Information Retrieval
Topic- Index Construction (Hardware basics, BSBI)
Lecture-17

Prepared By

Dr. Rasmita Rautray & Dr. Rasmita Dash
Associate Professor
Dept. of CSE

Content

- Hardware basics
- Block Sort Based Indexing (BSBI)

Hardware basics

- Many design decisions in information retrieval are based on hardware constraints.
- We begin by reviewing hardware basics that we'll need in this course.

Hardware basics

- Access to data is much faster in memory than on disk. (roughly a factor of 10)
- Disk seeks are “idle” time: No data is transferred from disk while the disk head is being positioned.
- To optimize transfer time from disk to memory: one large chunk is faster than many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks). Block sizes: 8KB to 256 KB
- Servers used in IR systems typically have many GBs of main memory and TBs of disk space.
- Fault tolerance is expensive: It’s cheaper to use many regular machines than one fault tolerant machine.

Data collection

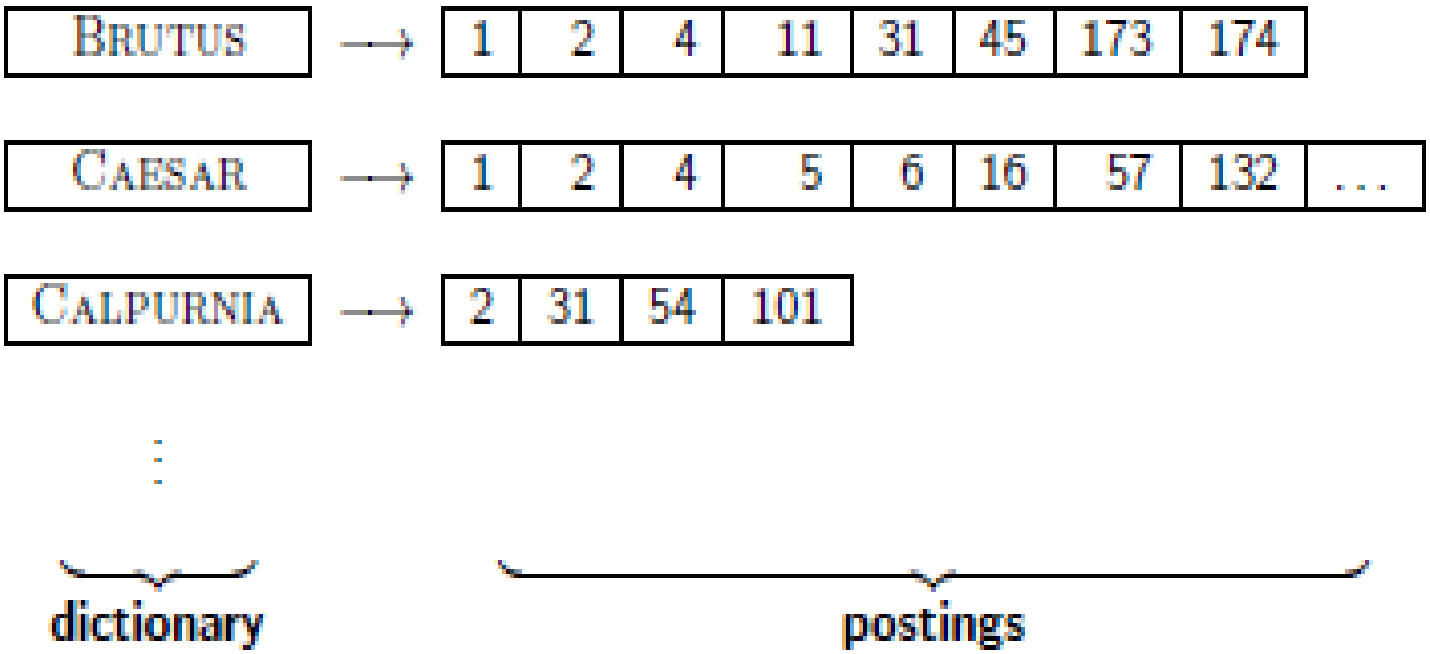
- Shakespeare's collected works are not large enough for demonstrating many of the points in this course.
- As an example, Reuters RCV1 (Reuters Corpus Volume 1) collection will be used for applying scalable index construction algorithms.
  - It is an English newswire articles sent over the wire in 1995 and 1996 (one year).

RCV1 statistics

N : documents	800,000
L : tokens per document	200
M : terms (= word types)	400,000
bytes per token (incl. spaces/punct.)	6
bytes per token (without spaces/punct.)	4.5
bytes per term (= word type)	7.5
T : non-positional postings	100,000,000

Index construction

Goal: construct the inverted index



Sort-based index construction

- As we build index, we parse docs one at a time.
- The final postings for any term are incomplete until the end.
- Can we keep all postings in memory and then do the sort in-memory at the end?
- No, not for large collections
- Thus: We need to store intermediate results on disk.

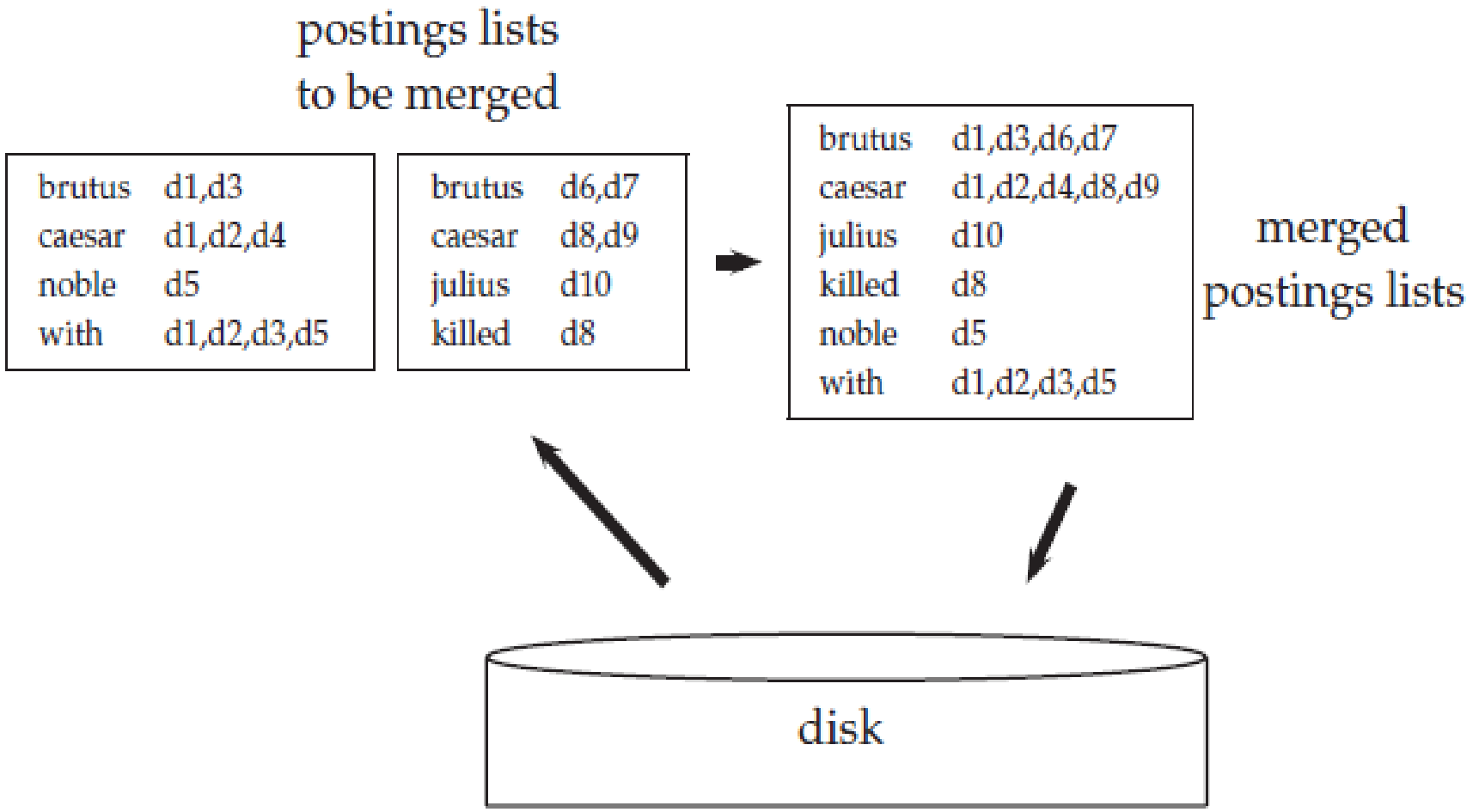
Same algorithm for disk?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?
- No: Sorting very large sets of records on disk is too slow – too many disk seeks.
- We need an external sorting algorithm.

“External” sorting algorithm (using few disk seeks)

- We must sort T = 100,000,000 non-positional postings.
  - Each posting has size 12 bytes (4+4+4: termID, docID, term frequency).
- Define a block to consist of 10,000,000 such postings
  - We can easily fit that many postings into memory.
  - We will have 10 such blocks for RCV1.
- Basic idea of algorithm:
  - For each block: (i) accumulate postings, (ii) sort in memory, (iii) write to disk
  - Then merge the blocks into one long sorted order.

Merging two blocks



Blocked Sort-Based Indexing

```
BSBINDEXCONSTRUCTION()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK(block, f_n)
7  MERGEBLOCKS(f_1, ..., f_n; f_merged)
```

Thank You