Indexing

- Extracts from: Witten, Moffat, and Bell, *Managing Gigabytes*, 2nd ed., Morgan Kaufmann, 1999.
- Melnik et al., *Building a Distributed Full-Text Index for the Web*. Proc. 10th Int. WWW Conf., 2001.

Indexing Documents

Basic task:

> Process document collection so docs containing a query word can be retrieved fast.

Input: document collection.

Output: search structure for collection.

Standard Solution

$\boxed{\text{Inverted file + lexicon}}$

- Inverted file = for each word w, list of docs containing w.
- Lexicon = dictionary over all words occuring in doc collection (key = word, value = pointer to inverted file + additional info for word, e.g. length of inverted list).

Lexicon

- Sorted list of occuring words + binary search. How to store variable length strings?
  - Array of pointes into concatenated strings.
  - Do. + blocking
  - Do. + blocking + front coding (prefix compression).
- Hash tables.
- Tries, ternary search trees, suffix arrays (later)
- External: blocking + lexicon over first string in each block. Repeat $\Rightarrow$ prefix B-tree.

Inverted File

Simple (one occurence per doc):

w_1: DocID, DocID, DocID
w_2: DocID, DocID
w_3: DocID, DocID, DocID, DocID, DocID, DocID. . .

Detailed (all occurences in docs):

w_1: DocID, Position, Position, DocID, Position. . .

Even more detailed:

Position annotated with info (heading, boldface, anchor text,. . .).
Useful for ranking.

Compressing the inverted file

- “Hand coding”
  - Store diffs between DocIDs, not absolute DocIDs
  - Code this diff efficiently (unary, γ, δ, Bernoulli (global or local),...).
- Use generic compression tools (gzip,. . .)
- Compress each entire inverted list
- Block the list file, compress each block.

Combine inverted list and lexicon

Melnik et al.:

- Use standard (embedded) DB library (e.g. Berkeley DB).
- Sample entries in inverted file evenly (such that parts between samples can be coded in a page size). Use DB with (key,value) = (sample, next coded part). Generic compression can be applied to parts too.

Preprocessing

- Find words
  - Remove mark-up, scripts,. . .
  - Coding scheme? Unicode, latin-1, ascii?
  - Lowercase
  - Definition of word? (suggestion: alphanumeric sequence, max 4 digits, max 256 chars).
- Stemming? (don’t).
- Stop words? (probably don’t - store all words, and allow stop words at query time).

Building the index

- Hashing only good within RAM. Normally not relevant for web.
- I/O-efficient sorting: OK.

Distribution

- Split on DocID (“local inverted files”).
- Split on WordID (“global inverted files”).

Split on DocID is probably better since for AND-queries, filtering of lists can be done at each machine (less communication).

Melnik et al. give further considerations on efficient distributed building. Among other things: interleave CPU, disk I/O, and net traffic (idea of interleaving CPU time and I/O is also useful for external sorting).