CSA4020

Multimedia Systems:

Adaptive Hypermedia Systems

Lecture 3: Inverted Files

(continued…)

Concerns

- Index represents collection of documents at the time it was indexed

- What if collection needs to be changed?

  Re-indexing from scratch is slow

  Most of the work has already been done, hasn’t it?

- Insertion/deletion overheads

  Depends on frequency of update

  If infrequent but large, consider complete re-indexing

  If frequent, but small, consider on-line insertion/deletion

  If frequent, but large, …

  Currency of index

  Does much information go out of date quickly?

  Workload of system

  Is system accessed 24/7? Is it off-line at certain time of the day/week anyway?

- What sort of system is it?

  Library applications (used mainly during the day by visitors to the library, collection changes infrequently): insertion overhead not a problem

  At the other extreme, newspaper and Web collections pose a problem…

- Factors to consider:

  Size of main memory

  If index fits in memory, then updates will be faster

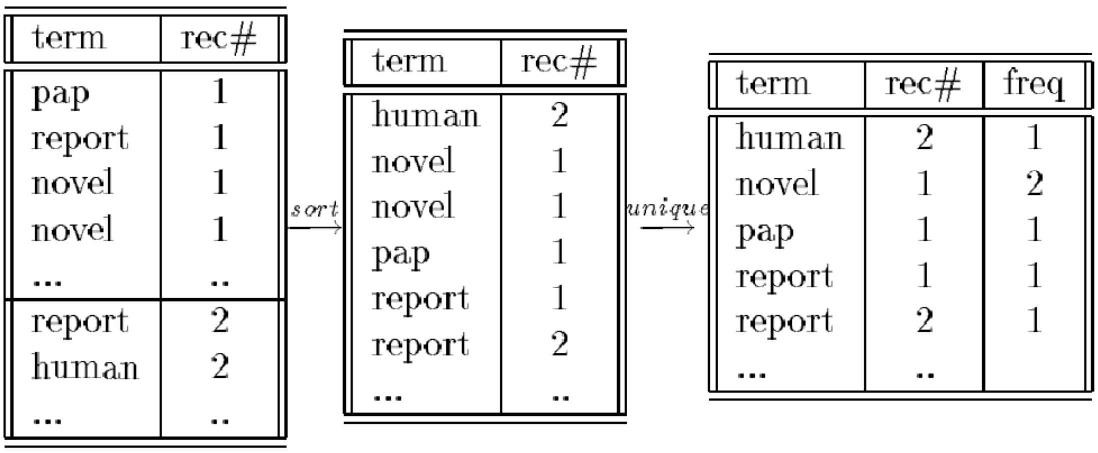
  Temporary disk space

  Can still be a problem if size of index is in terabytes!

  Batched/online updates

Batched Insertion by sorting

- Collect all new documents
- Extract terms for each document
- Prepare inverted index



term	rec#
pap	1
report	1
novel	1
novel	1
...	..
report	2
human	2
...	..

sort

term	rec#
human	2
novel	1
novel	1
pap	1
report	1
report	2
...	..

unique

term	rec#	freq
human	2	1
novel	1	2
pap	1	1
report	1	1
report	2	1
...	..	

- Inverted index can be “inserted” into existing index

  Large overheads if disk blocks need to be moved!

  Don’t write master inverted index to disk sequentially…

  Borrow from db technology to improve efficiency

  Problem if new term needs adding to vocab!

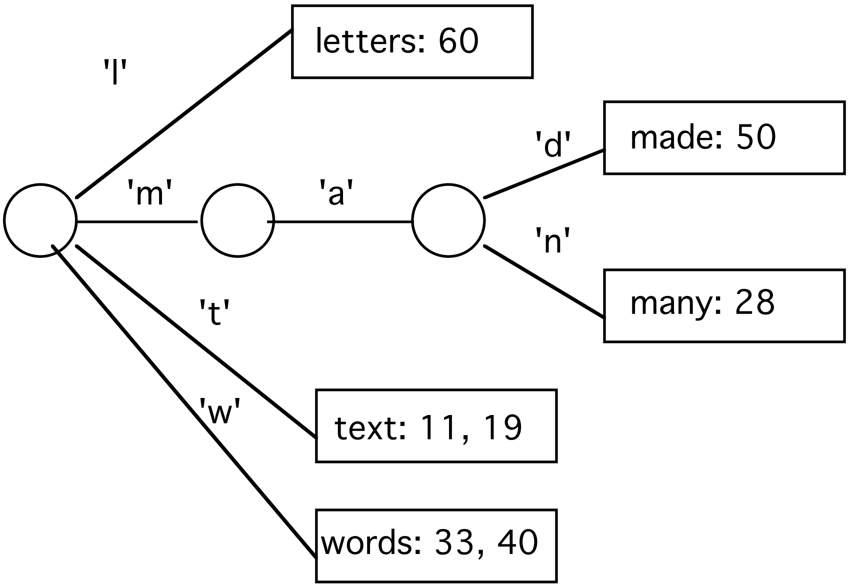
Batched insertions using tries

- Previous example has huge overheads if (db) index blocks need to be split
- Also if new terms need to be added to vocabulary
- Better if index is stored on disk in manner which enables easy insertion
- How is index is constructed from scratch?

  A trie is a digital search tree which uses labelled trees to store strings

  Retrieval time is proportional to length of string

  Each edge of the tree is labelled with a letter



Each word in the text is searched for in the trie (in memory)

If it exists, update occurrence (postings) list

If it doesn’t, add it

Continue until memory is exhausted

Write trie to disk

Flush memory

Repeat until all text is processed

Several partial tries will be written to disk

Merge tries on disk

- Insertions: merge new/old tries

Deletions

- Documents removed from collection…
- Must (ultimately) remove references from postings lists
- Normally, documents in postings lists are indirectly referred to (to save space)
- Can replace entry in lookup table with null reference, indicating doc is deleted
- But operations on inverted index still involve deleted docs… inefficient
- Again, can borrow from db technology to create index of references to docs in postings lists…