Chapter 7: Indexing Structures and Search Techniques

Overview

- Introduction
- Sequential searching:
  - finite state automaton
- Indexing structures
  - inverted files
  - indices of XML-marked documents
- Compression and searching

Search approaches

- Because retrieval models usually incorporate term matching
  - important task : **detection of occurrence or non occurrence of a term in the document texts**
  - given the current huge document collections:

    efficiency !
  - when incorporating metadata (e.g., structural data obtained from XML-tagged documents, extracted information, user added tags):

    efficiency !!!

Search approaches

1) Sequential searching:

- scan the text sequentially or online when searching for the query
- appropriate when :
  - the text collection is small (i.e., a few megabytes)
  - the text collection is very volatile (i.e., undergoes modifications very frequently)
  - the index space overhead cannot be afforded
  - searching compressed text directly

Search approaches

2) Build **data structures** over the text (called **indices**)

- to speed up the search:
- inverted files, suffix arrays, and signature files
- worthwhile: large, semi-static text collection: indices updated at regular intervals (e.g., daily)
- mostly done in practice

3) Combination of online and index searching

Sequential searching

- Sequential or online searching =
  finding occurrences of a pattern in the text which is not preprocessed
  - exact string matching
  - string matching allowing errors
  - searching of regular expressions
- **Exact string matching**:
  - given a short pattern *P* of length *m* and a long text *T* of length *n*

    find all text positions where the pattern occurs
  - possible to perform a word, prefix, suffix and substring search

Sequential searching

- many algorithms:
  - window of length *m* slides over the text
  - checks whether the text in the window is equal to the pattern
  - algorithms differ in the way they check and shift the window
  - e.g., brute-force, Knuth-Morris-Pratt, Boyer-Moore algorithms, Shift-Or
- Searching for **regular expressions**:
  - recognized by finite state automaton
  - useful for recognition of phrases and proximity queries (e.g., neglect of erroneous spaces, punctuation, intervening words)

Regular expression

- Regular expression can be expressed by a regular grammar or syntax
- Many lexical patterns can be expressed by a regular grammar:

  example: a simple arithmetic expression <ARITH>:

  - the letters and arithmetic operators are terminal symbols

  - asterisk * indicates zero, one, or more repetitions

<ARITH> ::= (“a” | ”b” | … | “z”) ((”+” | “-“ | “*” |”/”)(“a” | ”b” | … | “z”))*

Finite state automaton

- A **finite state automaton FSA** or finite state machine is a quintuple $\langle K,\Sigma,\delta,y_0,F\rangle$, where:

  K is a finite set, the set of states

  Σ is a finite set, the alphabet (e.g., any finite set of characters or strings)

  $y_0 \in K$, the initial state

  $F \subseteq K$, the set of final states

  δ is a function from $K \times \Sigma$ into K, the transition function or next-state function

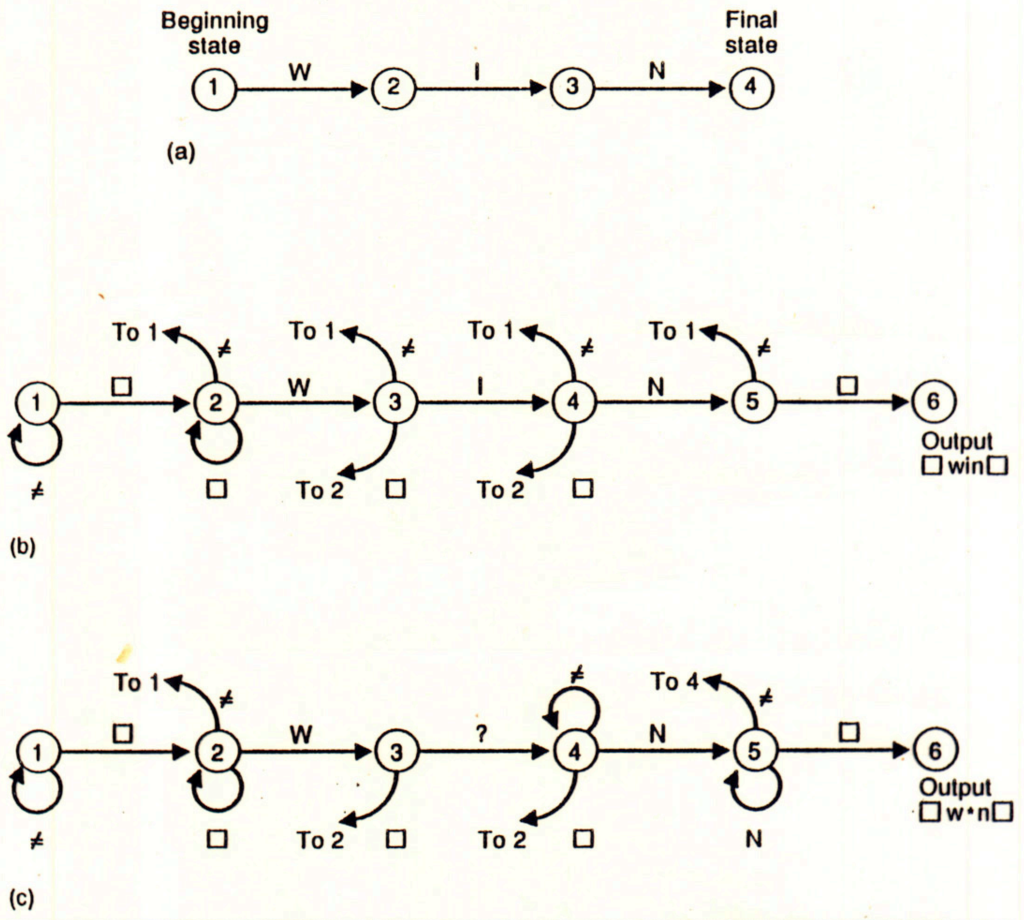
Finite state automaton

- **Deterministic finite state automaton**: with exactly one transition for each given state
- **Non-deterministic finite state automaton**: a state can have any finite number of outgoing transitions: instead of δ we have:
  - a transition relation Δ is a relation from $K \times \Sigma$ into K
- **Finite state transducer**: an output entity ω is constructed when a final state is reached

Finite state automaton (FSA)

- Starting from an initial state y_0, a FSA will move to a next state if it is correctly triggered by the presence of a member of the alphabet in the input string
- When analyzing a valid string, it will eventually end up in one of its final states by repeating this process over a finite number of transitions: this signals a correct parse
- When none of these final states can be reached – i.e. when at a certain point the automaton's definition does not allow for a transition from y_i to y_{i+1} given a certain input character, or when the end of the input string is reached and the automaton has not reached a final state yet – its analysis will fail

Figure 8.19 Finite-state automaton configuration for term recognition (adapted from [2]). (a) Basic FSA diagram for recognition of WIN. (b) Complete automaton for recognition of □WIN□. (c) Automaton for recognition of □W*N□.



[Salton 1971]

Data structures

- Data structures (**indices**):
  - auxiliary data structures to speed up the search of the document representations when querying the document collection
  - important:
    - search cost
    - space overhead for storing the indices
    - cost of building and updating
  - examples:
    - inverted files
    - indices of XML marked documents, link information
    - suffix arrays
    - signature files

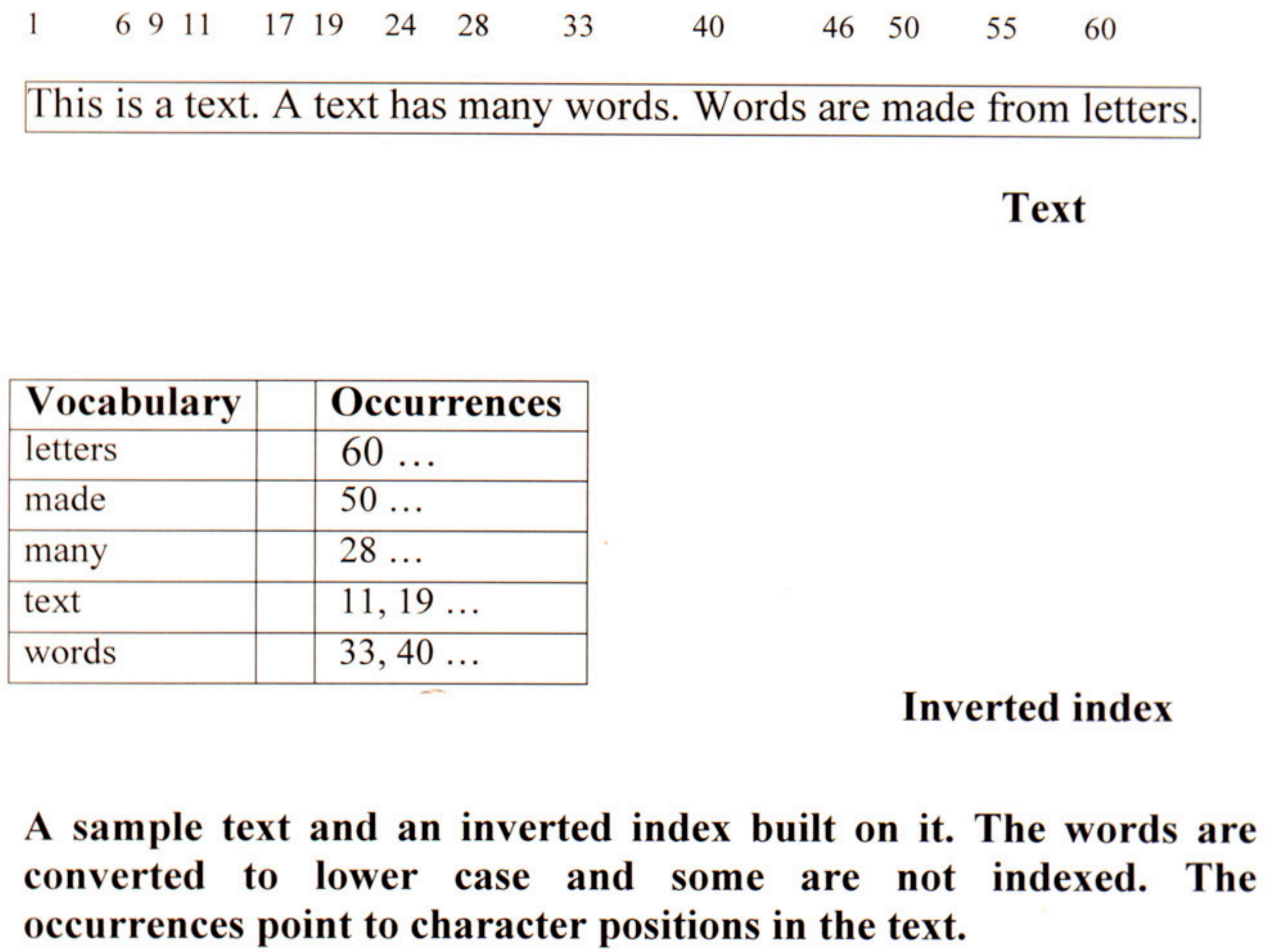
} not treated in this course

Inverted files: concept

- Given:
  - set of documents
  - each document is assigned a list of index terms
- **Inverted file** or index = sorted **list of index terms**
  - index term: can be:
    - unique (stemmed) word (not stopword) or phrase that occurs in the document collection
    - assigned descriptor (e.g., thesaurus term, term/label assigned by means of text categorization/information extraction, content of XML-tag, tag given by user, ...)
  - index term has **pointers** to the documents in which it occurs (document (region) id, address, URL, ...)

Inverted files: concept

- in addition can be stored for each index term:
  - text position: “full inverted index”:
    - character position of the first character of the term
    - word position: advantageous for phrase searching and proximity queries
  - block position:
    - blocks of fixed size (e.g., 256K, 64K)
  - term weight



1 6 9 11 17 19 24 28 33 40 46 50 55 60

This is a text. A text has many words. Words are made from letters.

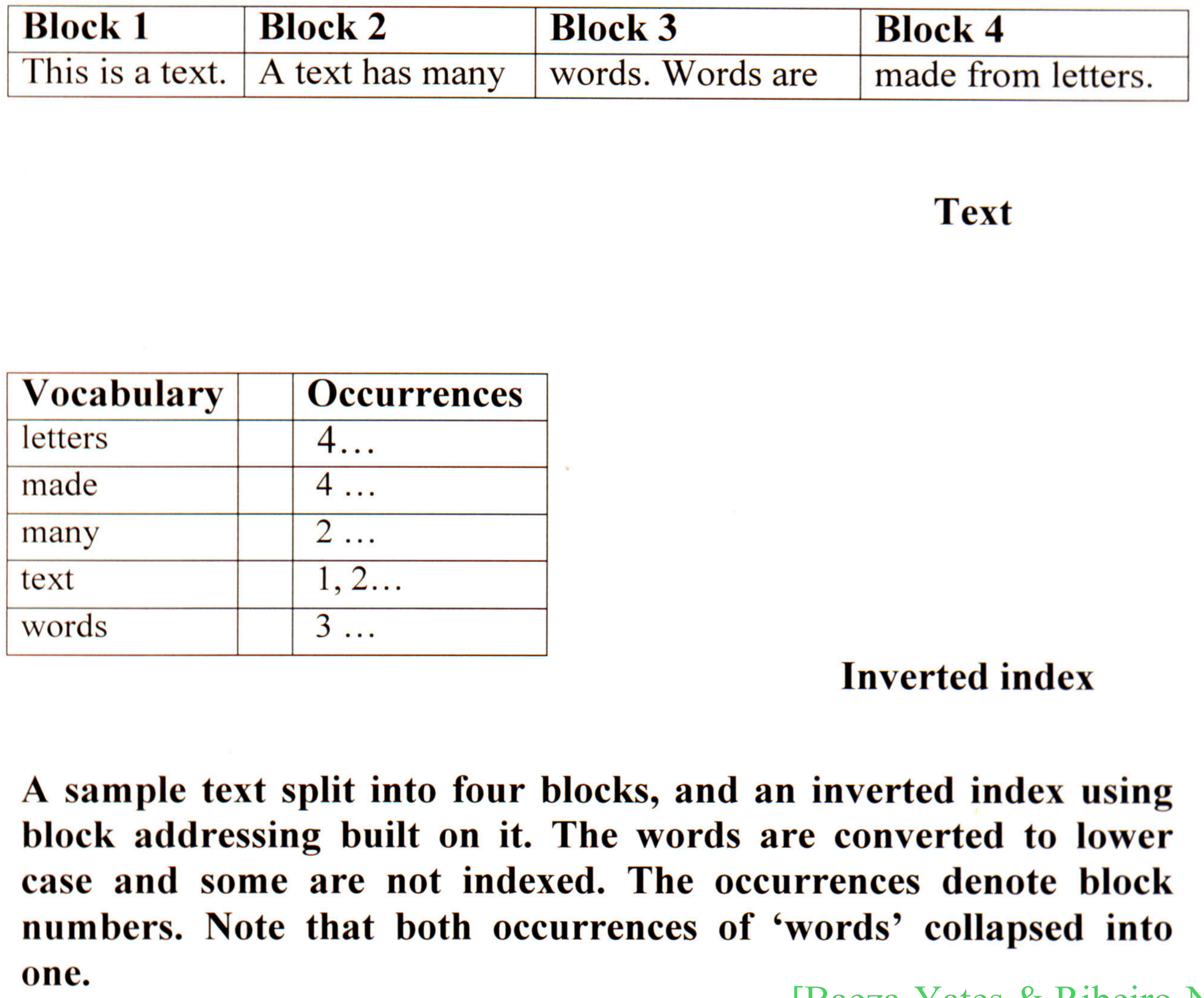
Text

Vocabulary		Occurrences
letters		60 …
made		50 …
many		28 …
text		11, 19 …
words		33, 40 …

Inverted index

A sample text and an inverted index built on it. The words are converted to lower case and some are not indexed. The occurrences point to character positions in the text.

[Baeza-Yates & Ribeiro-Neto 1999]



Block 1	Block 2	Block 3	Block 4
This is a text.	A text has many	words. Words are	made from letters.

Text

Vocabulary		Occurrences
letters		4…
made		4 …
many		2 …
text		1, 2…
words		3 …

Inverted index

A sample text split into four blocks, and an inverted index using block addressing built on it. The words are converted to lower case and some are not indexed. The occurrences denote block numbers. Note that both occurrences of 'words' collapsed into one.

[Baeza-Yates & Ribeiro-Neto 1999]

Inverted files: concept

- Usually each posting of a term is a triple of the form:

$$\left\langle d, f_{d,t}, \left[o_1, \ldots, o_{f_{d,t}} \right] \right\rangle$$

where

d = identifier of a document containing term t

$f_{d,t}$ = frequency of t in d

o = positions in d at which t is observed

- The posting is usually preceded by the number of documents in which the term occurs

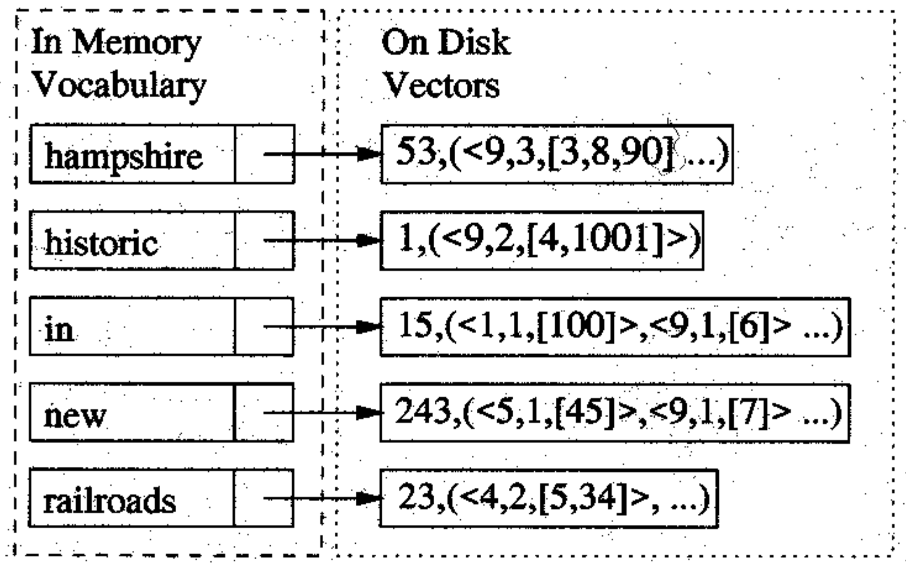


Figure 1: An inverted file for a collection with a vocabulary of five words.

[Bahle et al. 2002]

Distributed indices

- Document-distributed index
- Term-distributed index
- Network remains the bottleneck

Document-distributed index

- Query is sent to distributed document indices
- Problem: query sent to all servers (scalability)
- But: sometimes needed: distributed surveillance videos indexed locally

Term-distributed index

- Processing nodes or servers maintain complete index information for a subset of terms of the collection vocabulary
- Each query is referred to a subset of the nodes that hold relevant information
- **Pipelined query evaluation**:
  - query is evaluated in stages by the sequence of nodes that hold the inverted lists corresponding to the query terms
  - **load balancing** based on dynamic load monitoring and estimation: making that some inverted lists that contain high-workload terms are duplicated

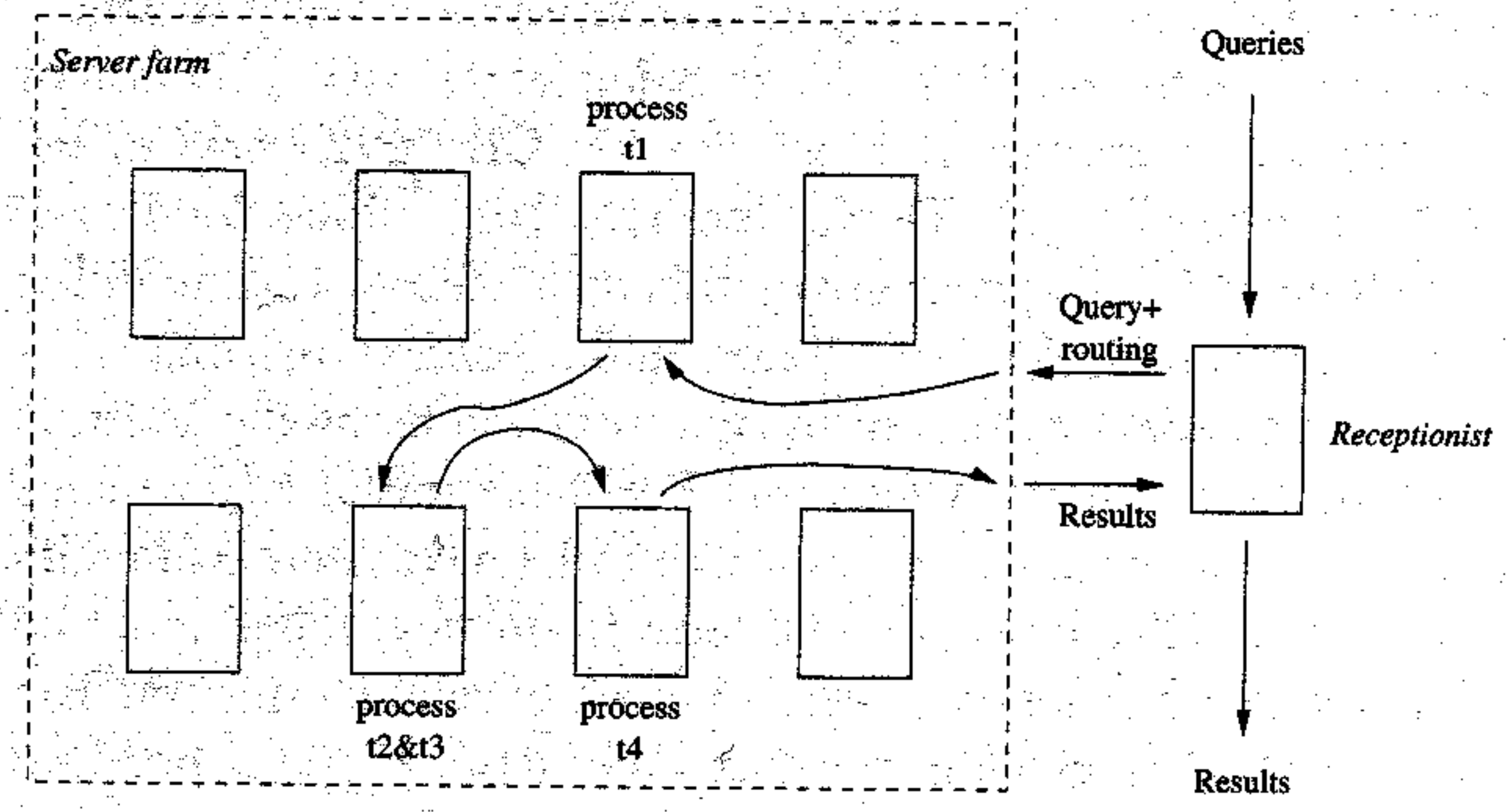


Figure 1: Example of pipelined query evaluation.

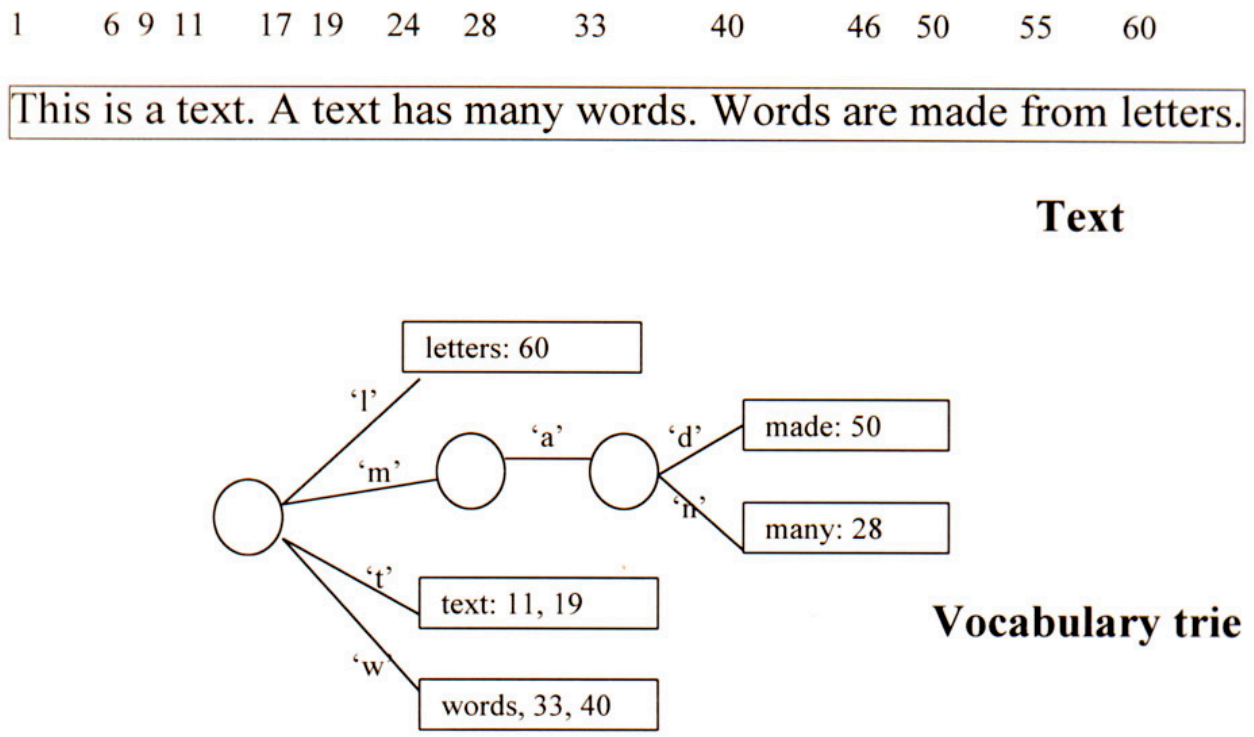
[Moffat et al. 2007]

Inverted files: searching

Search algorithm: three steps

1) **Vocabulary search**:

- queries with phrases and proximity queries are split into single words
- searching:
  - to speed up the search:
    - extra data structures: e.g., vocabulary trie, B-tree
    - hashing
  - searching lists of key terms in lexicographical order:
    - cheap in space
    - very competitive in performance: binary searched at sublinear search time: O (log n) where $n =$ the number of terms



Building an inverted index for the sample text

[Baeza-Yates & Ribeiro-Neto 1999]

Inverted files: searching

2) **Retrieval of occurrences**:

=> the list of all the documents in which the terms occur

3) **Manipulation of occurrences**:

- to solve phrases, proximity, or Boolean operations
- if block addressing is used it may be necessary to directly search the text to find the information missing from the occurrences (e.g., in proximity query)

Inverted files: searching

- Example phrase searching:
  - usually only single words and their positions are stored in the inverted file:
    - search for single word components of the phrase
    - identify documents in which the components co-occur in the right text order
  - alternatives:
    - store phrases in inverted file
    - combined inverted file and nextword index

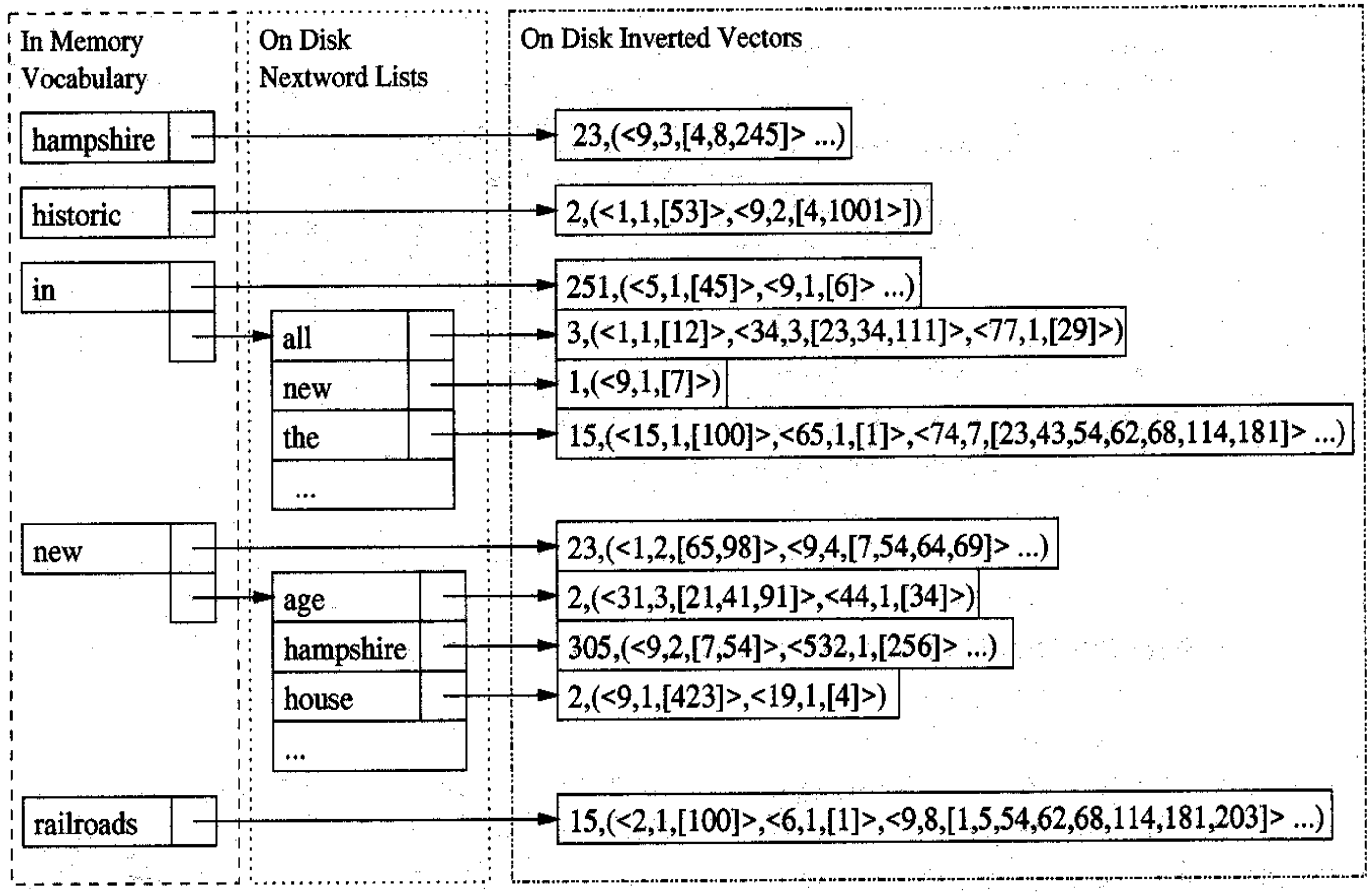


Figure 3: A combined inverted file and nextword index.

[Bahle et al. 2002]

Inverted files: space requirements

- Text databases:
  - vocabulary (unique terms): less than the size of document collection :
    - Heaps' law: vocabulary of size *n* grows as $O(n^{\beta})$ where $\beta =$ between 0 and 1 dependent on the type of text (often $\beta =$ between 0.4 and 0.6)
    - stopword removal, stemming and use of thesaurus terms further reduce the vocabulary
    - sublinear space requirement

Inverted files: space requirements

- occurrences demand more space, especially when text positions are included:
  - space depends on the number of documents (and positions within documents) that contain the term
  - in practice usually extra space requirements 30% to 40% of the text size
  - often compression is used for the occurrence pointers
- weights and other attributes demand extra space

Indices of structured documents

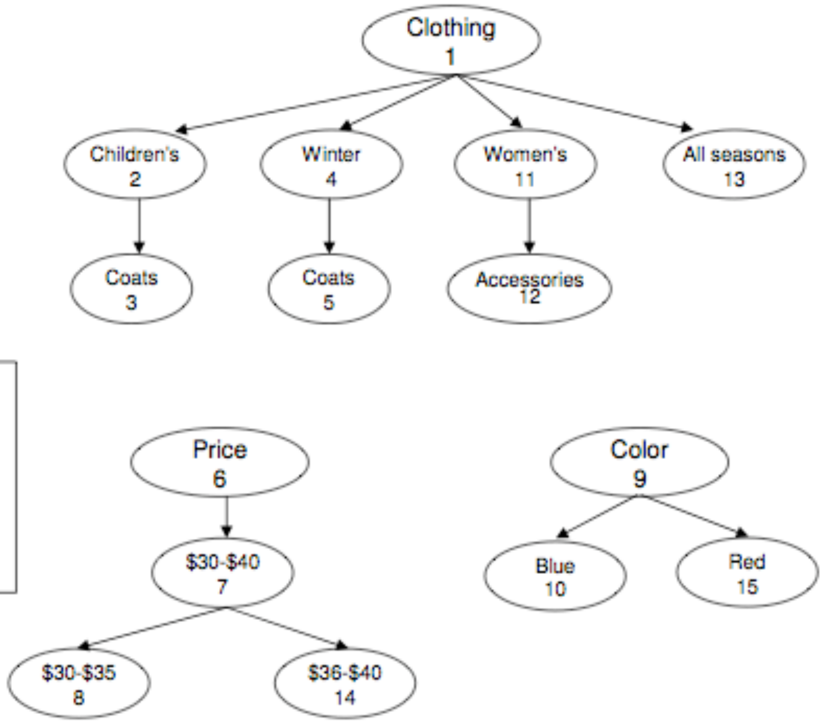
- Additional data about text regions, document structure (XML marked), extracted information, ...
- Example: taxonomy indices
- Inverted files complemented with extra tables (e.g., parent child relationships, positions of text regions and their semantic labels)

Doc 1

clothing/children's/coats
clothing/winter/coats
price/30-40/30-35
color/blue

Doc 2

clothing/women's/accessories
clothing/all seasons
price/30-40/36-40
color/red



The taxonomy index:
- Maps human readable labels of each taxonomy node
- Maps nodes to their ancestors
- Maps nodes to their children

[Lempel 2008]

<statute>^{0} <title 1>^{1} KB^{2} <chapter 1>^{3} algemene^{4} bepalingen^{5} <section 1>^{6} diefstal^{7} <article 1>^{8} voertuigen^{9} </article 1>^{10} alarm^{11} <article 2>^{12} voertuig^{13} </article 2>^{14} </section 1>^{15} <section 2>^{16} douane^{17} <article 3>^{18} douane^{19} controle^{20} </article 3>^{21} </section 2>^{22} </chapter 1>^{23} <chapter 2>^{24} voorzieningen^{25} <article 4>^{26} wet^{27} </article 4>^{28} <article 5>^{29} besluit^{30} </article 5>^{31} </chapter 2>^{32} </title 1>^{33} <title 2>^{34} diefstal^{35} onderdelen^{36} <article 7>^{37} radio^{38} </article 7>^{39} <article 8>^{40} banden^{41} </article 8>^{42} <article 9>^{43} auto^{44} onderdeel^{45} </article 9>^{46} </title 2>^{47} </statute>^{48}

Statute of which the structure is XML tagged: postings of tokens and labels. Only a few preprocessed text words are included. The numbers indicate the token positions in the statute.

Compression and searching

- Text compression: techniques for encoding text in fewer bits or bytes:
  - to reduce storage space, for faster transmission over communication channel, …
- Text compression in IR systems:
  - word is considered as symbol of alphabet of symbols
  - techniques:
    - for index compression
    - for text compression that allow searching the compressed text directly

Index compression

- Compression of e.g, position information, terms of vocabulary, etc., but query times on compressed or decompressed indices are reported to be roughly similar
- But useful if index is larger than main memory or cannot be buffered

What have we learned?

- Important: **inverted files**
  - sublinear search time and space requirements
  - very popular: found in most retrieval systems
- Compression:
  - research into direct indexing and searching on compressed text: could provide better time performance and less space overhead
- Searching in general:
  - still very much word based
  - is becoming more complex because of:
    - enhanced query facilities
    - **document indices that incorporate text structure and text semantics**

Research questions to be solved

- Coping with rich document indices (facets, taxonomies, semantic labels at sentence and discourse level, ...)
- Distributed architectures

Guaranteeing efficiency of the search !!!

Further reading

Baeza-Yates, R. & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. New York: ACM : Addison Wesley. (p. 180-207, summary of remainder of chapter 8).

Bahle, D., Williams, H.E. & Zobel, J. (2002). Efficient phrase querying with an auxiliary index. In *Proceedings of the Twenty-Fifth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 215-221). ACM: New York.

Brisaboa, N.R., Fariña, A., Navarro, G. & Paramá, J.R. (2007). Lightweight natural language text compression In *Information* Retrieval, 10, 1-33.

De Vries, A. P., List, J.A. & Blok, H.E. (2003). The Multi-model DBMS architecture and XML information retrieval. In H.M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel and G. Wekum (Eds.). *Intelligent Search on XML* (*Lecture Notes in Computer Science/ Lecture Notes in Artificial Intelligence* 2818) (pp. 179-192). Berlin: Springer.

Lempel, R. (2008). Towards task completion assistance in Web information retrieval. Slides of presentation at K.U.Leuven.

Moffat, A., Webber, W., Zobel, J. & Baeza-Yates, R. (2007). A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10, 205-231.

Partee, B. H., ter Meulen A. & Wall R. E. (1990). *Mathematical Methods in Linguistics*. Kluwer, Dordrecht.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Reading, MA: Addison-Wesley (concept of inverted files in chapter 8).

Silva de Moura, E., Ziviani, N., Navarro, G., & Baeza-Yates (1998). Fast text searching on compressed text allowing errors. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 298-306). New York: ACM.

Zobel, J. & Moffat, A. (2006). Inverted files for text search engines. In *ACM Computer Surveys*, 38 (2), Article 6 (http://doi.acm;org/10.1145/1132956.1132959).

IR tools to download:

- Lemur & Indri (http://www.lemurproject.org/)
- Lucene (http://lucene.apache.org/)
- Terrier (http://ir.dcs.gla.ac.uk/terrier/)
- Zebra (http://www.indexdata.dk/zebra/)
- Zettair (http://www.seg.rmit.edu.au/zettair/)