11-02-2022

Date

Information Retrieval

- web crawlers

- extraction of essential required info from unstructured data

- Info Retrieval vs. Relational Database

User task ⟷ how to get rid of mice in correct way

↓

Info need ⟷ info about removing mice without killing them

↓

Query ⟷ how to trap mice alive

↓

Search engine ← Collection

↓

Results → how to "trap mice" alive

Query refinement ← more refined query

- Core concepts of IR
  - Query Representation
  - Document Representation
  - Retrieval Model

— Precision : fraction of retrieved docs that are relevant to user's information need $\frac{70}{100}$

— Recall : Fraction of relevant docs in collection that are retrieved $\frac{70}{150}$

Information Retrieval: Challenges

— data is unstructured

— query is unstructured

— computers can't guess person's intent

Date February 14th, 2022

- inferring relevance and intent from data

Unstructured data in 1620: (example)

- slow
- NOT Calpurnia non trivial (include 2, exclude 1)

but queries are not always easy.

- 'find words ~~~~ Romans near countrymen' not feasible, complex, 'near' is ambiguous.
- ranked retrieval (criteria according to which ranking is done)

Term-document incidence matrices

- makes it easier to search
- easy to ~~insert~~ add more plays, just one more column

docs

play

1 — contains word

0 — play doesn't contain word

terms

Incidence Vectors: ← not efficient for large data collection will make extremely sparse matrix

Brutus	1 1 0 1 0 0	AND
Caesar	1 1 0 1 1 1	AND
NOT Calpurnia	1 0 1 1 1 1	= ← after complement of Calpurnia
	1 0 0 1 0 0	← plays containing Brutus and Caesar but not Calpurnia

Solution: if a word is not present in a doc. don't add/enter it. But can't be a matrix now.

Date February 15th, 2022.

Inverted Index

backward index → works with modern IR systems

- each doc has a doc ID

Dictionary | Brutus | 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174 | ← Posting (sorted)

- fixed size array can't be used because it can't grow, some positions may be empty and wasted so we use linked list.

- need variable size postings lists ← sorted by doc ID

dictionary: Brutus ⇨ . . . . ← posting linked list
Caesar ⇨ . . . .
Calpurnia ⇨ . . . .

dictionary → in memory
Postings → in disk
pointers

Inverted Index Construction

Documents → Tokenizer → Linguistic modules (stemming) → Indexer

Friends, Romans, Countrymen → Friends | Romans | Countrymen → friend | roman | countryman → friend ⇨ 2→4→ ; roman ⇨ 1→2→ ; country man ⇨ 13→16

Initial stages of Text processing

- tokenization
- normalisation
- Stemming
- Stop words

① Indexer Steps : Token Sequence
add all words, even repeated ones

② Sort alphabetically and then doc ID, remove repeated entries

caesar 1 → caesar 1
caesar 1 → caesar 2
caesar 2

Caesar, 2 → 1 | 2 (linked list)

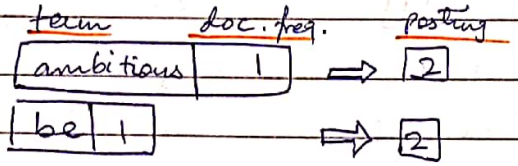
2 = total docs in which it occurs

Date

- Sorted on Terms firstly so repeated terms come together ← takes linear time for computer to search

Dictionary + Postings

- multiple entries are merged in a single document
- split into dictionary + postings



Issues of Storage: term, doc. freq. (pointers,) list in memory
postings in disk / permanent storage
(take it in memory when using) need be

Query Processing w/ inverted index

AND: Brutus AND Caesar
retrieve postings and merge / intersection.

len x	Brutus	2	4	8	16	32	64	128
len y	Caesar	1	2	3	5	8	13	21	34

set pointers on start, no match? take pointer to next ~~to~~ where number is small.

$O(x+y)$
one sorted

if not sorted then $x*y$
$O(xy)$

Algorithm in slides (33 pg)

DIY
take 5 docs, make inverted index, do linear search, do query search

Slide Set 02.

Date February 22nd, 2022. 22-2-2022

Example: Westlaw Boolean Search

- What's statute of limitations in cases involving federal tort claims act?

=> LIMIT! /3 STATUTE ACTION /S FEDERAL /2

LIMIT! ↙ word should have limit, followed by anything i.e. limited, limits, limitation

/3 ↓ within 3 words

STATUTE ACTION ↳ OR

/S ↓ in same sentence

TORT /3 CLAIM /P same paragraph

- disjunction OR (space) — conjunction AND

- Brutus and Not Caesar: O(x+y) can be done in linear time different? add in list, same? pop

- Brutus or not Caesar: DIY.

Brutus ∪ (not Caesar) can be in linear time

- Merging combine 2 lists, OR.

AND: first consider list with min elements (check from freq. list) then take intersection (AND) with next min list then next.

Slide 4

* rhyme zone. com shakespeare with search features you think it could do better.

Date 24th February, 2022.

Phrase Queries

"Stanford university" ← both words should come in this order like this.

- phrases can't be stored in inverted indexes. <term : docs> ×
  maybe standard & uni come in a document but inv. index doesn't tell if both occur together or not.

Solution 1: Biword Indexes

excluding stop words from phrases

stanford, 3	1, 3, 5
university, 5	1, 5, 11, 16, 23
stanford uni, 1	5
international cricket	
cricket council	

stanford uni {1, 5}
↙ ↓
don't occur together | occur together

Bigrams:
Stanford university international cricket council

- Stanford university palo alto ← Longer queries can't be solved with biword indexes

stanford uni AND uni palo AND palo alto
{1, 5, 7, 11} {2, 3, 5, 7} {5, 11, 22} → {5}

can give false positives as no assurance whether these 3 bigrams occur together in doc or not which we want.

- We need to have positions stored with each word.

Issues : ① false +ves ② can be part of a compound strategy

Date

Solution 2: Positional Indexes → example in slides.

- not for biwords, only for single terms
- <term, no. of docs it occurs in;
  doc1 ← 1: 7, 18 → position in doc
  doc2 ← 2: ...
  ⋮
  99 3427: ... >
- documents every occurance in the doc.

- proximity queries: positional indexes can be used, biword indexes cannot
- positional index size depends on avg. document size.
- pos. index > 2-4 Times than non-pos. index
  pos. index size 35-50% of volume of original Text → pos. index is faster than linear search

Combination Schemes

- Positional + Bigram indexes in a hybrid scheme
- Study / Paper showed it takes $\frac{1}{4}$th of time (faster) but 26% more space.

watch video

Date March 1st, 2022.

RCV1 dataset ← UCI repo

token ← words + stop words

Terms ← only words.

Registers
Caches
Main memory/RAM
HDD/secondary SSD storage
(speed decreases)

avg. no. bytes per token < avg. no. of bytes per Term

↓ (stop words: 'was', 'for', 'the' etc)

↓ (only words, short words 'for', 'the' are removed so average size increases.)

- Hardware Stuff

Index Construction

Sort based Index Construction

(termID, docID) ← 8 bytes
(4B, 4B)

Term1 → 1
Term2 → 2
⋮

school → 1
form → 2
⋮

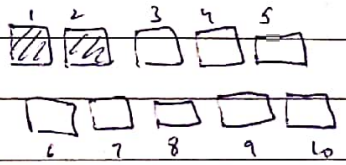
If T = 100,000,000 docs → 800,000,000 bytes
/1024
/1024
= 0.75 GB

Scaling Index Construction

BSBI

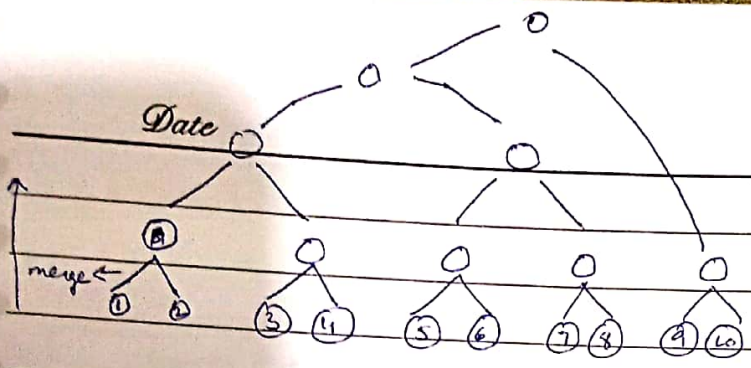
T = 100,000,000 → too big to read in memory
so divide in 10 chunks of 10,000,000 each

(1) 8 byte (TermID, docID) → 100,000,000
(2) Now have to sort 100,000,000 by TermID
(3) divide T and make chunks
(4) sort each chunk



Code in slides

now 1, 2 are merged, after this sort the merged



LON: if ① and ② are 10,000,000 so 20M, 000,000 needs to be loaded in memory

How to merge sorted runs?

~~take~~ 5 million chunks each of 10 million, merge them (will take 2 secs)

Multi-way merge

Another way: Take all 10 chunks of 100,000,000 at once and process a part of all 10 chunks. like 2 million from each 10 chunks. Equal to 20 million

Single Pass in Memory Indexing

1. generate separate dics for each 10 blocks
   no Term – Term ID × only Term – DOCID
2. don't sort. accumulate postings in postings list
3. inverted index for each block
4. sparate indexes can be merged into one big index

Code in Slide

Date March 3rd, 2022.

SPIMI : Compression

Store index in Main memory. If too large then store in secondary storage. If it's too large then distributed storage. Put it in multiple nodes (systems).

Distributed Indexing:

- one master machine : directs indexing job
  important + safe
- master divides indexing tasks + assigns it to idle machines from pool

Parallel Tasks

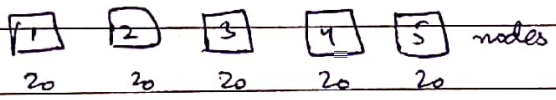
- (1) Parsers (2) Inverters

after parsing we get : (Term, docID) pairs

sort (term, docID) → postings → inverted index

- split doc. collection, each split is a subset of doc (corresponding to blocks in BSBI/SPIMI).

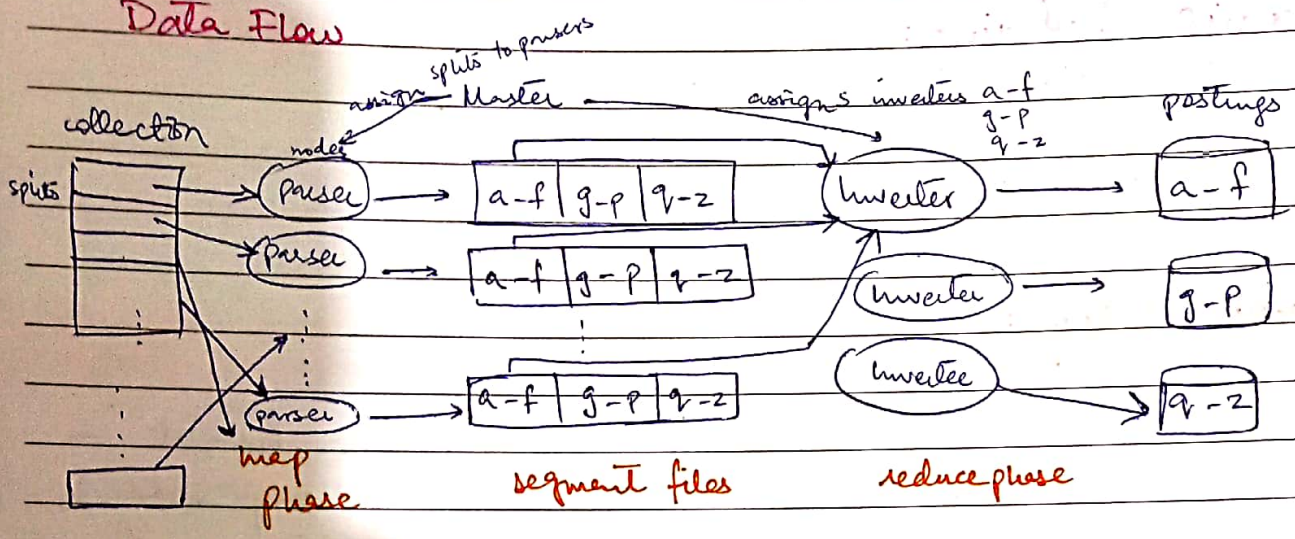
100 docs



(1) first parse, each node does it for their own 20
it's possible >1 nodes have same terms so we have to combine all terms afterwards

Date

Data Flow



map phase $Map(f, [a_1, a_2 \cdots a_n]) \quad f(a_1) \to f(a_2) \cdots f(a_n)$

f ↓ parser $[a_1, a_2 \cdots a_n]$ ↓ collection

reduce phase $Merge(a\text{-}f_{p_1}, a\text{-}f_{p_2}, \cdots a\text{-}f_{p_n})$

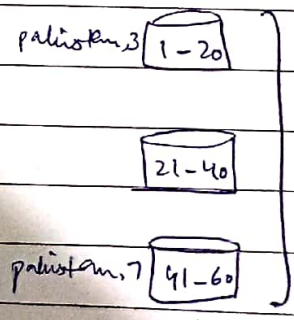
Parsers

$j = 3$ partions a-f, g-p, q-z

problem w/ postings based on alphabetic Terms: issues with balancing of loads. if everybody is searching for pakistan then [g-p] will have too much load

solution: do partition on bases of doc. IDs.

Postings



now load is distributed

MapReduce

application: Hadoop

map: collection → list (TermID, docID)

reduce:

example in slides

Date

Dynamic Indexing:

before this collection was static

March 8th, 2022. (watch video)

Logarithmic Merge:

Z_0 : put terms in it, when reaches n length. Put it in disk with name I_0. Again build Z_0 (auxillary index), it merges with I_0, I_0 gets erased, now I_1 is built and stored in disk.

Algo in slides.

Issues w/ multiple indexes:

incidence matrix
inverted index
- pos. indexing
- boolean queries

Date March 15th, 2022.

Scoring, Term Weightage, Ranking

• Ranked Retrieval:

- Jaccard coefficient:

$$\text{jaccard}(A, B) = |A \cap B| / |A \cup B|$$

$$(A, A) = 1$$

$$(A, B) = 0 \text{ if } A \cap B = 0$$

- A + B may not have same size
- gives value b/w 0 + 1

CON: doc. size affects the score

- rare terms are more informative

• Query-doc matching scores

- assign score to query/doc pair
  - if does not occur in doc : score 0
  - if frequency ↑, score ↑

- in index : replace 1 with freq count of terms in a doc (vector representation)

Bag-of-words Model:

- in vector representation, order of terms occurring is not considered

Term frequency (tf): $tf_{t,d}$ → term freq of term t in doc d

Raw term freq:

	doc 1	doc 2
school:	1	10

← This doesn't mean doc2 is 10x more relevant than doc1. It only means doc 2 is more relevant

- relevancy doesn't increase proportionally with tf.

tf → Term freq
idf → doc freq

Date

used it so we get weights in smaller range so ease in calculations.

Log-frequency weighting:

$$W_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- now put weights in index

→ calculate score for only common words b/w query + doc.

$$\text{Score} = \sum_{t \in q \cap d} (1 + \log_{10} tf_{t,d})$$

- now put scores in vector index style
- tf is giving us score

(-idf) tf-idf

idf weight: → inverted style

- df_t → doc. frequency of t: no. of docs that contain t

→ total no. of docs

$$idf_t = \log_{10} \left(\frac{N}{df_t} \right)$$

→ how many docs contain the term

$\frac{100}{5} = 20$ idf = 2 → giving more weight to more rare doc

$\frac{100}{50} = 2$ idf = 0.3

- capricious person
  - tf ↓ idf ↑ → occ rare
  - tf ↑ idf ↓ → occ common

$$W_{t,d} = 1 + \log(tf_{t,d}) \times \log_{10}(N/df_t)$$

- 1 + log(tf): ↑ if freq increases of a term
- tf
- idf
- ↑ if rarity increases of a term