Searching with Inverted Files

Information Retrieval

Lecture 4

Motivation and Recap

- Users search the database with short queries
- Query components usually not present in every document
- Sequential search not efficient for large collections
- An *index* speeds up access by query term

Types of Queries

- Basic
  - set of *n* words
  - phrase, proximity, pattern
- Logical (Boolean)
  - basic queries joined with AND, OR, BUT
- Structural
  - basic queries keyed to document structures (sections, headers, hyperlinks)

Inverted Index

- Most common search structure for text
- Vocabulary
  - a.k.a dictionary or lexicon
  - set of all words in text
- Occurrences
  - a.k.a postings
  - each occurrence of the word in the text

How big is the index?

For an *n* word collection:

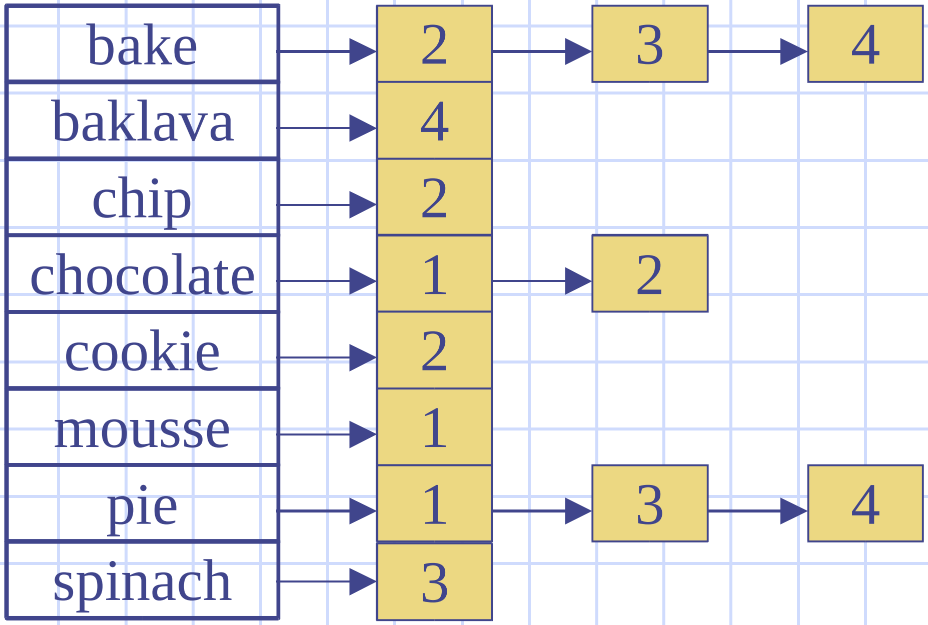
- Lexicon
  - Heaps' Law: $V = O(n^{\beta})$, $0.4 < \beta < 0.6$
  - TREC-2: 1 GB text, 5 MB lexicon
- Postings
  - at most, one per occurrence of the word in the text: $O(n)$

Inverted Search Algorithm

1. Find query elements (words, patterns, etc) in the lexicon
2. Retrieve postings for each lexicon entry
3. Manipulate postings according to the retrieval model

Inverted Search Example

1. Chocolate mousse pie
2. Chocolate chip cookies
3. Spinach Pie
4. Baklava

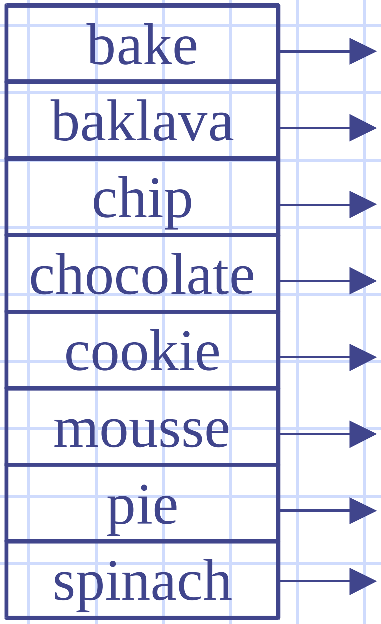


bake	2	3	4
baklava	4		
chip	2		
chocolate	1	2	
cookie	2		
mousse	1		
pie	1	3	4
spinach	3		

"I want to **bake** something with **chocolate**"

Lexicon

- Every query goes here first
  - keep in memory or a separate file
  - search should be fast
  - support prefix or pattern matching
  - support updating
- Each entry in the vocabulary has
  - the word
  - a pointer into the postings structure
  - word metadata



bake

baklava

chip

chocolate

cookie

mousse

pie

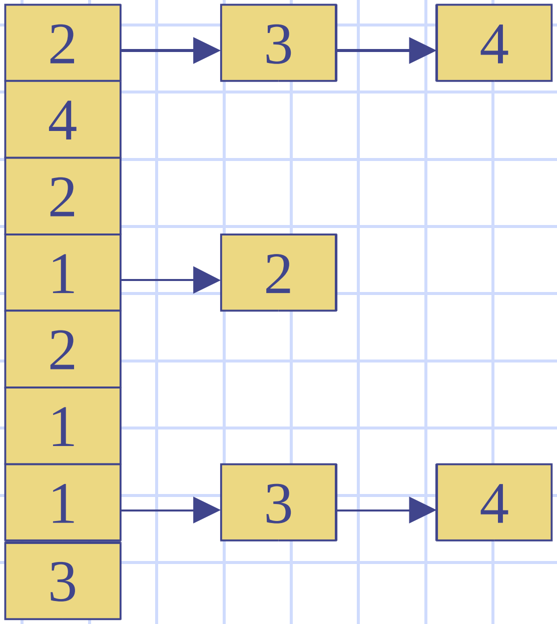
spinach

Lexicon Data Structures

- Hash table
  - O(1) lookup, with constant h() and collision handling
- Trie
  - O(c) lookup, c = length(word)
- B-Tree
  - On-disk storage with fast retrieval and good caching behavior

Postings

- Addresses of words in text
- Indexing *granularity*
  - character, word, document, logical block
- A posting usually holds
  - document ID
  - count in document
  - positions within document?



Inversion Example

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.

(from *Managing Gigabytes*)

In-memory Inversion

1. Create an empty lexicon
2. For each document d in the collection,
    1. Read document, parse into terms
    2. For each indexing term t,
        1. $f_{d,t}$ = frequency of t in d
        2. If t is not in lexicon, insert it
        3. Append $<d, f_{d,t}>$ to postings list for t
3. Output each postings list into inverted file
    1. For each term, start new file entry
    2. Append each $<d,f_{d,t}>$ to the entry
    3. Compress entry
    4. Write entry out to file.

Complexity of In-memory Inv.

- Time: O(n) for n-byte text
- Space
  - Lexicon: space for unique words + offsets
  - Postings, 10 bytes per entry
    - document number: 4 bytes
    - frequency count: 2 bytes (allows 65536 max occ)
    - "next" pointer: 4 bytes
- Is this affordable?

A Sample 5GB collection

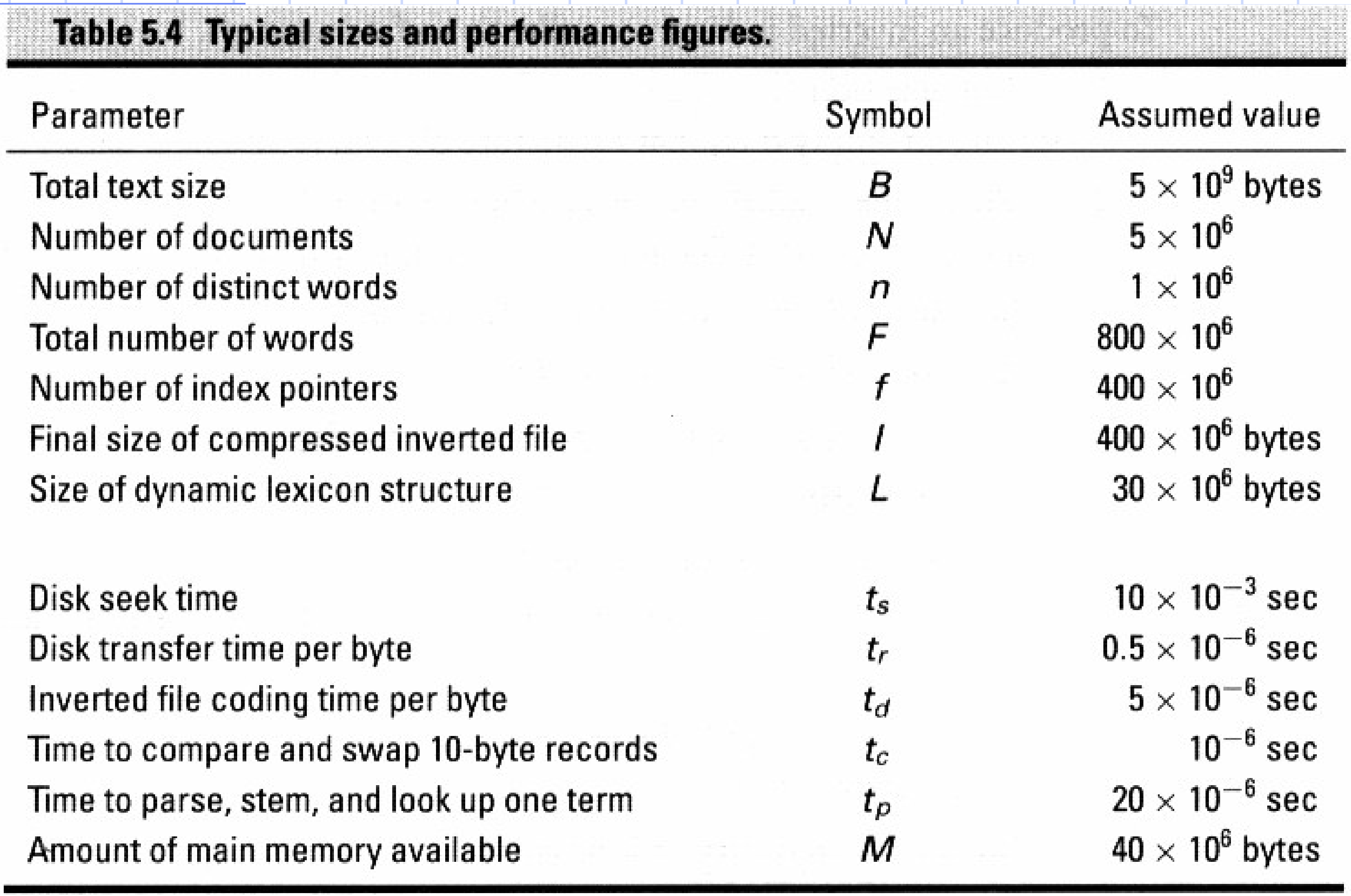


Table 5.4 Typical sizes and performance figures.

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

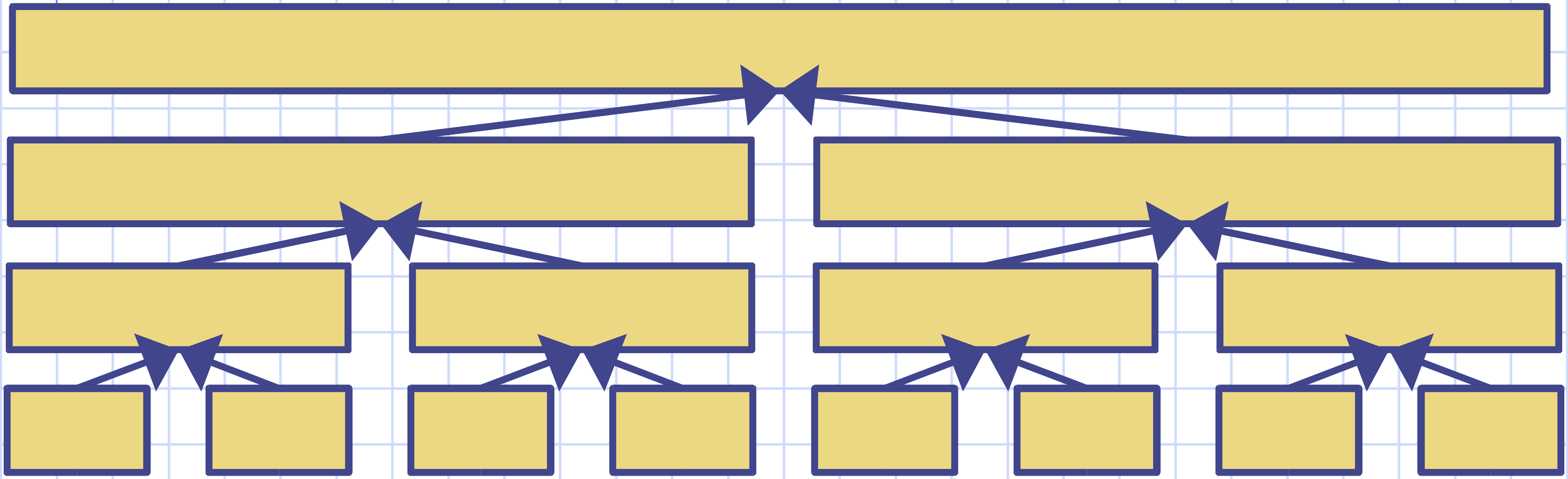
from *Managing Gigabytes*

Inverting the 5GB collection

- Time to invert in-memory
  - At 2MB/sec, ~40 minutes to scan 5GB
  - With parsing, stemming, lookup: 4 hours
  - Writing out inverted file: ~40 min
- Space required
  - at 10 bytes/entry, for 400M entries, need 4GB of main memory
- OK for small collections, not for large

Idea 1: Partition the text

- Invert a chunk of the text at a time
- Then, merge each sub-indexes into one complete index



Idea 2: Sort-based Inversion

- Invert in two passes
  1. Output records <t, d, f_t> to a temp. file
  2. Sort the records using external merge sort
     - read a chunk of the temp file
     - sort it using Quicksort
     - write it back into the same place
     - then merge-sort the chunks in place
  3. Read sorted file, and write inverted file

The Moral of the Story

- Indexing is something done rarely
- It pays to trade space for time
  - disk is cheap!
- It pays to use RAM and disk wisely
  - disk may be cheap, but disk access is expensive.