Corso di Biblioteche Digitali

- Vittore Casarosa
  - Mail: casarosa@isti.cnr.it
  - Tel: 050 621 3115 (office) 348 397 2168 (mobile)
  - Skype: vittore1201
- "Ricevimento" at the end of the lessons or by appointment
- Final assessment
  - 70% oral examination
  - 30% project (development of a small digital library))
- Reference material:
  - Ian Witten, David Bainbridge, David Nichols, How to build a Digital Library, Morgan Kaufmann, 2010, ISBN 978-0-12-374857-7 (Second edition)
  - Material provided by the teacher
- **http://cloudone.isti.cnr.it/casarosa/BDG/**

Modules

- Computer Fundamentals and Networking
- A conceptual model for Digital Libraries
- Bibliographic records and metadata
- Information Retrieval and Search Engines
- Knowledge representation
- Digital Libraries and the Web
- Hands-on laboratory: the Greenstone system

Information Retrieval

- Information Retrieval and Search Engines
  - Indexing a collection of documents
  - Ranking query results
  - Search engines in the Web
  - Ranking in Web search engines

Information Retrieval (IR)

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need, from within large collections (usually stored on computers).
- Research in Information Retrieval started in the seventies, as a field complementary to data base querying (retrieval of structured data)
- Very often Information Retrieval is also called “full text retrieval” or “free text retrieval” or “to google”
- Today, the search engines have made free text retrieval the normal way to search for information

Model of Information Retrieval

- The model of free (or full) text retrieval is:
  - There is a collection of digital documents
  - The user enters a query (usually a few words)
  - The system returns a list of documents **ranked in order of relevance** to the query

- In order to do that efficiently:
  - it is necessary first to build an **index**
  - it is necessary also to **represent the documents (and the query)** in a way suitable for an **algorithm to compute the relevance** of a document with respect to a query

Indexing

- In normal life, is the most common way to find content in a book or in a journal
- For libraries and books, it started a long time ago
  - Table of contents (to know where chapters are in a book)
  - Analytical index (to know where a topic is in a book)
  - Catalog (to know where a book is in the library)
  - Concordances (to know where a word is in a book)
- Free text retrieval is the extension (by computers) of the concept of “concordance”
- In general, “not even think” of doing a linear scan of the document(s) at the time of the query
- What is needed is an index of the words (terms) contained in the whole collection

Concordance of “search” in the Bible

search

enquire and made *s.* *Deut* 13:14
that *s.* may be made in. *Ezra* 4:15
5:17
s. hath been made, and it. 4:19; 6:1
prepare thyself to the *s.* *Job* 8:8
hast thou walked in the *s.*? 38:16*
accomplish a diligent *s.* *Ps* 64:6
my spirit made diligent *s.* 77:6
not found it by secret *s.* *Jer* 2:34

search, *verb*

he shall not *s.* whether. *Lev* 27:33
to *s.* out a resting place. *Num* 10:33
that they may *s.* the land. 13:2
which we have gone to *s.* it. 32
passed through to *s.* it, is. 14:7
of the men that went to *s.* 38
men, and they shall *s.* *Deut* 1:22
before to *s.* you out a place. 33
men to *s.* the country. *Josh* 2:2, 3
the Danites sent men to *s.* *Judg* 18:2
land, I will *s.* him out. *1 Sam* 23:23
servants unto thee to *s.* the city?
2 Sam 10:3; *1 Chr* 19:3
servants, and they shall *s.* *1 Ki* 20:6
s. that none of the servants.
2 Ki 10:23
is it good that he should *s.*? *Job* 13:9
shall not God *s.* this out? *Ps* 44:21

Information in the index

- A collection is a set of "documents", each described (indexed) by a set of "representative terms"
- Need to define beforehand what is a "document"
  - a file, a chapter, a page, a sentence, a word, ...
- Need to define the "granularity" of the index, i.e. the resolution at which term locations within each document are recorded
- The index will contain
  - a list of the different terms that appear in the whole collection
  - for each term, the list of documents where the term appears
  - additional term-related information

Sample "collection"

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.

The documents in the collection are first numbered from 1 to N

Index at document level

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

document numbers

lexicon or vocabulary

inverted list (postings list)

document frequency

Index at word level

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

document numbers

position of word within document

lexicon or vocabulary

inverted list (postings list)

<u>document</u> frequency

Processing of input text

- Obtain character sequence (i.e. the text in the document)
  - Find enconding (e.g. UTF-8), language, document format, etc
- Tokenization
  - Apostrophe, hyphens, compounds, etc.
- Normalization (equivalence classes)
  - Accents and diacritics (e.g. *naive* and *naïve*, *resume* and *résumé*)
  - Capitalization, case folding
    - C.A.T.→ CAT → cat
  - Stemming
    - organize, organizes, organization → organiz
  - Lemmatization
    - am, are, is → be
    - car, cars, car's, cars' → car
- Stop words
  - a, and, at, be, by, for, ...

Tokenization

Input: Friends, Romans, Countrymen, lend me your ears;

Output: | Friends | Romans | Countrymen | lend | me | your | ears |

O'Neill

- neill
- oneill
- o'neill
- o' | neill
- o | neill

aren't

- aren't
- arent
- are | n't
- aren | t

Test collections

		Collection			
		Bible	*GNUbib*	*Comact*	*TREC*
Documents	N	31,101	64,343	261,829	741,856
Number of terms	F	884,994	2,570,906	22,805,920	333,338,738
Distinct terms	n	8,965	46,488	36,660	535,346
Index pointers	f	701,412	2,226,300	12,976,418	134,994,414
Total size (Mbytes)		4.33	14.05	131.86	2070.29

- Bible: we all know (each verse is a document)
- GNUbib: citations to papers in computer science (very short documents)
- Comact: Commonwealth Acts of Australia (about 1 page documents)
- TREC: Text Retrieval Conference (documents – some very long - from different sources, such as news, US Dept of Energy, Wall Street Journal, etc.)

a term is an alphanumeric sequence up to 256 chars or a 4-digit number

Index at document level

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

document numbers

lexicon or vocabulary

inverted list (postings list)

document frequency

Index size

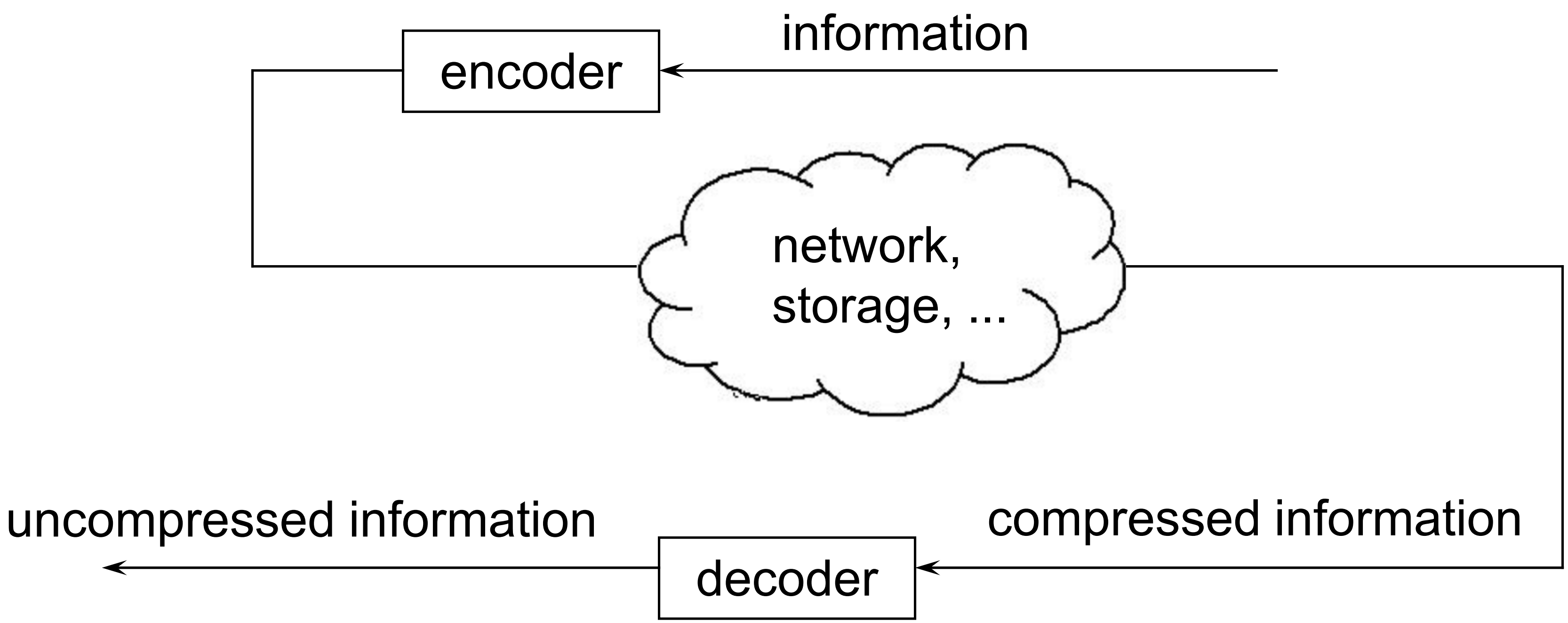
- A document level index needs a value (a number) for each "pointer" (a <term, document> pair)
- With N documents the minimum number of bits required to identify a document is *k*, where 2^k must be ≥ *N)*
- With N documents and f pointers the minimum number of bits required to hold the index is *f* x *k*
  - For TREC document level: 134.994.414 x 20bits = ~ 324 Mbyte (20 is the lowest integer greater than $\log_2$ 741.856)
  - For TREC word level: 333.338.738 x 29bits = ~ 1200 Mbyte (assuming 9 bits for the index of a word within a document)
- A word-level index needs a value for each word in the collection
- An uncompressed inverted file can take as much as the text itself
- For a word level index, assuming that each word appears only once in the documents, we could have 4 bytes for the document pointer and 2 bytes for the "word number" within the document, resulting is six bytes of index for each occurrence of a term
  - assuming an average of six bytes per term (in English), the index takes as much space as the text itself;
- The use of stop words might give significant savings (about 30%)

Powers of 2

$2^0=1$
$2^1=2$
$2^2=4$
$2^3=8$
$2^4=16$
$2^5=32$
$2^6=64$
$2^7=128$
$2^8=256$

$2^9=512$
$2^{10}=1024$ 1K
$2^{11}=2048$ 2K
$2^{12}=4096$ 4K
$2^{13}=8192$ 8K
$2^{14}=16384$ 16K
$2^{15}=32768$ 32K
$2^{16}=65356$ 64K
. . . . . .
$2^{20}=1.048.576$ 1024K 1M
$2^{30}=1.073.741.824$ 1024M 1G
$2^{32}=4.271.406.736$ 4096M 4G

Compression of information



lossless compression: the uncompressed information is identical (bit by bit) to the original information

lossy compression: the uncompressed information contains less "information" than the original information

Index compression methods

Method	Bits per pointer			
	Bible	*GNUbib*	*Comact*	*TREC*
Global methods				
Unary	262	909	487	1918
Binary	15.00	16.00	18.00	20.00
Bernoulli	9.86	11.06	10.90	12.30
γ	6.51	5.68	4.48	6.63
δ	6.23	5.08	4.35	6.38
Observed frequency	5.90	4.82	4.20	5.97
Local methods				
Bernoulli	6.09	6.16	5.40	5.84
Hyperbolic	5.75	5.16	4.65	5.89
Skewed Bernoulli	5.65	4.70	4.20	5.44
Batched frequency	5.58	4.64	4.02	5.41
Interpolative	5.24	3.98	3.87	5.18

Storing the lexicon

- The lexicon is usually stored in the main memory, while the posting (or inverted) lists are usually stored on disk
- The lexicon (the vocabulary) must store the terms, and for each term the address of the inverted file (the postings) stored on disk
- Usually also the document frequency (the number of documents containing the term) is stored in the lexicon
- Other values, such as the term frequency (number of times a term appears in a document), usually needed after the inverted list has been retrieved, are stored as part of the inverted lists
- The lexicon is usually accessed with a binary search or through a hash table
- Memory requirements depend on the structure of the lexicon

Binary search

Example: Find 6 in { 1, 5, 6, 9, 15, 18, 19, 25, 46, 78, 102, 114}.

Step 1 (middle element is 19 > 6, take first half):

1 5 6 9 15 18 19 25 46 78 102 114

Step 2 (middle element is 9 > 6, take first half):

1 5 6 9 15 18 19 25 46 78 102 114

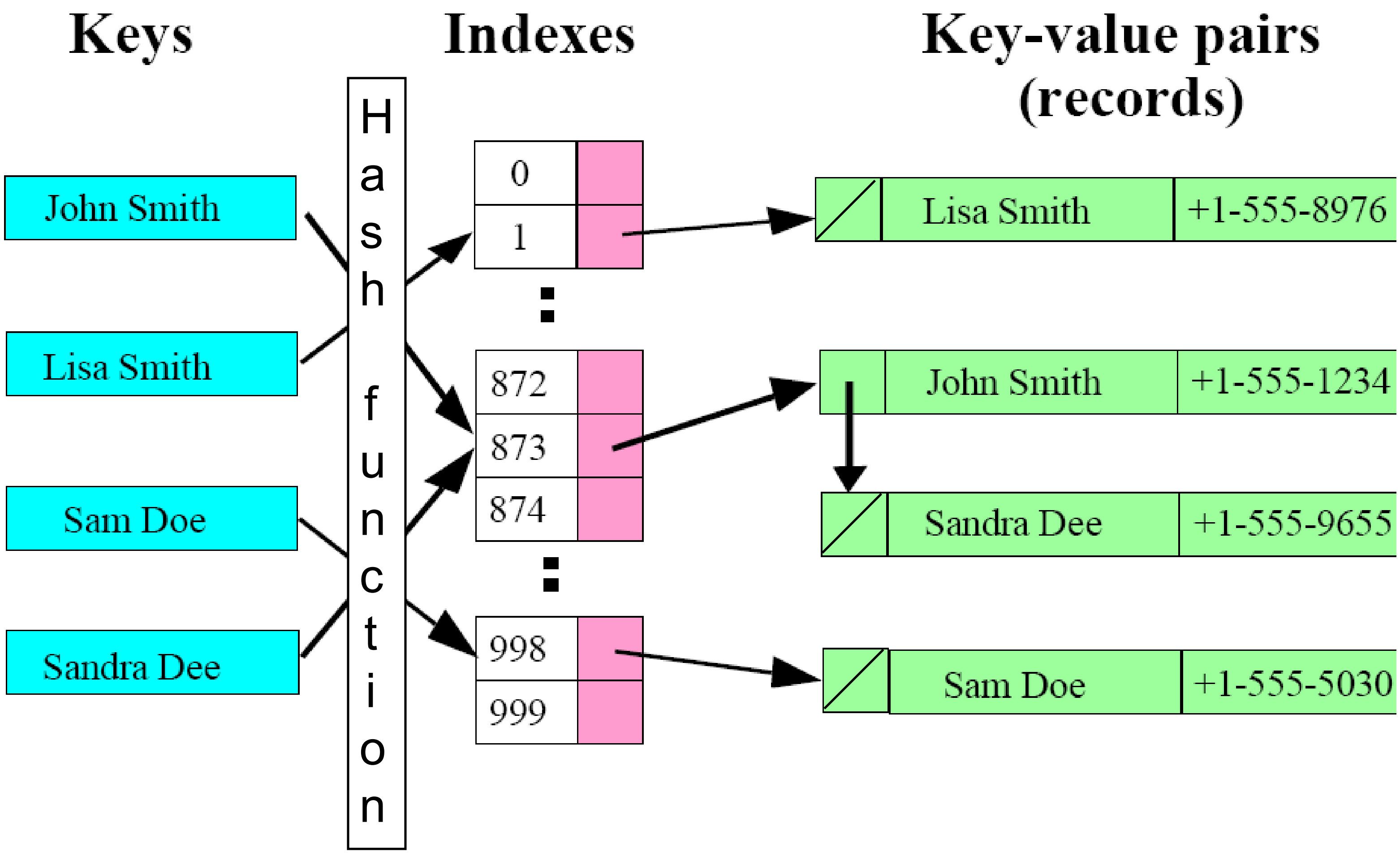
Step 3 (middle element is 5 < 6, take last half):

1 5 6 9 15 18 19 25 46 78 102 114

Step 4 (middle element is 6 == 6, done):

1 5 6 9 15 18 19 25 46 78 102 114

Hash tables



Keys
Indexes
Key-value pairs (records)
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Hash function
0
1
872
873
874
998
999
Lisa Smith
+1-555-8976
John Smith
+1-555-1234
Sandra Dee
+1-555-9655
Sam Doe
+1-555-5030

Storage requirements for the lexicon

Method	Storage
Fixed-length strings	28 Mbytes
Terminated strings	20 Mbytes
Four-entry blocking	18 Mbytes
Front coding	15.5 Mbytes
Minimal perfect hashing	13 Mbytes

storage requirements for one-million-term lexicon

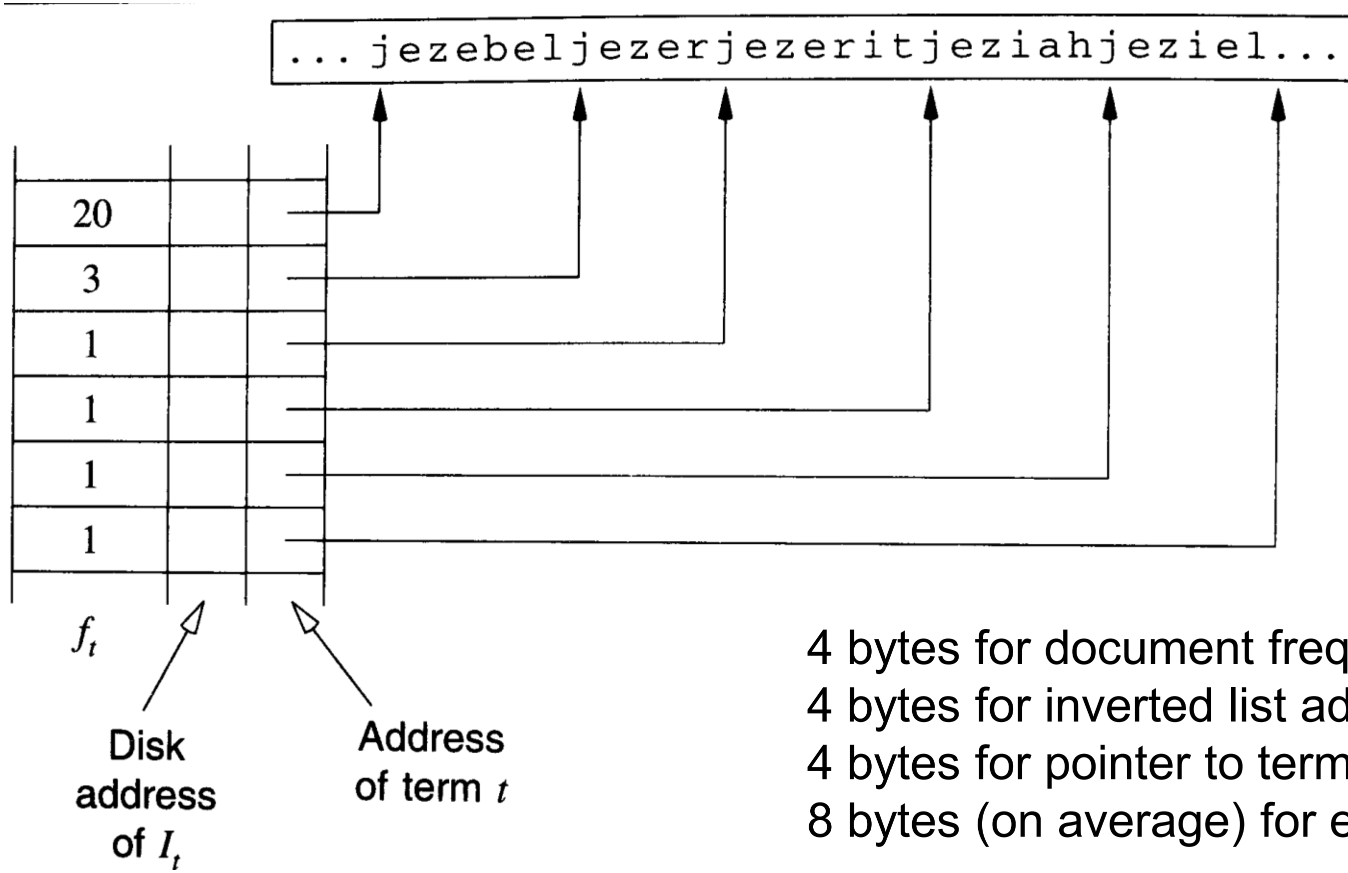
Fixed length strings

Term t	f_t	Disk address of I_t
jezebel	20	→
jezer	3	→
jezerit	1	→
jeziah	1	→
jeziel	1	→
jezliah	1	→
jezoar	1	→
jezrahiah	1	→
jezreel	39	→

20 bytes per term
4 bytes for the document
_ frequency value
4 bytes inverted list address

1 million terms
28MB storage for lexicon

Terminated strings



4 bytes for document frequency
4 bytes for inverted list address
4 bytes for pointer to term
8 bytes (on average) for each term

1 million terms
20MB storage for lexicon

Building the index (sample collection)

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.

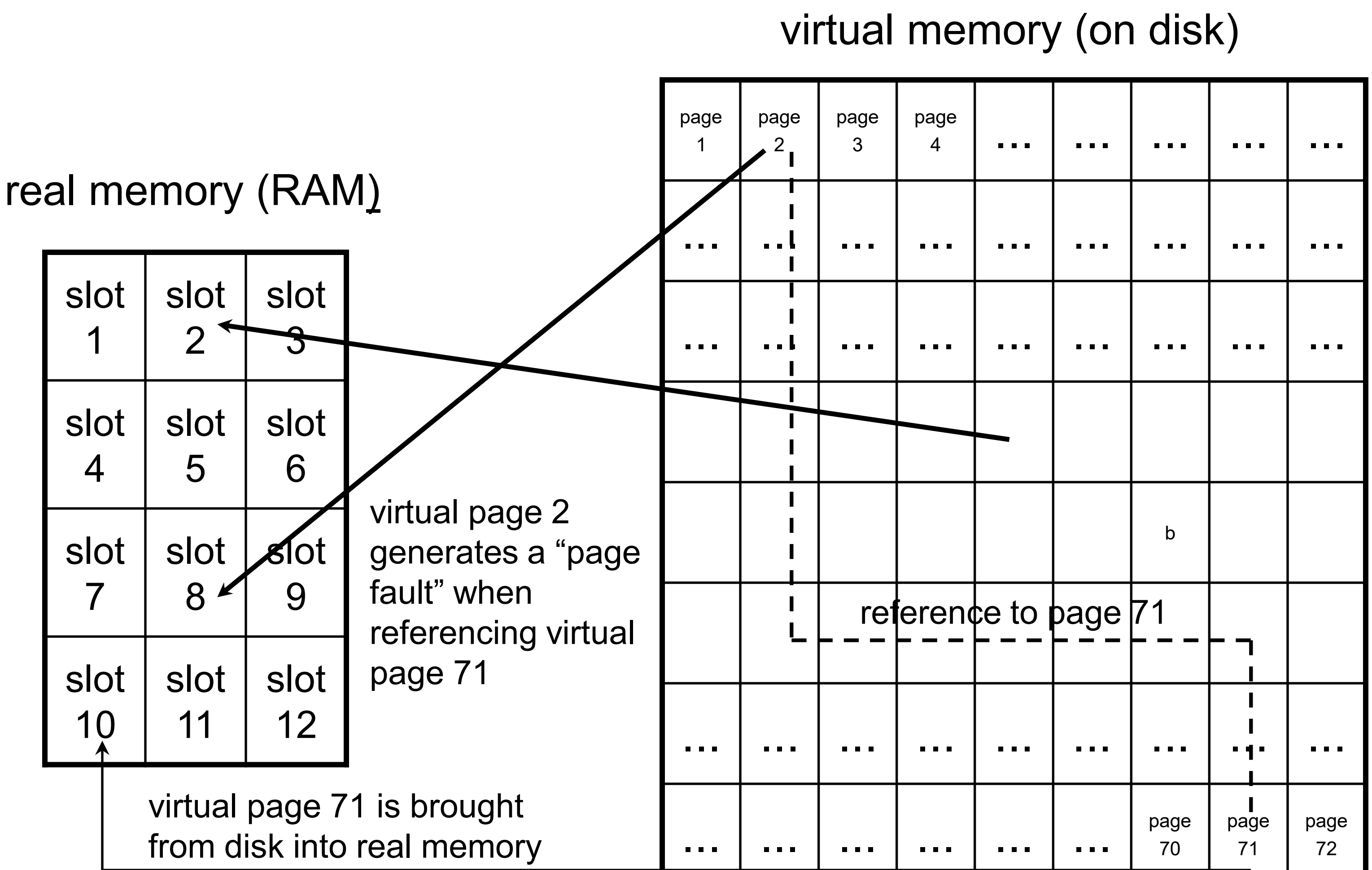
Building the index (frequency matrix)

Number	Term	Document					
		1	2	3	4	5	6
1	cold	1	—	—	1	—	—
2	days	—	—	1	—	—	1
3	hot	1	—	—	1	—	—
4	in	—	1	—	—	1	—
5	it	—	—	—	2	1	—
6	like	—	—	—	2	1	—
7	nine	—	—	1	—	—	1
8	old	—	—	1	—	—	1
9	pease	2	1	—	—	—	—
10	porridge	2	1	—	—	—	—
11	pot	—	1	—	—	1	—
12	some	—	—	—	2	1	—
13	the	—	1	—	—	1	—

Building the index

- Ideal case:
  - Read the text documents one after the other, building one column of the frequency matrix at a time (insert rows when finding new terms)
  - Write the matrix to disk, row by row, in term order (inverted lists)
- Not possible because of memory requirements
  - Assuming 4 bytes for the term frequency
  - Bible → 4 bytes X 8.965 terms X 31.101 docs is about 1 GB
  - TREC → 4 bytes X 535.346 X 741.856 is about 1400 GB
- Use of large “virtual memory” (paging done by the operating system) not possible because of too many “page faults”
  - For the Bible, assuming one page fault per pointer, there will be about 700.000 page faults
  - Assuming 50 page replacement per second, it will take about 14.000 seconds (about 4 hours) for the Bible, and about two months for TREC
- Use of external storage (disk), writing each column as soon as completed not possible because of too much “seek time” when reading back the columns in order to build the inverted list

Paged virtual memory



Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - And then docID

Core indexing step

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

→

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

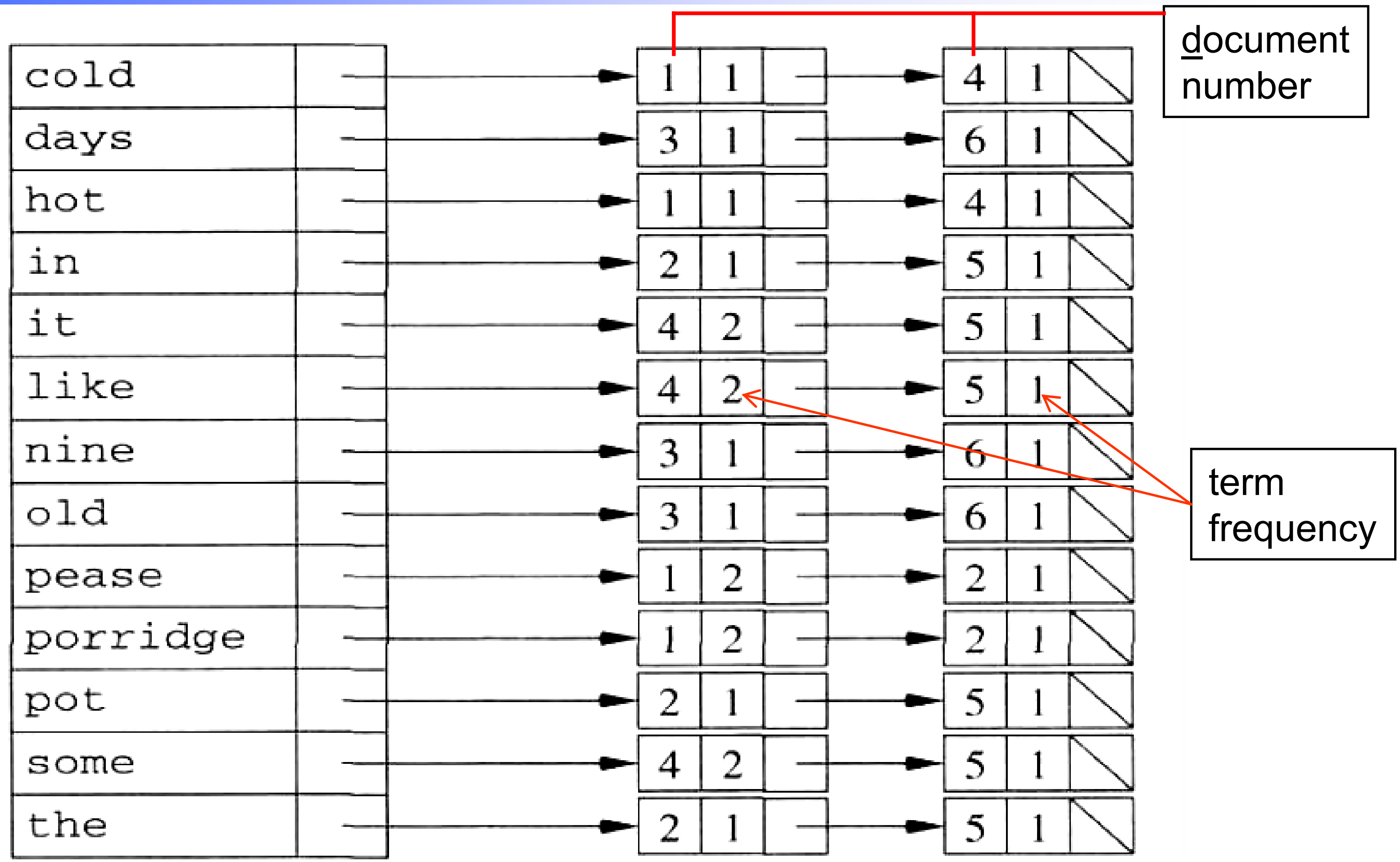
Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged
- Split into Dictionary and Postings
- Document frequency information is added

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Linked lists in memory



Search structure S Linked lists storing $\langle d, f_{d,t} \rangle$ pairs

Typical size and performance figures

Parameter	Symbol	Assumed value
Total text size	B	5×10^9 bytes
Number of documents	N	5×10^6
Number of distinct words	n	1×10^6
Total number of words	F	800×10^6
Number of index pointers	f	400×10^6
Final size of compressed inverted file	I	400×10^6 bytes
Size of dynamic lexicon structure	L	30×10^6 bytes
Disk seek time	t_s	10×10^{-3} sec
Disk transfer time per byte	t_r	0.5×10^{-6} sec
Inverted file coding time per byte	t_d	5×10^{-6} sec
Time to compare and swap 10-byte records	t_c	10^{-6} sec
Time to parse, stem, and look up one term	t_p	20×10^{-6} sec
Amount of main memory available	M	40×10^6 bytes

Different building methods (5M documents, 1M lexicon)

Method	Memory (Mbytes)	Disk (Mbytes)	Time (hours)
Linked lists (memory)	4,000	0	6
Linked lists (disk)	30	4,000	1,100
Sort-based	40	8,000	20
Sort-based compressed	40	680	26
Sort-based multiway merge	40	540	11
Sort-based multiway in-place	40	150	11
In-memory compressed	420	1	12
Lexicon-based, no extra disk	40	0	79
Lexicon-based, extra disk	40	4,000	12
Text-based partition	40	35	15