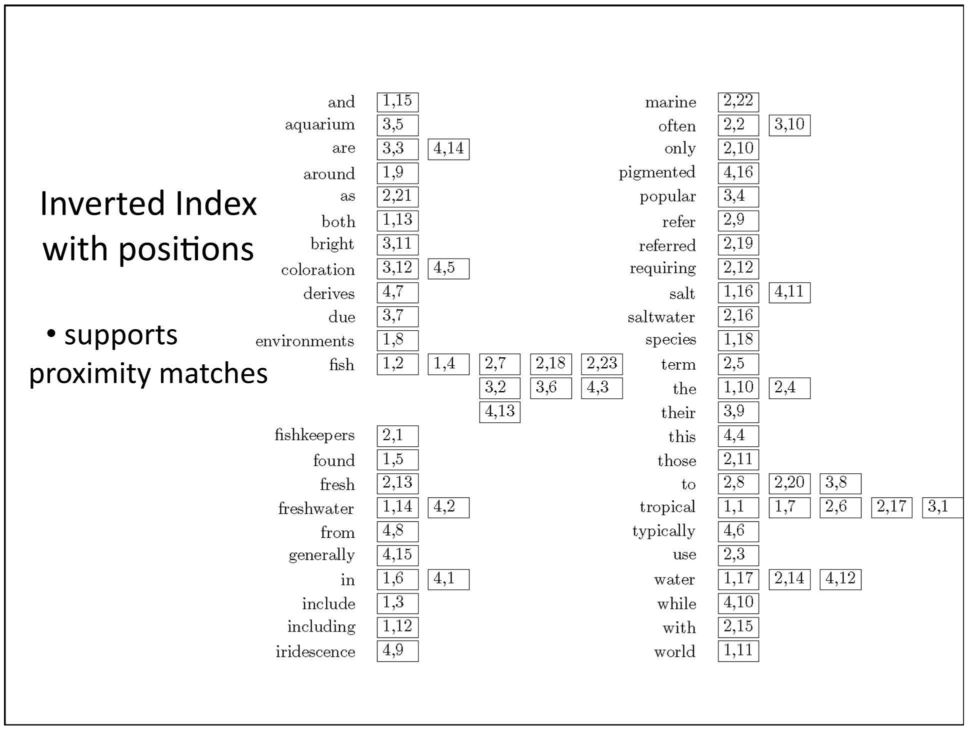
and	1			
aquarium	3			
are	3	4		
around	1			
as	2			
both	1			
bright	3			
coloration	3	4		
derives	4			
due	3			
environments	1			
fish	1	2	3	4
fishkeepers	2			
found	1			
fresh	2			
freshwater	1	4		
from	4			
generally	4			
in	1	4		
include	1			
including	1			
iridescence	4			
marine	2			
often	2	3		
only	2			
pigmented	4			
popular	3			
refer	2			
referred	2			
requiring	2			
salt	1	4		
saltwater	2			
species	1			
term	2			
the	1	2		
their	3			
this	4			
those	2			
to	2	3		
tropical	1	2	3	
typically	4			
use	2			
water	1	2	4	
while	4			
with	2			
world	1			



Inverted Index with counts

- supports better ranking algorithms

and	1:1			
aquarium	3:1			
are	3:1	4:1		
around	1:1			
as	2:1			
both	1:1			
bright	3:1			
coloration	3:1	4:1		
derives	4:1			
due	3:1			
environments	1:1			
fish	1:2	2:3	3:2	4:2
fishkeepers	2:1			
found	1:1			
fresh	2:1			
freshwater	1:1	4:1		
from	4:1			
generally	4:1			
in	1:1	4:1		
include	1:1			
including	1:1			
iridescence	4:1			
marine	2:1			
often	2:1	3:1		
only	2:1			
pigmented	4:1			
popular	3:1			
refer	2:1			
referred	2:1			
requiring	2:1			
salt	1:1	4:1		
saltwater	2:1			
species	1:1			
term	2:1			
the	1:1	2:1		
their	3:1			
this	4:1			
those	2:1			
to	2:2	3:1		
tropical	1:2	2:2	3:1	
typically	4:1			
use	2:1			
water	1:1	2:1	4:1	
while	4:1			
with	2:1			
world	1:1			



Inverted Index with positions

- supports proximity matches

and	1,15								
aquarium	3,5								
are	3,3	4,14							
around	1,9								
as	2,21								
both	1,13								
bright	3,11								
coloration	3,12	4,5							
derives	4,7								
due	3,7								
environments	1,8								
fish	1,2	1,4	2,7	2,18	2,23	3,2	3,6	4,3	4,13
fishkeepers	2,1								
found	1,5								
fresh	2,13								
freshwater	1,14	4,2							
from	4,8								
generally	4,15								
in	1,6	4,1							
include	1,3								
including	1,12								
iridescence	4,9								
marine	2,22								
often	2,2	3,10							
only	2,10								
pigmented	4,16								
popular	3,4								
refer	2,9								
referred	2,19								
requiring	2,12								
salt	1,16	4,11							
saltwater	2,16								
species	1,18								
term	2,5								
the	1,10	2,4							
their	3,9								
this	4,4								
those	2,11								
to	2,8	2,20	3,8						
tropical	1,1	1,7	2,6	2,17	3,1				
typically	4,6								
use	2,3								
water	1,17	2,14	4,12						
while	4,10								
with	2,15								
world	1,11								

Proximity Matches

- Matching phrases or words within a window
  - e.g., "`tropical fish`", or “find tropical within 5 words of fish”
- Word positions in inverted lists make these types of query features efficient
  - e.g.,

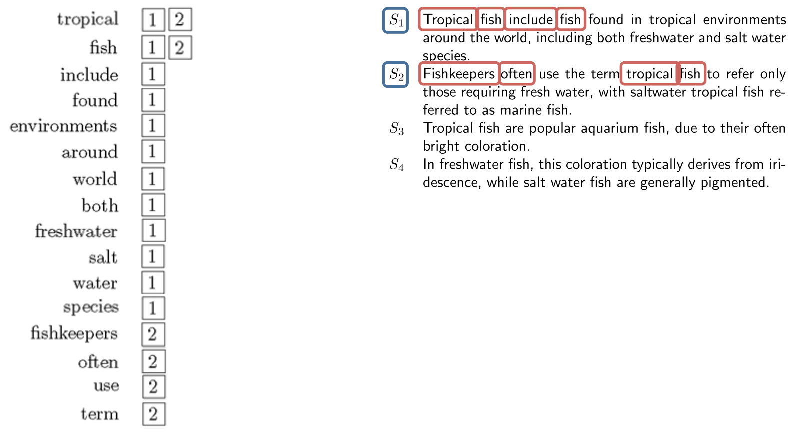
tropical	1,1		1,7	2,6	2,17		3,1			
fish	1,2	1,4		2,7	2,18	2,23	3,2	3,6	4,3	4,13

Index Construction

- Simple in-memory indexer

```
procedure BuildIndex($D$)                    ▷ $D$ is a set of text documents
    $I \leftarrow$ HashTable()                        ▷ Inverted list storage
    $n \leftarrow 0$                                 ▷ Document numbering
    for all documents $d \in D$ do
        $n \leftarrow n + 1$
        $T \leftarrow$ Parse($d$)                ▷ Parse document into tokens
        Remove duplicates from $T$
        for all tokens $t \in T$ do
            if $d \notin I$ then
                $I_d \leftarrow$ Array()
            end if
            $I_d$.append($n$)
        end for
    end for
    return $I$
end procedure
```

Simple Indexing Example



tropical	1 2
fish	1 2
include	1
found	1
environments	1
around	1
world	1
both	1
freshwater	1
salt	1
water	1
species	1
fishkeepers	2
often	2
use	2
term	2

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

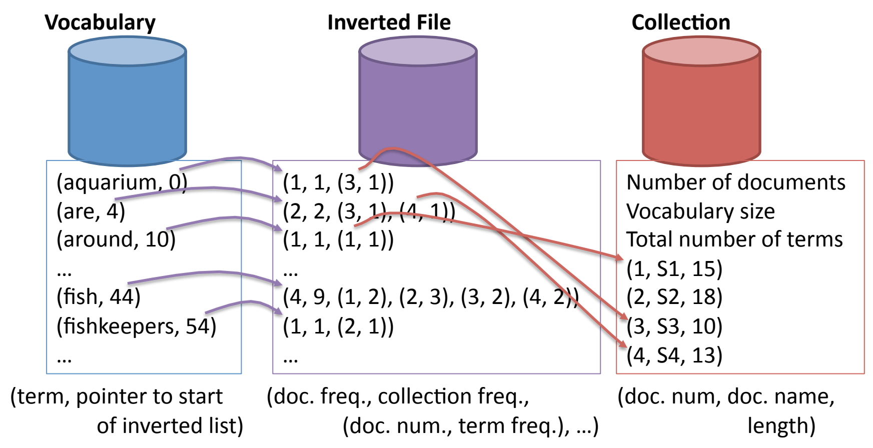
S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Auxiliary Structures

- Inverted lists usually stored together in a single file for efficiency
  - *Inverted file*
- *Vocabulary* or *lexicon*
  - Contains a lookup table from index terms to the byte offset of the inverted list in the inverted file
  - Either hash table in memory or B-tree for larger vocabularies
- Term statistics stored at start of inverted lists
- Collection statistics stored in separate file

Inverted File + Auxiliary Structures



Vocabulary	Inverted File	Collection
(aquarium, 0)	(1, 1, (3, 1))	Number of documents
(are, 4)	(2, 2, (3, 1), (4, 1))	Vocabulary size
(around, 10)	(1, 1, (1, 1))	Total number of terms
…	…	(1, S1, 15)
(fish, 44)	(4, 9, (1, 2), (2, 3), (3, 2), (4, 2))	(2, S2, 18)
(fishkeepers, 54)	(1, 1, (2, 1))	(3, S3, 10)
…	…	(4, S4, 13)
(term, pointer to start of inverted list)	(doc. freq., collection freq., (doc. num., term freq.), …)	(doc. num, doc. name, length)

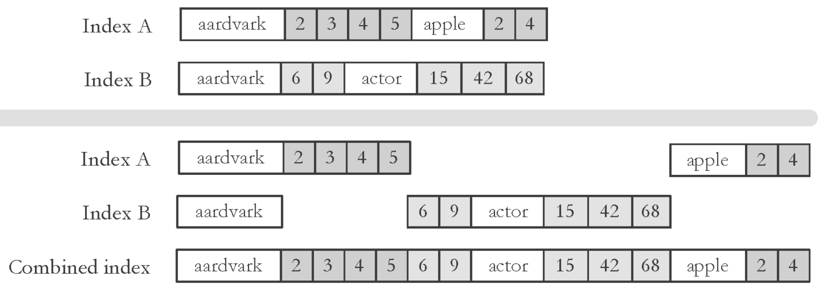
Size of Inverted Lists

- Inverted lists are smaller than our matrix idea, but still can be quite large.
  - 1M documents with 2M terms = 3.7Gb space.
  - Heaps’ law and Zipf’s law can be used to predict the space required by an inverted index.
- Too big to hold in memory.
- Two solutions:
  - Merging on disk.
  - Compression.

Merging

- Merging addresses limited memory problem
  - Build the inverted list structure until memory runs out
  - Then write the partial index to disk, start making a new one
  - At the end of this process, the disk is filled with many partial indexes, which are merged
- Partial lists must be designed so they can be merged in small pieces
  - e.g., storing in alphabetical order

Merging


Index A
aardvark 2 3 4 5 apple 2 4
Index B
aardvark 6 9 actor 15 42 68
Index A
aardvark 2 3 4 5
apple 2 4
Index B
aardvark
6 9 actor 15 42 68
Combined index
aardvark 2 3 4 5 6 9 actor 15 42 68 apple 2 4

Other Solutions

- Inverted files are the most frequently used data structure.
- Others exist:
  - Bitmaps.
  - Signature files.

Bitmap Index

	tropic	fish	include	found	environ	around	world	both	freshwat	salt	water	speci
S_1	1	1	1	1	1	1	1	1	1	1	1	1
S_2	1	1	0	0	0	0	0	0	0	0	1	0
S_3	1	1	0	0	0	0	0	0	0	0	0	0
S_4	0	1	0	0	0	0	0	0	1	1	1	0
doc. freq.	3	4	1	1	1	1	1	1	2	2	3	1

- For each (term, document) pair, store 1 if document contains term, 0 otherwise.
  - Requires 1 bit instead of 4 bytes.
  - Reduces space from 7Tb to 233Gb.
  - But loses a lot of information about terms.

Signature Files

- Bitmaps still have very many 0s.
- Signature file idea: use some of those 0s to store information about other terms.
  - *Freshwat* and *water* could share a bit.
  - Pro: more efficient space usage.
  - Con: someone searching for *freshwater* will get documents about *water* in general (false drops).

Signature Files

- Hash each term to a bitstring of length *k*.
  - H: t → $\{0,1\}^k$
    - k << V
- Document bitstring = bitwise OR of term bitstrings.

tropic	1000 0010 0001 0000
fish	0101 0110 0001 0000
include	0001 0010 0110 0001
found	1001 0100 0010 0000
environ	0001 0100 0100 0001
around	0001 0010 0100 0001
world	0000 0000 0011 0001

Signature Files

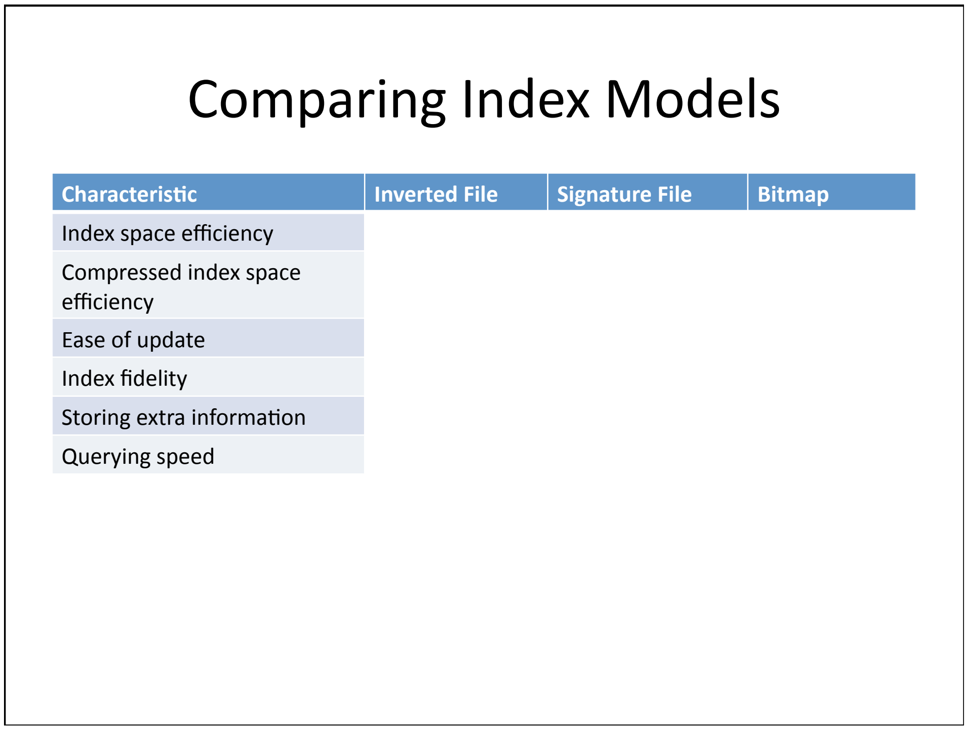
- To query a signature file, produce query bitstring the same way as document bitstring.

tropic	1000 0010 0001 0000
fish	0101 0110 0001 0000
query	1101 0110 0001 0000

- How do you compare queries to documents?
  - Bitwise AND: if (Q && S_1 == Q) then S1 matches.

Signature Files

- Because terms hash to bitstrings that are much smaller than vocabulary size, there will be collisions.
  - Two terms can hash to the same bitstring.
  - Two terms can overlap in many positions.
- Collisions cause *false drops*.
  - Documents match query even though they do not contain any query terms.
- Signature file trade-off:
  - Smaller *k* → less space used by index.
  - Bigger *k* → fewer false drops and better user experience.



Comparing Index Models

Characteristic	Inverted File	Signature File	Bitmap
Index space efficiency			
Compressed index space efficiency			
Ease of update			
Index fidelity			
Storing extra information			
Querying speed			