Inf 2B: Indexing and Sorting for the WWW

Kyriakos Kalorkoti

School of Informatics
University of Edinburgh

Inverted Index

Large set D of documents (possibly from WWW).

We have a set of terms appearing in the documents.
The set of terms is called the lexicon.

Definition: An inverted file entry consists of a single term, followed by a list of the locations where the term appears in the set of documents.

Definition: An Inverted Index is a list of inverted file entries, one for each of the terms in the lexicon, presented in order of term number.

Example ‘Set of Documents’

Document	Text
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	Some like it hot, some like it cold,
5	Some like it in the pot,
6	Nine days old.

A childrens rhyme, each line being treated as a document

Inverted Index for our Example

Number	Term	Documents
1	cold	$\langle 2; 1, 4 \rangle$
2	days	$\langle 2; 3, 6 \rangle$
3	hot	$\langle 2; 1, 4 \rangle$
4	in	$\langle 2; 2, 5 \rangle$
5	it	$\langle 2; 4, 5 \rangle$
6	like	$\langle 2; 4, 5 \rangle$
7	nine	$\langle 2; 3, 6 \rangle$
8	old	$\langle 2; 3, 6 \rangle$
9	pease	$\langle 2; 1, 2 \rangle$
10	porridge	$\langle 2; 1, 2 \rangle$
11	pot	$\langle 2; 2, 5 \rangle$
12	some	$\langle 2; 4, 5 \rangle$
13	the	$\langle 2; 2, 5 \rangle$

Note: Frequency refers to number of documents.

Another Inverted Index for our Example

Number	Term	Documents;Words
1	cold	$\langle 2; (1; 6), (4; 8) \rangle$
2	days	$\langle 2; (3; 2), (6; 2) \rangle$
3	hot	$\langle 2; (1; 3), (4; 4) \rangle$
4	in	$\langle 2; (2; 3), (5; 4) \rangle$
5	it	$\langle 2; (4; 3, 7), (5; 3) \rangle$
6	like	$\langle 2; (4; 2, 6), (5; 2) \rangle$
7	nine	$\langle 2; (3; 1), (6; 1) \rangle$
8	old	$\langle 2; (3; 3), (6; 3) \rangle$
9	pease	$\langle 2; (1; 1, 4), (2; 1) \rangle$
10	porridge	$\langle 2; (1; 2, 5), (2; 2) \rangle$
11	pot	$\langle 2; (2; 5), (5; 6) \rangle$
12	some	$\langle 2; (4; 1, 5), (5; 1) \rangle$
13	the	$\langle 2; (2; 4), (5; 5) \rangle$

Inverted Index - Lexicon

1. Set of all words that appear in the set of Documents? OR
2. Set of given keywords forming the allowed vocabulary for search?

Option 1 is most common.

all words is misleading - after parsing a document, we will do some lexical analysis to

- remove “stop words” (for WWW documents, may be many).
- perform case folding (upper case/lower case letters)
- perform stemming

Inverted Index - Granularity

Granularity is the precision to which our Inverted Index locates terms in our set of documents.

First index for “Pease porridge" documents - granularity is document-level (this is the default through this lecture).

Second Index for “Pease porridge" - granularity is word-level (very fine).

Granularity of Index will affect quality of query results.

Inverted Index - Querying

Each term has a term number.

The inverted file entries in the Inverted index are stored in order of term number (in our examples, alphabetical).
Queries:

- A single term, eg “*pease*”:
  Binary search in Inverted Index for term number of “pease" (given by lexicon). return the file entry for this.
- Boolean queries, eg “*pease*" AND “*cold*":
  Binary search for each of the file entries. Then perform **merge**-like linear scan of these lists ($\cap$ for AND, $\cup$ for OR).

Memory-Based Inversion

The “obvious" method for Inversion.

Work entirely in memory, as we have always done (till now).

Dictionary data structure stores items of the form *(term,list)*, where *term* is a term of the lexicon, and *list* is a list of $\langle d, f_{d,t} \rangle$ (document, frequency of t in document) entries.

AVL tree is a good choice for dictionary S.

Phase 1: consider each document d, recovering terms, and appending an entry for each term t in d into the list for t in S.

Phase 2: Read off $\langle t, d, f_{d,t} \rangle$ terms in order from S and into the inverted file.

Memory-Based Inversion

```
Algorithm memoryBasedInversion($D$)
1.  Create a Dictionary data structure $S$.
2.  for $i \leftarrow 1$ to $|D|$ do
3.      Take document $d_i \in D$ and parse it into index terms.
4.      for each index term $t$ in $d_i$ do
5.          Let $f_{d_i,t}$ be the frequency of $t$ in $d_i$.
6.          If $t$ is not in $S$, insert it.
7.          Append $\langle d_i, f_{d_i,t} \rangle$ to $t$'s list in $S$.
8.  for each term $1 \leq t \leq T$ do
9.      Make a new entry in the inverted file.
10.     for each $\langle d, f_{d,t} \rangle$ in $t$'s list in $S$ do
11.         Append $\langle d, f_{d,t} \rangle$ to $t$'s inverted file entry.
12.     Append $t$'s entry to the inverted file.
```

Running Time

Officially, $T_I(D)$ is the sum of:

- $T_p(D)$ (for work in line 3 for all documents)
- $T_q(D)$ (time for lines 4-7 over all $\langle t, d \rangle$ terms in Index)
- $T_w(D)$ (time for the loop in lines 8-12, linear in size of inverted index)

But asymptotic analysis is not relevant here.
Our scenario: pack as many Documents as possible into memory.

Disk space instead of memory

Could we implement Algorithm memoryBasedInversion(*D*) to keep some Documents (and part of the Index) on disk during the algorithm's execution?
. . . so as to pack more into memory.

NO! (lines 8-12 are the problem - need to "hop around" the disk)

Sort-Based Inversion uses **merge** to merge small sorted runs on disk (not in memory).

Careful (Non-sequential) Disk accesses are very expensive. Use two disks *A* and *B*.

- In phase 1 disk *A* is for input, disk *B* for output.
- Roles are revered with each phase.

external MergeSort

```
Algorithm externalMergeSort($A$)
 1.  for $i = 1$ to $n/K$ do
 2.      read block-$i$ of disk-A ($K$ items) into memory;
 3.      sort block-$i$ in memory using ‘in-place’ algorithm, output it.
 4.      /* disk-B now becomes current input-disk */
 5.  for $j = 1$ to $\lceil \lg(n/K) \rceil$ do
 6.      for $i = 1$ to $(n/2^{j+1}K)$ do
 7.          buffer $K/3$ entries of block-$i$ and block-$i+1$ from
             current input-disk into memory;
 8.          initialize the output buffer $b$ (of size $K/3$);
 9.          while there are items left to sort do
10.              do externalMerge on small in-memory blocks
11.              /* output buffer $b$ if full, stream block-$i$ and $i+1$. */
12.      swap role of current input-disk between A and B.
```

Sort-Based Inversion

```
Algorithm sortBasedInversion($D$)
1. Create a Dictionary data structure $S$.
2. Create an empty temp file on disk.
3. for $i \leftarrow 1$ to $|D|$ do
4.         Take document $d_i \in D$ and parse it into index terms.
5.         for each index term $t$ in $d_i$ do
6.                 Let $f_{d_i,t}$ be the frequency of $t$ in $d_i$.
7.                 Check whether $t \in S$ (and check term number $\tau$).
8.                 If $t \notin S$, insert it (with the next free term number $\tau$).
9.                 Write $\langle \tau, d_i, f_{d_i,\tau} \rangle$ to temp file ($\tau$ is $t$'s term number).
```

Algorithm sortBasedInversion(D)

```
1. Call externalMergeSort on *temp file*, to sort in order of $\langle \tau, d \rangle$;
2. /* *temp file* now sorted. Output inverted file. */
3. **for** $1 \leq \tau \leq T$ **do**
4.         Start a new *inverted file entry* for $t$ (term number $\tau$).
5.         Read the triples $\langle \tau, d, f_{d,\tau} \rangle$ from *temp file* into $t$'s entry.
6.         Append $t$'s entry to the *inverted file*.
```

Note that memory size is K above.

Further Reading

Managing Gigabytes by Ian. H. Witten, Alistair Moffat, and Timothy. C. Bell (Chapter 5 and Chapter 3).
Witten et al. give numbers (in terms of hours, Gigabytes).

Lots on the web:

- `Wikipedia`
- Building a distributed Full-test Index for the Web, by S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. *ACM Transactions on Information Systems (TOIS)*, **19**(3). Online at: `http://www10.org/cdrom/papers/275/`
- Very Large Scale Information Retrieval, by David Hawking. Online at: `http://www.inf.ed.ac.uk/teaching/courses/tts/papers`