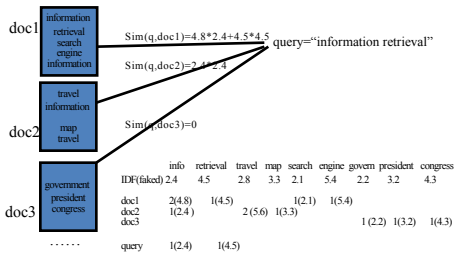
Inverted Indexes

(Search and Data Mining)

Hui Fang
Department of Electrical and Computer Engineering
University of Delaware

A simple VS retrieval function



How to implement it?

Example: Pivoted Normalization Formula

$$S(Q,D)=\sum_{t\in Q\cap D}\frac{1+\ln(1+\ln(c(t,D)))}{(1-s)+s\frac{|D|}{avdl}}\cdot c(t,Q)\cdot\ln\frac{N+1}{df(t)}$$

Observations:

The problem of search requires us to quickly find which documents contain a term.

Indexing

- Indexing
  - Convert documents to data structures that enable fast search
- Inverted index
  - the dominating indexing method for supporting basic search algorithms
  - Provides fast access to all documents containing a given term
  - Given a query, all the indexes of the query term will be fetched for ranking.

A simple solution: Term-Document Matrix

Term	Doc1	Doc2	Doc3	Doc4	Doc5
Sample	1	0	1	0	1
Test	0	1	0	0	1
Dog	1	1	1	0	0
Cat	1	1	0	0	0
Car	0	1	1	0	1
Weather	0	1	1	1	0
Complicated	0	0	0	0	1
Matrix	1	0	0	1	0
Idea	0	1	0	0	1

The term-document matrix is very sparse

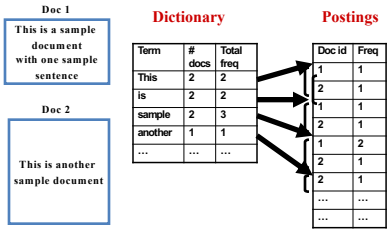
Postings

Term	Doc1	Doc2	Doc3	Doc4	Doc5
Sample	1	0	1	0	1
Test	0	1	0	0	1
Dog	1	1	1	0	0
Cat	1	1	0	0	0
Car	0	1	1	0	1
Weather	0	1	1	1	0
Complicated	0	0	0	0	1
Matrix	1	0	0	1	0
Idea	0	1	0	0	1

Postings
1,3,5
2,5
1,2,3
1,2,
2,3,5
2,3,4
5
1,4
2,5

What goes in the postings?
- Document ID
- Term Weights
- Positions

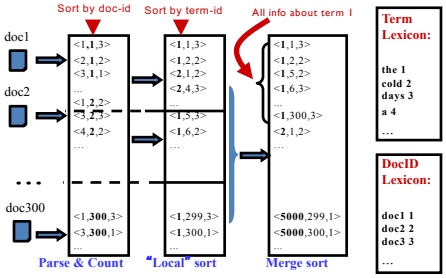
Inverted Index Example



Constructing Inverted Index

- The main difficulty is to build a huge index with limited memory
- Sort-based methods:
  - Step 1: collect local (termID, docID, freq) tuples
  - Step 2: sort local tuples (to make "runs")
  - Step 3: pair-wise merge runs
  - Step 4: Output inverted file

Illustration of the Sort-based Method



Data Structures for Inverted Index

- Dictionary: modest size
  - Needs fast random access
  - Preferred to be in memory
  - Hash table, B+ tree, …
- Postings: huge
  - Sequential access is expected
  - Can stay on disk
  - Compression is desirable

Inverted Index Compression

- Observations:
  - TF compression
    - Small numbers tend to occur more frequently.
  - Doc ID compression
    - "d-gap" (store differences): d1, d2-d1, d3-d2, ..
    - Feasible due to sequential access
- Implications
  - Exploit skewed frequency distribution and use variable-length encoding

Integer Compression Methods

- Binary: equal-length coding
  - 3=>00000011; 5=>00000101
- Unary: x≥1 is coded as x-1 one bits followed by 0
  - 3=> 110; 5=>11110
- γ-code: x=> unary code for $1+\lfloor \log x \rfloor$ followed by binary code for $x-2^{\lfloor \log x \rfloor}$ in $\lfloor \log x \rfloor$ bits
  - 3=>101, 5=>11001
- δ-code: same as γ-code, but replace the unary prefix with γ-code.
  - 3=>1001, 5=>10101