Building an Inverted Index

1

Algorithms

- Memory Based
- Sorting
  - Not compressed
  - Compressed
- Merging
  - regular merge
  - multiway merge
  - multiway merge, in place

2

Memory Based Inverted Index

- Phase I (parse and read)
  - For each document
    - Identify distinct terms in the document
    - Update, in memory the posting list for each term
- Phase II (write)
  - For each distinct term in the index
    - Write the inverted index to disk (feel free to compress the posting list while writing it)

3

Variables

B = Text size (figure 5 GB to get started)
N = Number of Documents (about 500,000)
n = Distinct Terms (index size)
F = Number of Words (about 800,000,000
f = Number of posting list entries (about 400,000,000
I = size of compressed file (about 400MB)
L = size of index = 3 MB
Disk seek time = t_s = .01 seconds
Disk Transfer time = t_r = .000005 seconds
Time to compare and swap 10 byte records = t_c. = 000001 seconds
Time to parse, stem, and look up one term = t_p =.000020 seconds
M = Main memory available = (figure 40 MB, a low number)

4

Memory Only Analysis

- Time to read and parse
  - $R = B\, t_r + F\, t_p$
- Time to write
  - $W = I(t_d + t_r)$
- Took about six hours to index a 5GB collection
- Fine, except for memory requirements.

5

Memory Requirements

- While in memory the posting list is not compressed.
- Typical entry

DocID (4 bytes)	tf (2 bytes)	nextPointer (4 bytes)

- For an 800,000,000 word collection, 400,000,000 posting list entries were needed (many terms do not result in a posting list entry because of stop words and duplicate occurences of a term within a document).
- With 400,000,000 posting list entries, at 10 bytes per entry, we obtain a memory requirement of 4GB.

6

Summary

- Pro
  - Very fast algorithm that is easy to implement.
- Con
  - Doesn't work at all if you run out of main memory. Once memory runs out you start swapping. For small document collections this is a great algorithm, for anything realistic it requires a lot of memory.

7

Simple Alternatives that do not Work at All

- Simply storing the posting lists on disk would require a tremendous amount of I/O. One estimate shows SIX WEEKS to run this.
- Alternatively, we could let the OS take care of the memory and just use virtual memory to solve the problem. This results in significant swapping.

8

Sort Based Inversion

- Phase I
  - Create temp file of triples (termID, docID, tf)
- Phase II
  - Sort the triples using external mergesort
- Phase III
  - Build Inverted index from sorted triples

9

Sort-based Inversion

- Phase I (parse and build temp file)
  - For each document
    - Parse text into terms, assign a term to a termID (use an internal *index* for this)
    - For each distinct term in the document
      - Write an entry to a temporary file with only triples <termID, docID, tf)
- Phase II (make sorted *runs*, to prepare for merge)
  - Do Until End of Temporary File
    - Read as much of the temp file that will fit into memory
    - Sort the triples in memory
    - Write them out in a sorted run

10

Sort-based Inversion (continued)

- Phase II (merge the runs)
  - Repeat until there is only one *run*
  - Merge two sorted pairs of runs into a single run
- Phase III
  - For each distinct term
    - Read all triples for a given term (these will be in sorted order)
    - Build the posting list (feel free to use compression)
    - Write this to the inverted index

11

Time (estimates from testing)

- Read and Parse (5 hours)
- Write temp file (30 minutes)
- Sort (4 hours, 3 hours sort + 1 hour for r/w of temp file)
  - With 40MB and 400,000,000 triples in the temporary file we can hold 4,000,000, 10 byte triples in memory. So we have 100 runs.
- Merge (7 hours for I/O, 2 hours computation)
  - log100 = 7 passes, each pass is a full r/w of temp file
  - compute time to do the merge (comparisons of runs)
- Read sorted temp file and build inverted index (1.5 hours)
- Total time (about 20 hours)

12

Analysis

- Time to read and parse, write file
  - $R = B\ t_r + F\ t_p + 10\ f\ t_r$
- Time to sort
  - $20\ f\ t_r$ (read and write) + $R(1.2k \log k)\ t_c$ (time to sort a run)
    - *k* number of triples that fit into memory
    - *R* number of runs to merge
- Time to merge
  - $(\log R)(20\ f\ t_r + f\ t_c)$ (read, write, compare a run)
- Write the final inverted file
  - $10\ f\ t_r + I(t_d + t_r)$

13

Disk Space Requirements

- Now that we have satisfied the low memory requirements, lets look at disk space.
- For a file of size T we need size 2T of disk space.
- Final merge will be writing a new temp file of size T and we will still be reading from a temp file of size T.

14

Summary

- Pro
  - Not as fast as memory based, but at least is usable. Sort and merge time dominates cost.
- Con
  - Requires twice the amount of disk space as the size of the original text.

15