Information Retrieval

Amin Nazari

Spring 2025

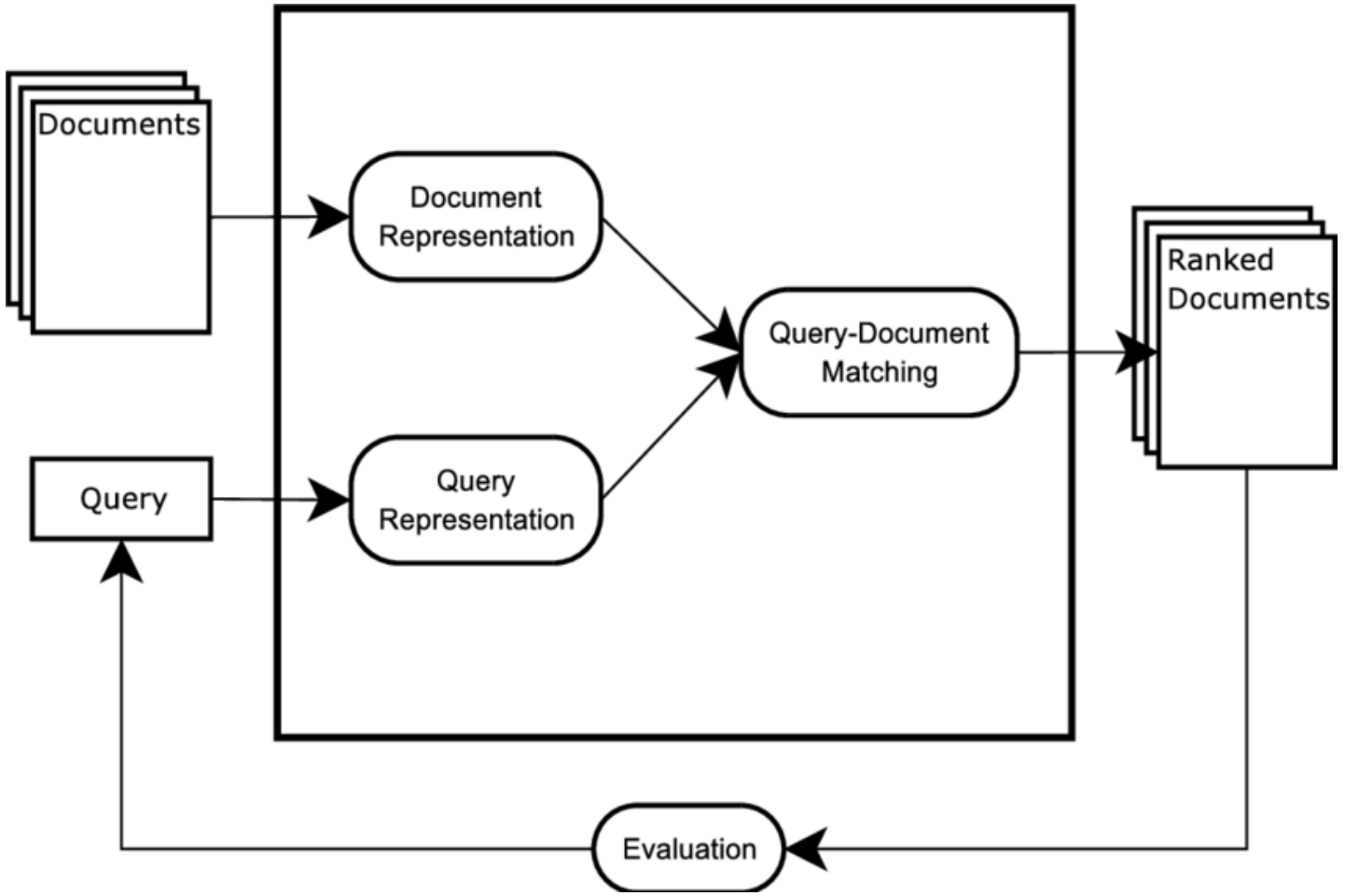
Chapter 1

Boolean retrieval

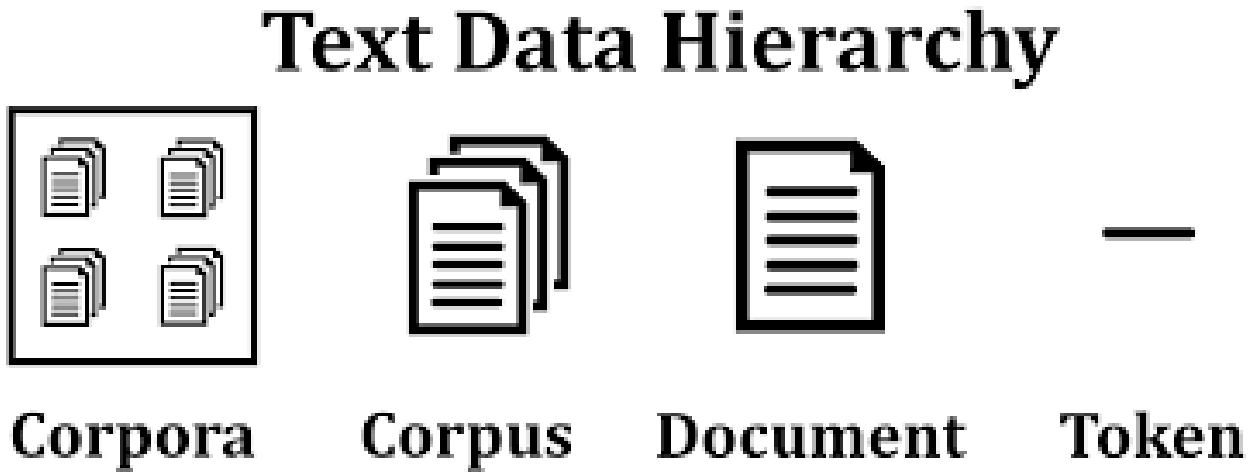
Outline

- Boolean retrieval
- Inverted index
- Query optimization
- Skip pointers
- Implementation in Python

Recall IR schema



Terms



Boolean retrieval

1. Vocab
2. Incidence Matrix
   - Term-Document Boolean representation
3. Operation (AND, OR, NOT)
4. Retrieval.

Boolean retrieval(Term-document incidence matrix)

- Vocab = {the, red, dog, cat, eats, food}

	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Example

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
. . .						

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
  - Take the vectors for BRUTUS, CAESAR AND NOT CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) and on the three vectors
  - 110100 AND 110111 AND 101111 = 100100

0/1 vector for BRUTUS

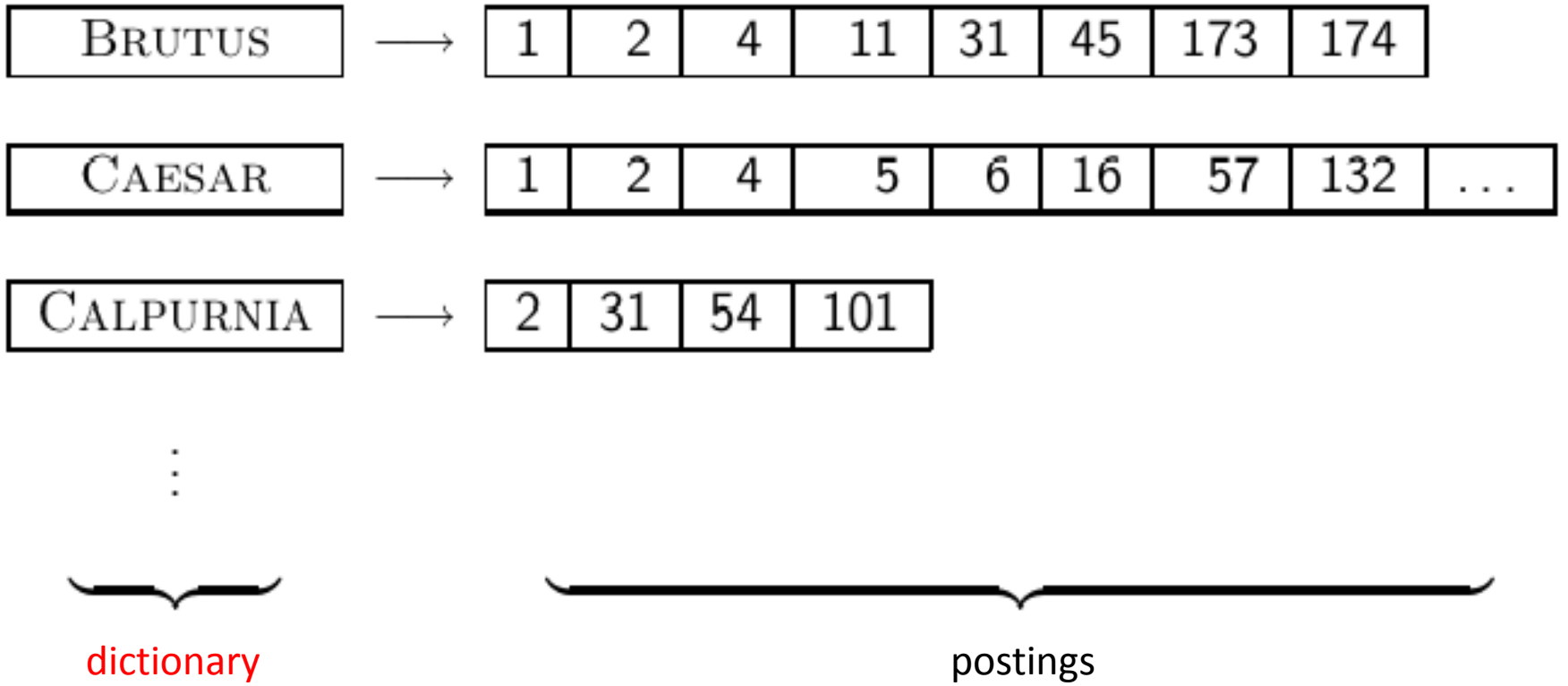
	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
. . .						
Result:	1	0	0	1	0	0

Bigger collections

- Consider N = 10^6 documents, each with about 1000 tokens ⇒ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation ⇒ size of document collection is about 6 * 10^9 = 6 GB
- Assume there are M = 500,000 distinct terms in the collection (Notice that we are making a term/token distinction.)
  - M = 500,000 × 10^6 = half a trillion 0s and 1s.
  - But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
  - What is a better representations?
    - We only record the 1s.

Inverted Index

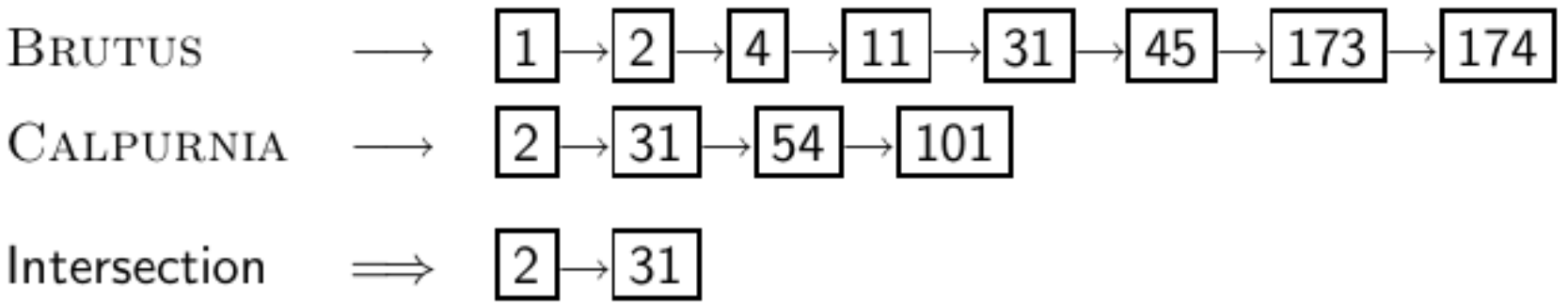
- For each term *t*, we store a list of all documents that contain *t*.



Inverted Index Retrieval

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
    1. Locate BRUTUS in the dictionary
    2. Retrieve its postings list from the postings file
    3. Locate CALPURNIA in the dictionary
    4. Retrieve its postings list from the postings file
    5. Intersect the two postings lists
    6. Return intersection to user

Inverted Index Retrieval



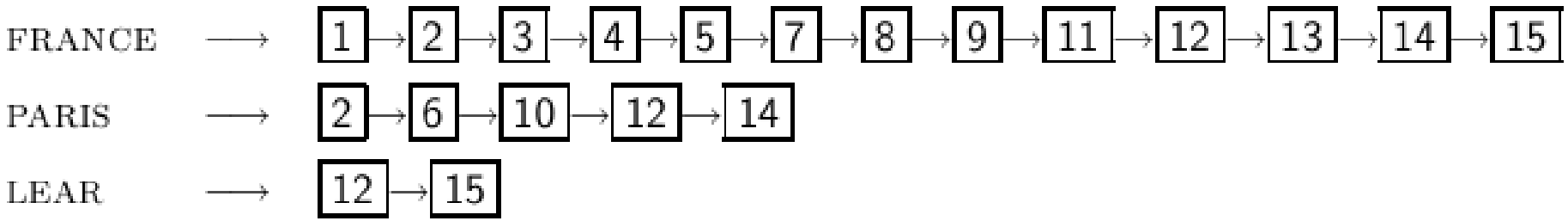
Intersect Algorithm

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Query processing: Exercise

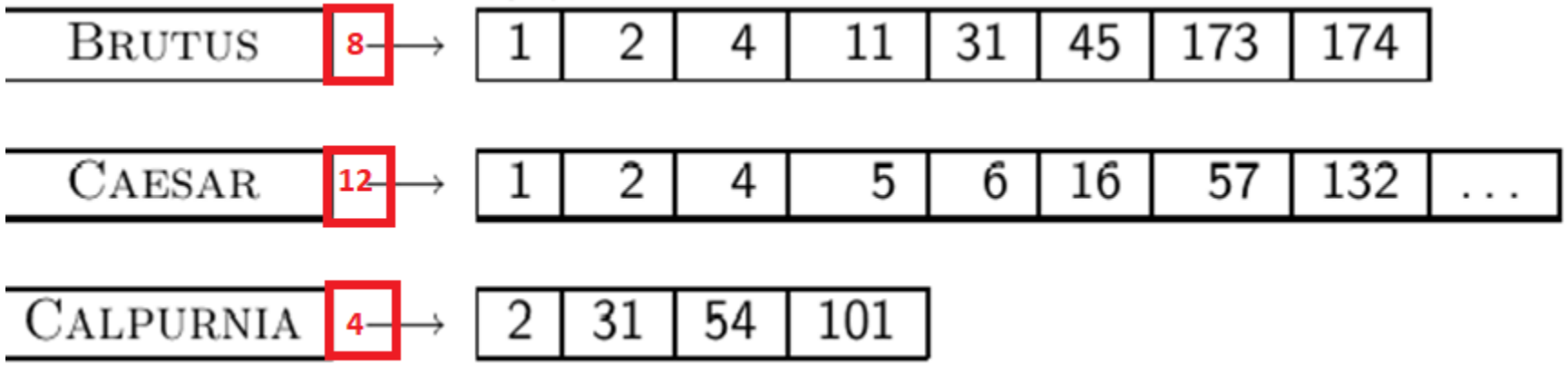
- Compute hit list for ((Paris AND NOT France) OR Lear)



Query optimization

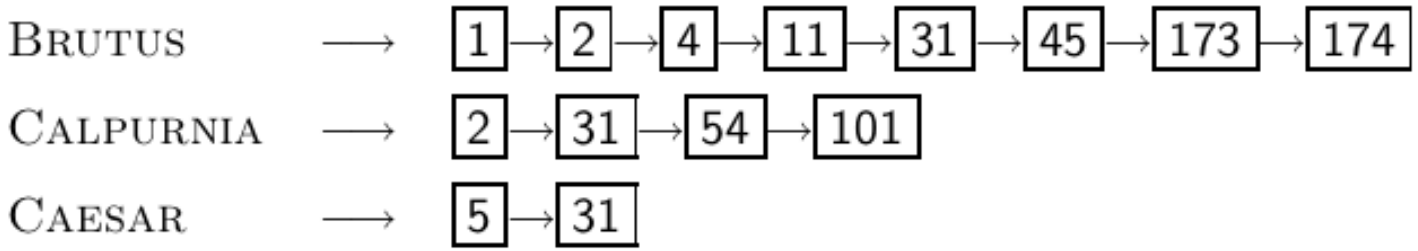
- Consider a query that is an and of n terms, n > 2
- For each of the terms, get its postings list, then AND them together
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- What is the best order for processing this query?

Create postings lists, determine document frequency



Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: Process in order of increasing frequency
- Start with the shortest postings list, then keep cutting further
- In this example, first CAESAR, then CALPURNIA, then BRUTUS



Optimized intersection algorithm for conjunctive queries

```
INTERSECT(⟨t_1, ..., t_n⟩)
 1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
 2  result ← postings(first(terms))
 3  terms ← rest(terms)
 4  while terms ≠ NIL and result ≠ NIL
 5  do result ← INTERSECT(result, postings(first(terms)))
 6     terms ← rest(terms)
 7  return result
```

More general optimization

- Example query: (MADDING OR CROWD) and (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each or by the sum of its frequencies (conservative)
- Process in increasing order of or sizes

Python codes

- All codes are available on GitHub
- Boolean Retrieval
- Inverted Index Retrieval
- Intersect
- Difference
- Union
- Optimal Query

Summary

- The Boolean retrieval model can answer any query that is a Boolean expression.
  - Boolean queries are queries that use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
- Many search systems you use are also Boolean: spotlight, email, intranet etc.

Remaining challenges

- Tokenizing
  - CAT, Cat, cat, CATS, Cats, cats all are one token.
  - Los Angeles, چهارمحال بختیاری, یک قل دو قل, each are one token
- Large Corpus:
  - How can we create inverted indexes for large collections?
  - How much space do we need for the dictionary and index?
- Ranked retrieval: what does the inverted index look like when we want the “best” answer?