CHAPTER#4: INDEX CONSTRUCTION

★ Indexing ⇒ The construction of an inverted index and the machine that does it is called indexer.

★ Hardware Basics ⇒

↳ IR systems are mainly dependent on hardware.

↳ Access to data on memory is much faster than disk.

↳ Caching ⇒ keep frequently used disk data in main memory.

↳ Seek Time ⇒ When doing a disk read ~~seek~~ or write, it takes a while for the disk head to move to the part of disk where data is located. This time is called seek time and no data is transferred during this time.

better to transfer big chunks for less seek time

Dated:

↳ Buffer ⇒ The part of main memory where data is being read or written. OS reads & writes in terms of blocks since time to read or write a byte is same as a block.

↳ Data transfers are handled by the system bus. Thus, during disk I/O, the processor can be exploited to speed up data transfers by storing compressed data on disk.

★ Reuters-RCV1 collection ⇒ The collection being discussed in this chapter has the following stats:

Symbol	Statistic	Value
N	documents	800,000
Lave	avg. tokens per doc	200
M	terms	400,000
	avg. bytes per token (space+punct)	6
	avg. bytes per token	4.5
	avg bytes per term	7.5
T	tokens	100,000,000

★ Blocked Sort-Based Indexing ⇒

↳ Concept

↳ Accumulate postings for each block, sort, write to disk.

↳ Then, merge the blocks into one long sorted order.

↳ Algorithm

↳ The algorithm makes use of in-memory sorting because of speed,

↳ The first step is to decide block sizes that can be brought into memory.

↳ Segment the collection into blocks of equal sizes.

inversion { ↳ Sort the termID-docID pairs of each part in memory.
↳ store each intermediate block on disk.

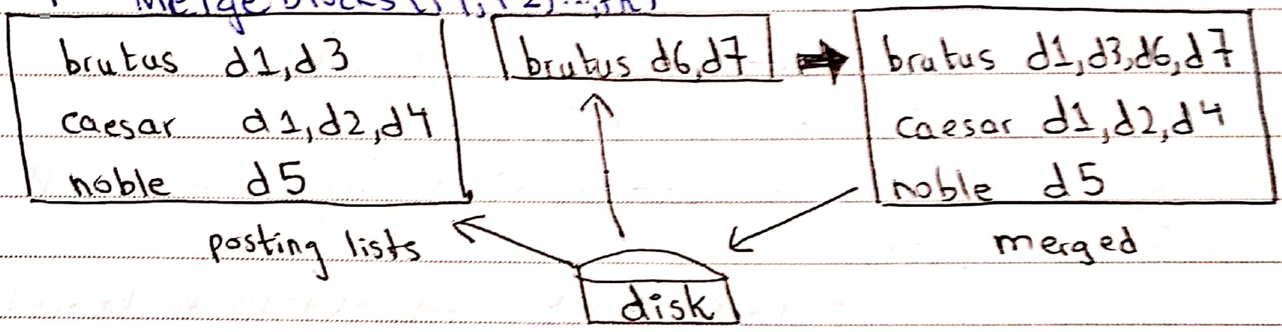
↳ merge all intermediate results into final index.

↳ read all lists at once then write one final block.

CamScanner

Dated: ___________

```
BSBIndexConstruction()
1   n ← 0
2   while (all docs have not been processed)
3       n ← n+1
4       block ← ParseNextBlock()
5       BSBI-Invert(block)
6       WriteBlockToDisk(block, f_n)
7   MergeBlocks(f_1, f_2, ..., f_n)
```



↳ Time Complexity
- ↳ Θ(T log T) because sorting dominates time and T is the number of items to sort.
- ↳ The indexing time is dominated by time to parse the docs and the final merge.

★ <u>Single-Pass In-Memory Indexing</u> =>

↳ Key Ideas =>
- ↳ Generate separate dictionaries for each-block individually.
- ↳ Don't sort, accumulate postings as they occur.
- ↳ now, we can generate a complete inverted index for each block.
- ↳ These indexes can be merged into one index.

↳ <u>Difference b/w BSBI and SPIMI</u>

SPIMI adds a posting directly to its postings list instead of collecting all termID-docID pairs and sorting them as in BSBI.

Dated:

↳Each posting list is dynamic and ~~it~~ it is immediately available to collect postings.
  ↳It is faster because of no sorting.
  ↳saves memory because we don't store term IDs.

SPIMI-INVERT(token_stream) → invoked per block

```
1   output_file = newFile()
2   dictionary = newHash()
3   while(free memory available)
4       token ← next(token_stream)
5       if term(token) ∉ dictionary:
6           postingList = addToDictionary(dictionary, term(token))
7       else
8           postingList = getPostingList(dictionary, term(token))
9       if full(postingList):
10          postingList = doublePostingList(dictionary, term(token))
11      addToPostingList(postingList, docID(token))
12  sortedTerms ← sortTerms(dictionary) } sorting by terms
13  writeBlockToDisk(sortedTerms, dictionary, output_file)
14  return output_file
```

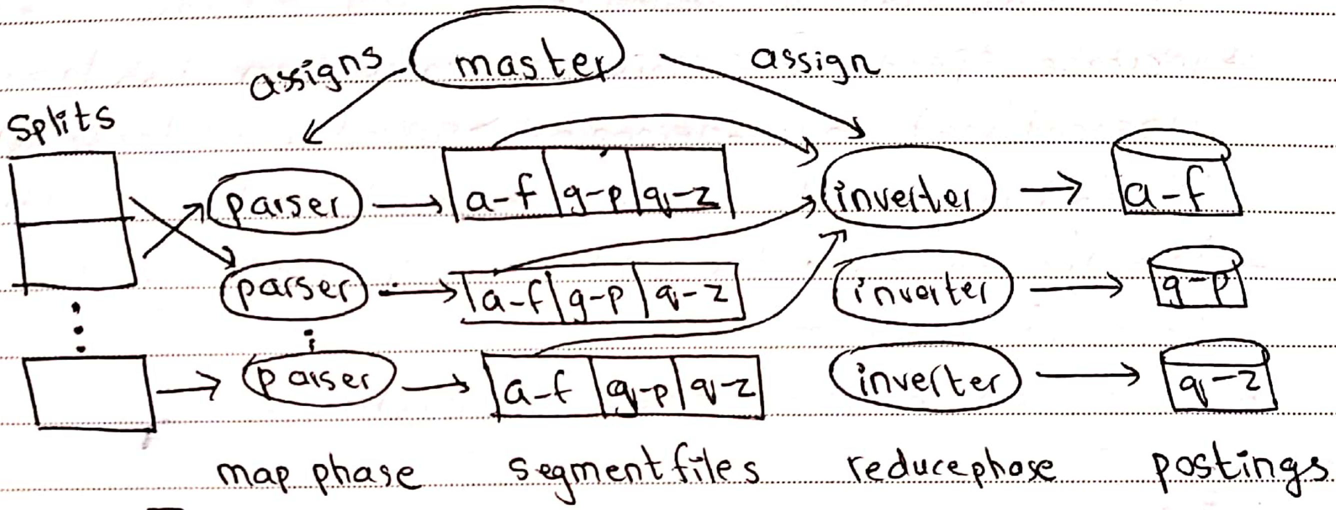
★ <u>Distributed Indexing</u> ⇒ → same PDC concepts

Collections are so large that index construction cannot be done on a single machine.

↳ <u>Map Reduce</u> ⇒ Distributed Indexing Algorithm
  ↳work assigned by <u>master node</u>.
  ↳master creates evenly distributed n splits (16-64 MB).
  ↳if a ~~for~~ machine finishes, it is assigned the next split else the task is reassigned.
  ↳ makes key-value pairs [termID → docID]
  ↳ <u>Map phase</u> maps splits into key-value pairs. Machines are called <u>parsers</u>.
  ↳ Intermediate files are called <u>segment files</u>.

Dated:

↳ Reduce phase collects segments files and collects & makes postings. Machines are called inverters. Each term partition is maintained by one inverter.



★ Dynamic Indexing ⇒

Documents collections are modified and deleted frequently so postings have to be updated.

The simplest way to acheive this is to periodically reconstruct the index if the number of changes are small and a delay in making docs searchable is acceptable.

If there is a requirement for documents to be included quickly, one solution is to maintain two indexes: a large main index and a small auxillary index which is kept in main memory. Searches are run across both and results merged. Deletions are marked with invalidation bits so deleted docs can be easily filtered it.

Each time the auxillary index becomes too large, we merge it into the main index.