Introduction to Information Retrieval

Session 3: **Boolean Retrieval**

Instructor: Behrooz Mansouri
Fall 2022, University of Southern Maine

Previous Session

In previous session we learned about:

- ✓ Coding in python
- ✓ Using Google Colab
- ✓ Revisited programming concepts in Python

Boolean Retrieval

Why Boolean?

Named Boolean Retrieval as

- There are two possible outcomes: TRUE (Document matches the Query) or FALSE
- Query is specified using Boolean logic operators: AND, OR, NOT

Exact-match Retrieval

Named Boolean Retrieval as

- There are two possible outcomes: TRUE (Document matches the Query) or FALSE
- Query is specified using Boolean logic operators: AND, OR, NOT

Also known as exact-match retrieval

- Documents are retrieved if they exactly match the query specification
- Otherwise are not retrieved

Is precision always 1?

Boolean Retrieval Results

Named Boolean Retrieval as

- There are two possible outcomes: TRUE (Document matches the Query) or FALSE
- Query is specified using Boolean logic operators: AND, OR, NOT

Also known as exact-match retrieval

- Documents are retrieved if they exactly match the query specification
- Otherwise are not retrieved

Not considered as ranking. Why?

- All documents in the retrieved set are equivalent in terms of relevance (no particular order)
  - Search systems with ranking, rank good hits (according to some estimator of relevance) higher than bad hits

Advantages and Disadvantages of Boolean Retrieval

Advantages

- The results of the model are very predictable and easy to explain to users
- The operands of a Boolean query can be any document feature, not just words
    - Straightforward to incorporate metadata such as a document date or document type in query
- From an implementation point of view, Boolean retrieval is usually more efficient than ranked retrieval

Advantages and Disadvantages of Boolean Retrieval

Advantages

- The results of the model are very predictable and easy to explain to users
- The operands of a Boolean query can be any document feature, not just words
  - Straightforward to incorporate metadata such as a document date or document type in query
- From an implementation point of view, Boolean retrieval is usually more efficient than ranked retrieval

Disadvantages

- Because of the lack of a sophisticated ranking algorithm, simple queries will not work well
  - Results can be sorted in some way (publication date), but not ranked
- Null outputs because of exact matching
- Complex query syntax (in earlier systems)

Document as Bag of Words

A bag or a multiset is an unordered collection (a set that can contain more than one instance of each element)

Documents are ‘bags of words’ means word order is ignored

A “bag of words” retrieval system treats the following documents identically:

- man bites dog
- dog bites man
- dog man bites

“Bags of words” models can be surprisingly good. Words alone tell us a lot about content

Boolean Retrieval considers document as a set of words

Boolean Retrieval (Shakespeare Example)

Searching Over Shakespeare’s Plays

Consider Shakespeare's plays: Anthony and Cleopatra, Julius Caesar, The Tempest, Hamlet, Othello, and Macbeth

We are interested in knowing which play contain Brutus and Caesar and not Calpurnia in them

How would you do this?

The Assassination of Julius Caesar by William Holmes Sullivan, c. 1888, Royal Shakespeare Theatre

Grep Style

Grep all Shakespeare’s plays for Brutus and Caesar; remove lines having Calpurnia

What are the problems with this approach?

Grep Style

Grep all Shakespeare's plays for Brutus and Caesar; remove lines having Calpurnia

What are the problems with this approach?

- Slow for large corpora
- Calculating "NOT" requires exhaustive scanning
- Grep is line-oriented, IR is document-oriented

Term-document Incidence Matrix

Word \ Document	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

Entry is 1 if term occurs. e.g., Calpurnia occurs in Julius Caesar

Entry is 0 if term does not occur. e.g., Calpurnia does not occur in The Tempest

Incidence Vectors

Each term, a vector of 0/1

Word \ Document	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0

Query: Brutus and Caesar and not Calpurnia

Which documents should be retrieved?

Incidence Vectors

Brutus and Caesar and not Calpurnia

Word \ Document	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0

Not (Complement)

Word \ Document	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	1	0	1	1	1	1

Incidence Vectors

Query: Brutus and Caesar and not Calpurnia

Word \ Document	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	1	0	1	1	1	1

Do a bitwise And on the three vectors

	Anthony and Cleopatra	Julius Caesar	The Tempest	**Hamlet**	Othello	Macbeth
Result	1	0	0	1	0	0

Incidence Vectors

Word \ Document	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

Find documents containing MERCY, WORSER and not ANTHONY

Issue with Current Design

Consider $N=10^6$ documents, each with about 1000 words → total of 10^9 words

- On average, 6 bytes per word
- Including space and punctuation
- Size of document collection is ~ 6.10^9 = 6 GB

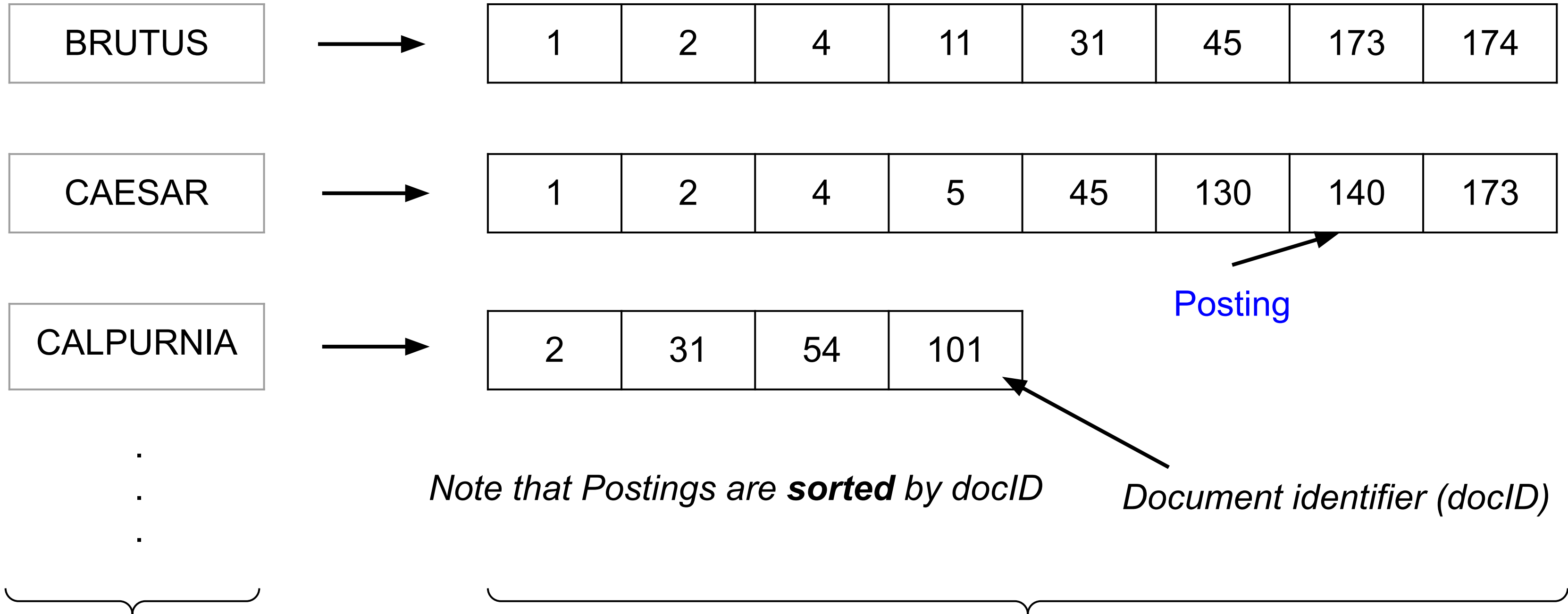
Assume there are M=500,000 distinct words in the collection

- 10^6 x 500,000 of 0s and 1s
- There are fewer number of 1s → sparse matrix

How about just recording 1s?

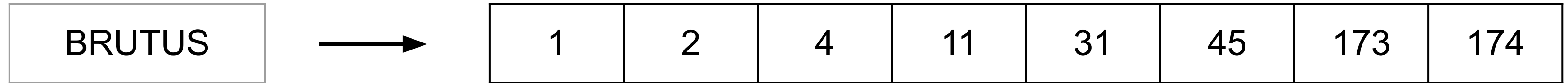
Inverted Index

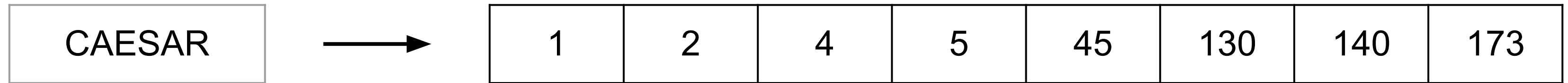
For each term(word) t, we store a list of all documents containing the term t (multiple occurrence merged)



Inverted Index

For each term(word) t, we store a list of all documents containing the term t (multiple occurrence merged)





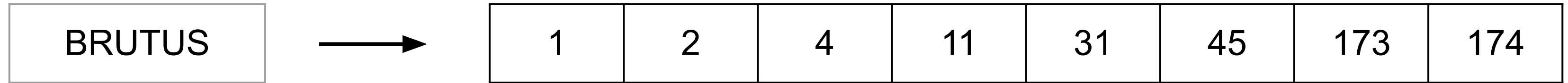
linked lists or variable length arrays

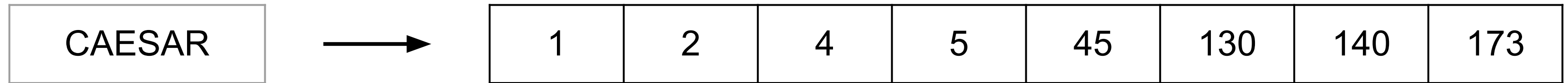
Dictionary (Vocabulary) *Usually kept in memory with pointers to each postings list*

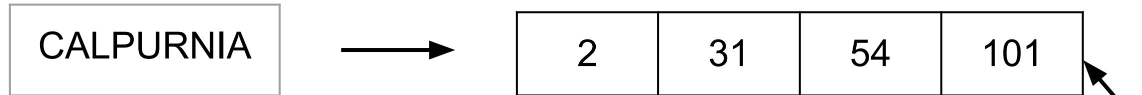
Postings *Usually kept in disk*

Inverted Index

For each term(word) t, we store a list of all documents containing the term t (multiple occurrence merged)







What does the length of postings (document frequency) tell us?

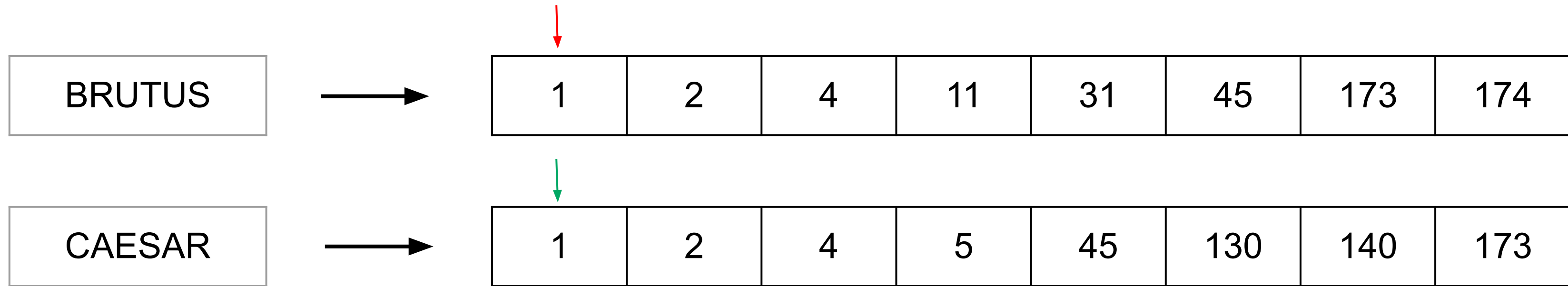
.
.
.

Dictionary (Vocabulary)

Postings

Processing the Query

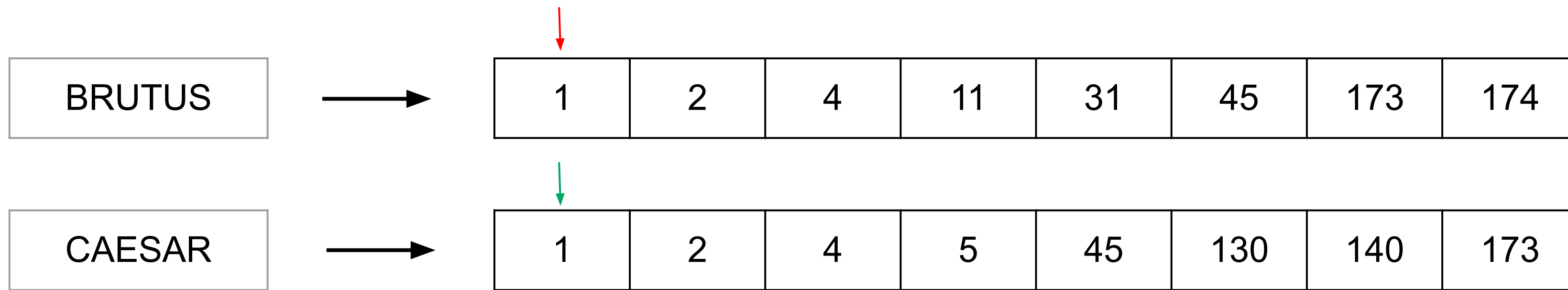
Query: Brutus and Caesar



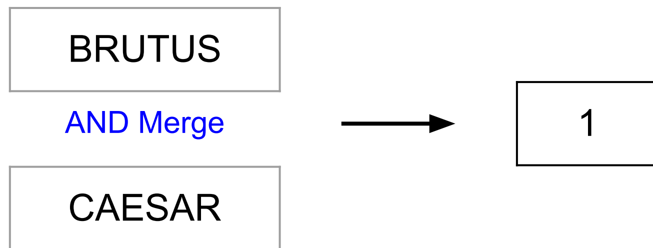
Locate the postings for Brutus and Caesar and Merge the postings (using merge algorithm)

Processing the Query

Query: Brutus and Caesar

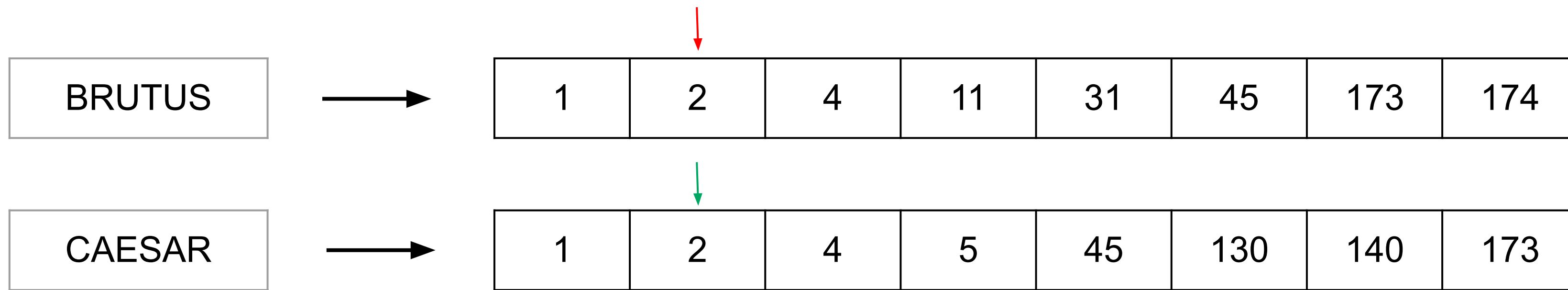


Locate the postings for Brutus and Caesar and Merge the postings

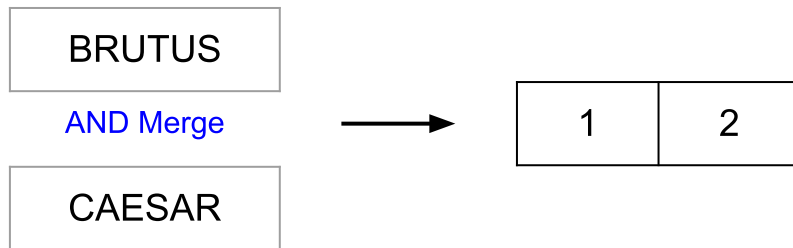


Processing the Query

Query: Brutus and Caesar

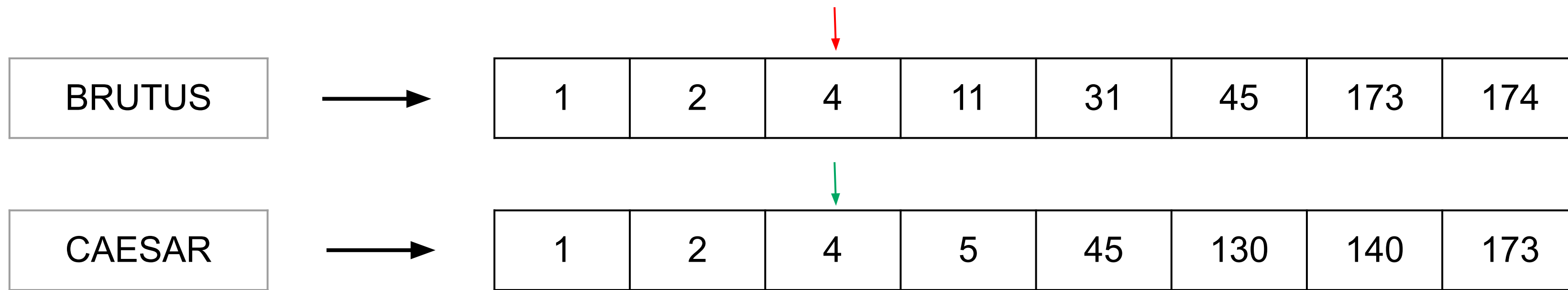


Locate the postings for Brutus and Caesar and Merge the postings

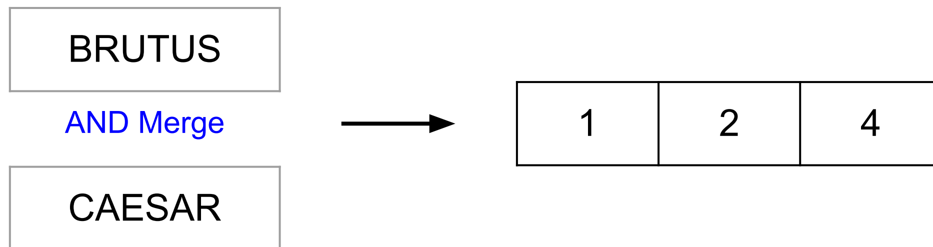


Processing the Query

Query: Brutus and Caesar

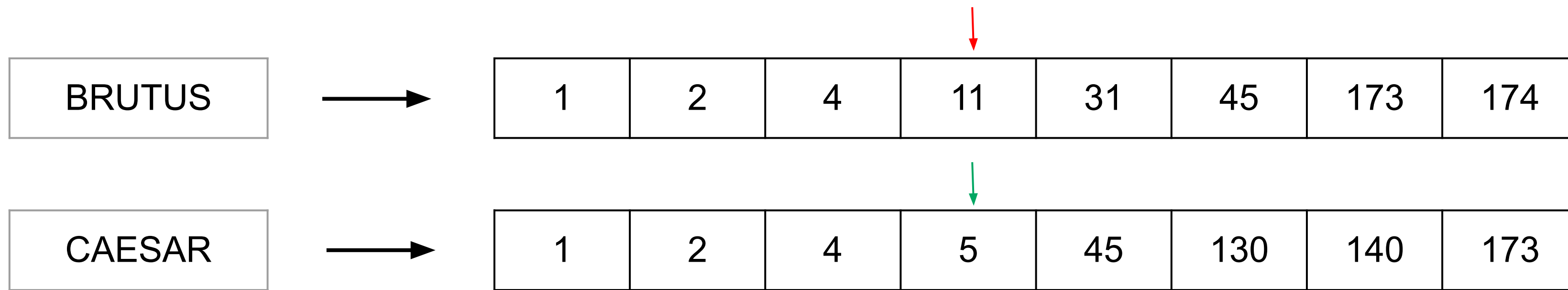


Locate the postings for Brutus and Caesar and Merge the postings

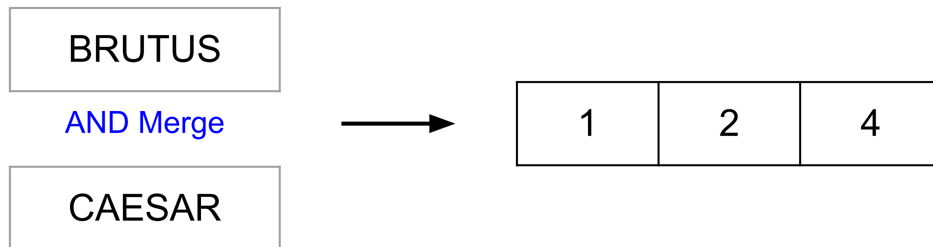


Processing the Query

Query: Brutus and Caesar

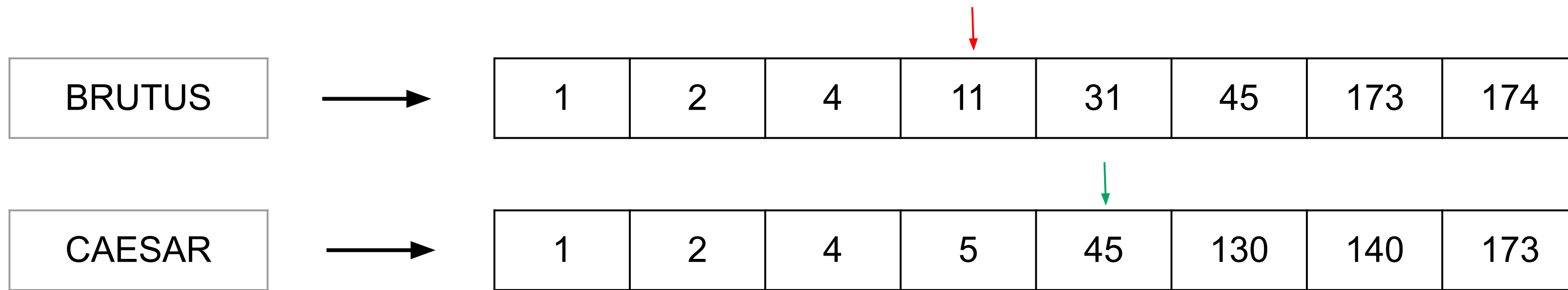


Locate the postings for Brutus and Caesar and Merge the postings

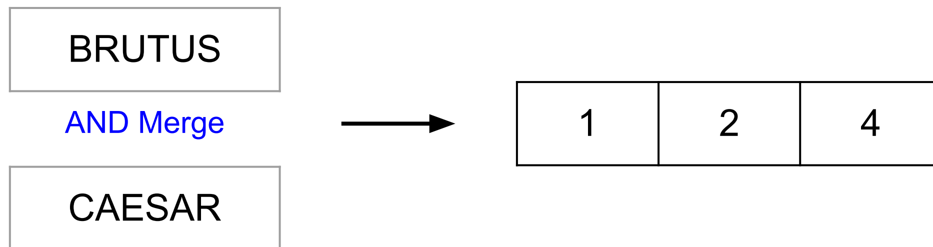


Processing the Query

Query: Brutus and Caesar

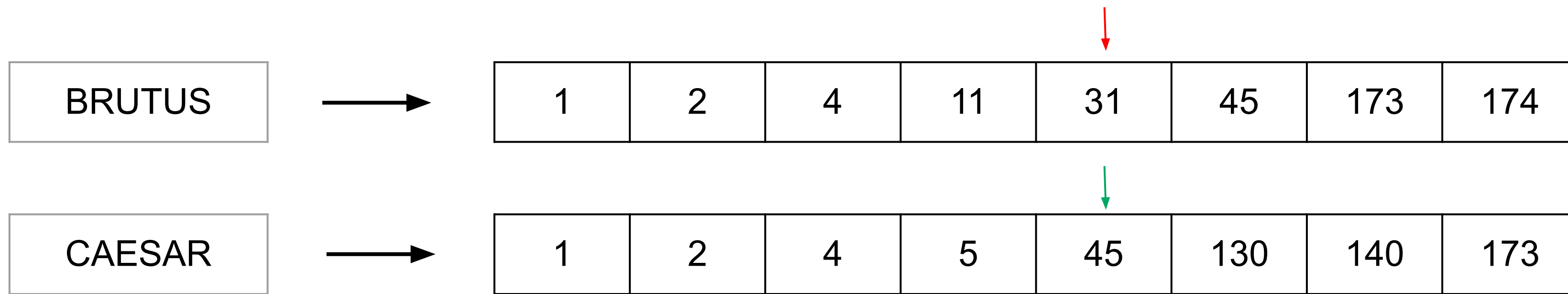


Locate the postings for Brutus and Caesar and Merge the postings

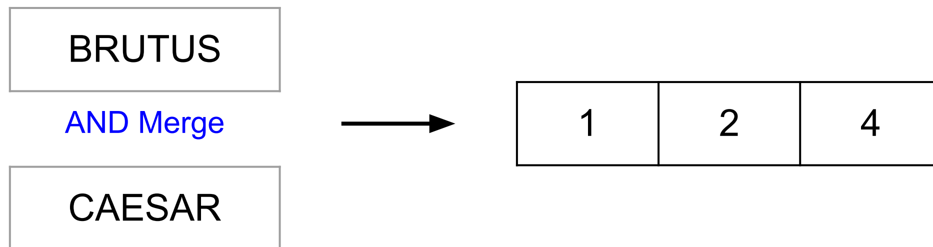


Processing the Query

Query: Brutus and Caesar

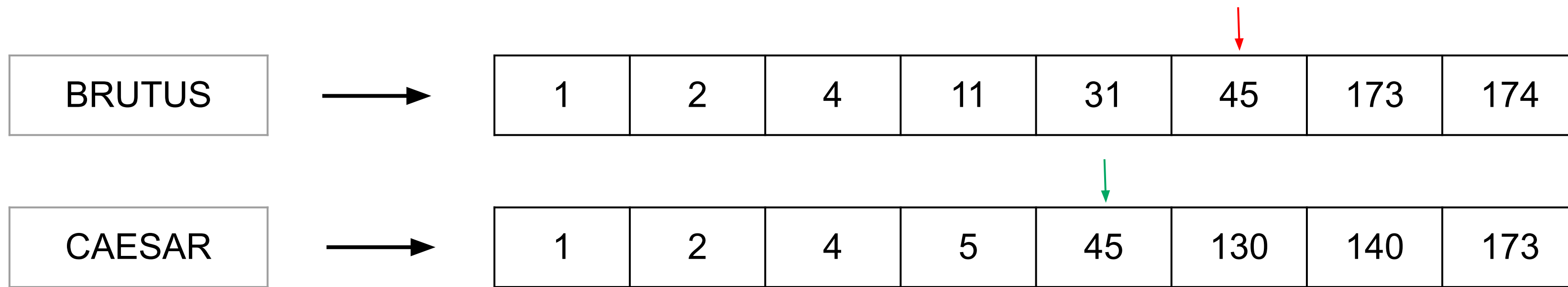


Locate the postings for Brutus and Caesar and Merge the postings

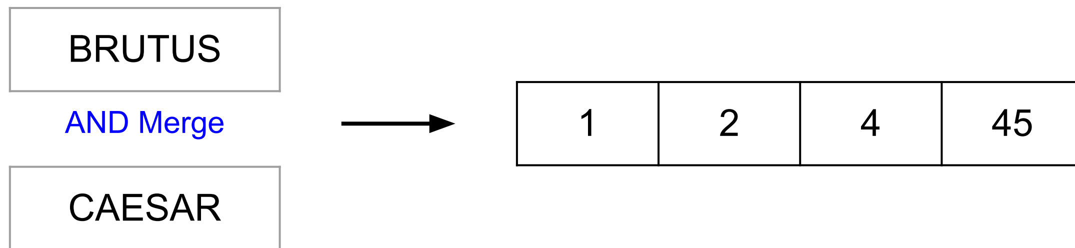


Processing the Query

Query: Brutus and Caesar

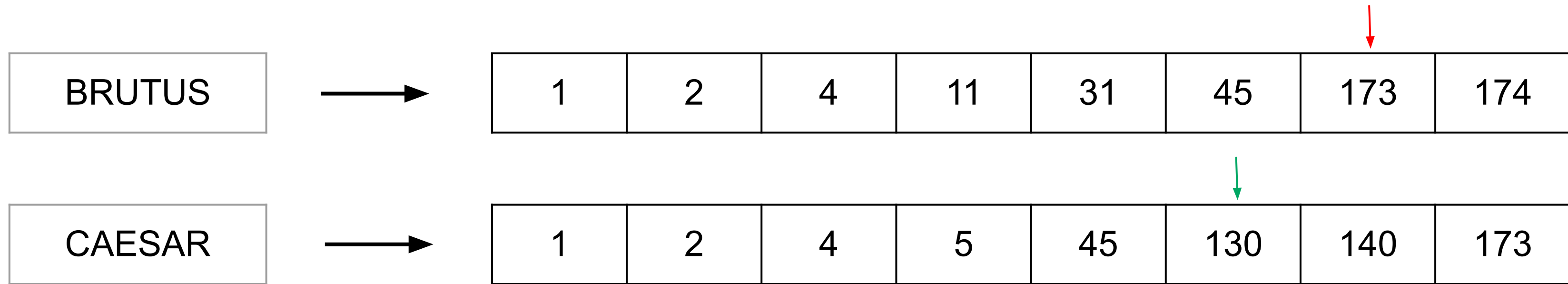


Locate the postings for Brutus and Caesar and Merge the postings

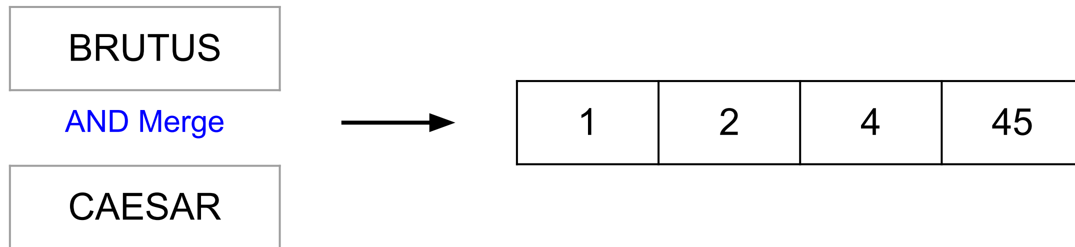


Processing the Query

Query: Brutus and Caesar

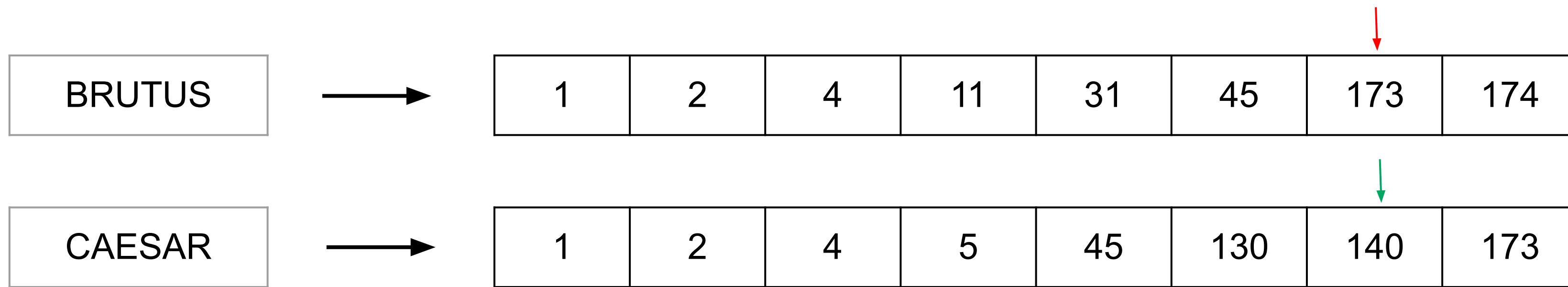


Locate the postings for Brutus and Caesar and Merge the postings

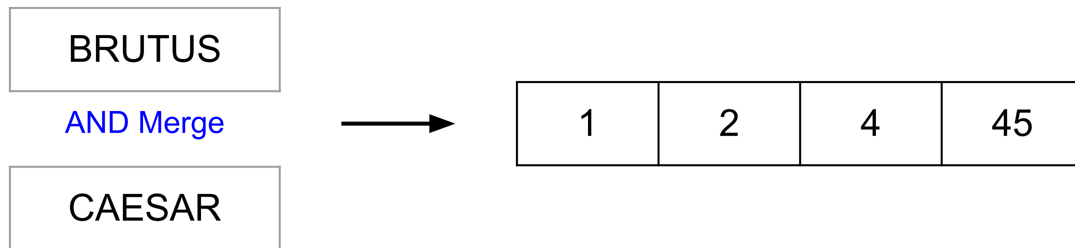


Processing the Query

Query: Brutus and Caesar

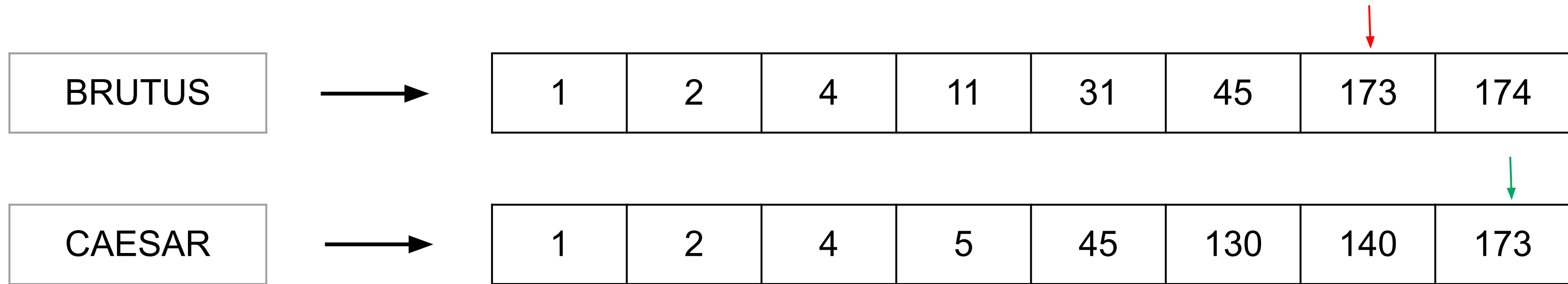


Locate the postings for Brutus and Caesar and Merge the postings

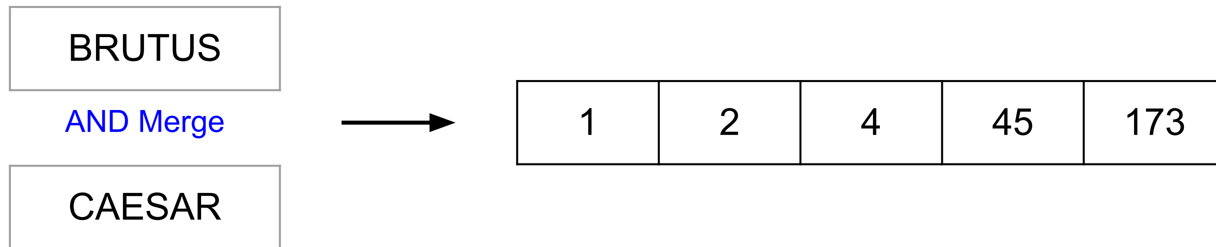


Processing the Query

Query: Brutus and Caesar

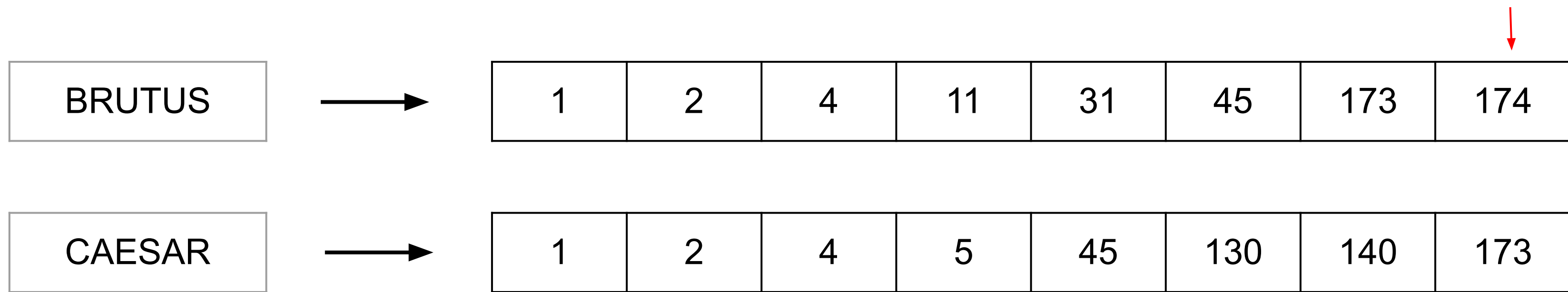


Locate the postings for Brutus and Caesar and Merge the postings

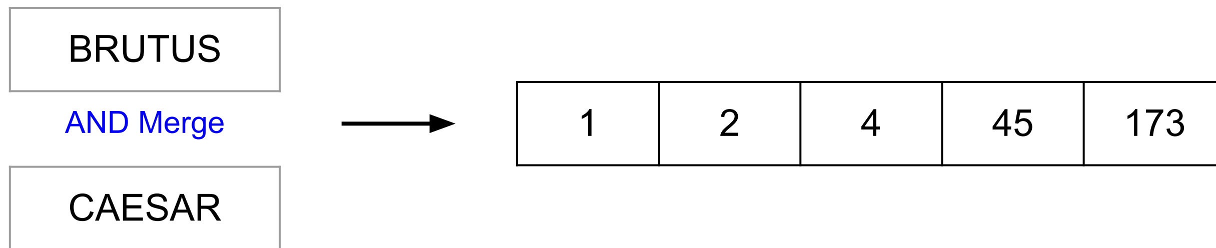


Processing the Query

Query: Brutus and Caesar



Locate the postings for Brutus and Caesar and Merge the postings



Intersecting Two Postings Lists

INTERSECT (P1,P2)

1 answer $\leftarrow$ []

2 **while** P_1 is not empty **and** P_2 is not empty

3 **do if** $\text{docID}(P_1) = \text{docID}(P_2)$

4 **then** ADD(answer, $\text{docID}(P_1)$)

5 $P_1 \leftarrow \text{next}(P_1)$

6 $P_2 \leftarrow \text{next}(P_2)$

7 **else if** $\text{docID}(P_1) < \text{docID}(P_2)$

8 **then** $P_1 \leftarrow \text{next}(P_1)$

9 **else** $P_2 \leftarrow \text{next}(P_2)$

10 **return** answer

Complexity of the Intersection Algorithm

Bounded by worst-case length of postings lists

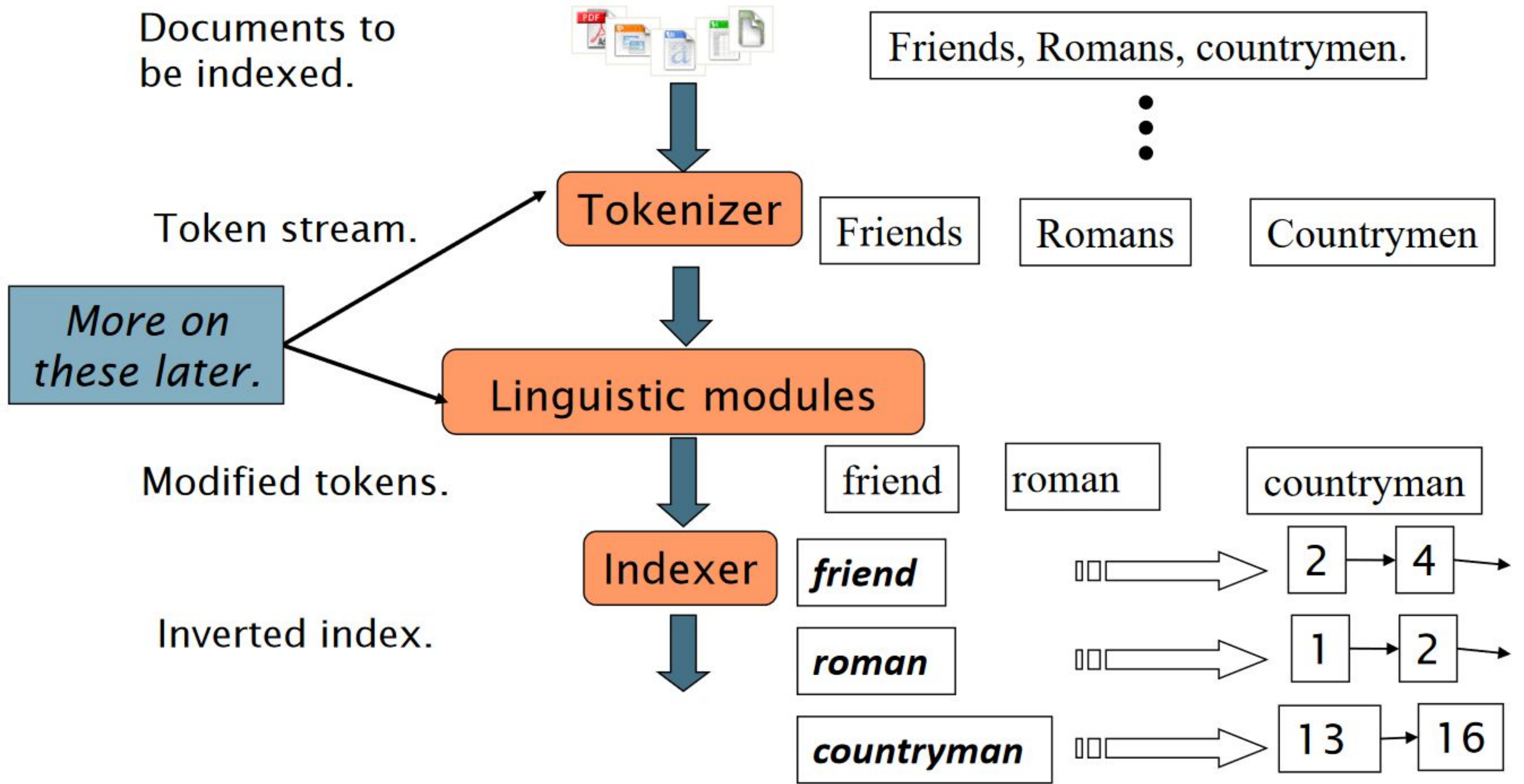
Thus, “officially” O(N), with N the number of documents in the collection

But in practice much better than linear scanning; also O(N)

How to Build Inverted Index

1. Collect the documents to be indexed
2. Tokenize the text, turning each document into a list of tokens
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings

How to Build Inverted Index



Now You Do It

Coffee →	5	11	20	25	30	40	52	58
Colombia →	4	20	30	37	40			
Espresso →	1	5	70	79	83	94		

Compute hit list for (Coffee and not Colombia) or Espresso

Query Optimization

Consider a query that is an and of n terms, n > 2

- For each of the terms, get its postings list, then merge them together
- Example query: Brutus AND Calpurnia AND Caesar
- What is the best order for processing this query?

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 Calpurnia, then Caesar, then Brutus

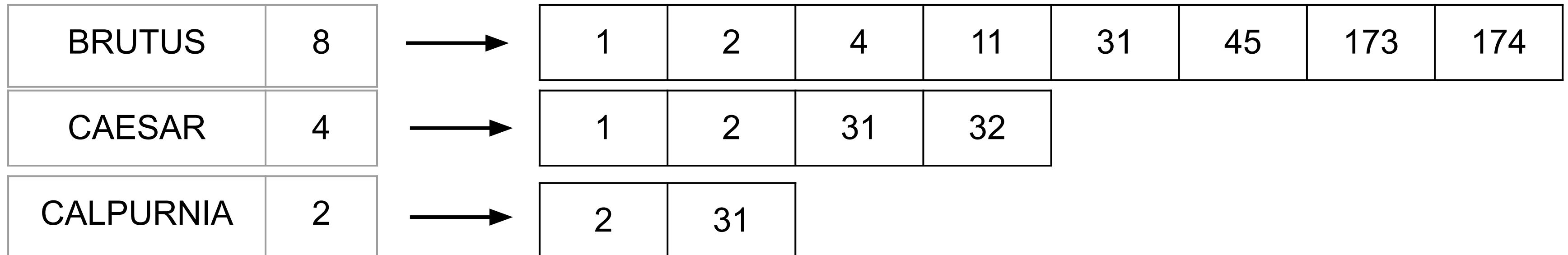
Query Optimization

Consider a query that is an and of n terms, n > 2

- For each of the terms, get its postings list, then merge them together
- Example query: Brutus AND Calpurnia AND Caesar
- What is the best order for processing this query?

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 Calpurnia, then Caesar, then Brutus



Optimized Intersection 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 is not empty and result is not empty
5    do result ← Intersect(result, postings(first(terms)))
6        terms ← rest(terms)
7    return result
```

Start by intersecting the two smallest postings lists
then all intermediate results must be no bigger than the smallest postings list,
and we are therefore likely to do the least amount of total work

Boolean Retrieval in Real-World

WestLaw

Westlaw is the largest commercial (subscribers pay) legal search service, created in 1975 (http://www.westlaw.com)

West (also known by its original name, **West Publishing**) is a business owned by Thomson Reuters

Include more than 40,000 databases of case law, state and federal statutes, administrative codes, newspaper and magazine articles, …

Evolved over the years:

- Ranking in 1992
- A new federated search model in 2010
- Still large percentage of users prefer Boolean queries

Example

Information need: What is the statute of limitations in cases involving the federal tort claims act?

Query: LIMIT! /3 **STATUTE ACTION** /S FEDERAL /2 TORT /3 CLAIM

Format: : /3 = within 3 words, /S = in same sentence !: wild card (Limited, Limitation)

PubMed

PubMed Advanced Search Builder

PubMed.gov

User Guide

Add terms to the query box

All Fields

Enter a search term

AND

Show Index

Query box

Headache & pain

Search

PubMed provides free access to MEDLINE (1966-current) and PREMEDLINE (https://pubmed.ncbi.nlm.nih.gov)

PubMed provides for Boolean searching through advanced search

Boolean operators must be CAPITALIZED in PubMed: **AND, OR, NOT**

Symbols for Boolean Operators**: &, OR,**

Does Google use the Boolean Retrieval?

On Google, the default interpretation of a query [w_1 w_2 ... w_n] is w_1 AND w_2 AND ... AND w_n

Cases where you get hits which do not contain one of the w_i :

- Page contains variant of w_i (morphology, misspelling,synonym)
- Long query (n is large)
- Boolean expression generates very few hits
- w_i was in the anchor text

Google also ranks the result set

- Simple Boolean Retrieval returns matching documents in no particular order
- Rank hits according to some estimator of relevance

WHAT HAVE YOU LEARNED?

Summary

Today we learned about:

- ✓ Boolean Retrieval Model
- ✓ Term-document Incidence Matrix
- ✓ Inverted Index

Next Session

Tokenization and Stemming

Applying natural language processing techniques to text before retrieval (Pre-processing)

We will explore

- Tokenization
- Stemming
- Lemmatization

To do:

- Assignment 1 is ready! Deadline September 15!
- Let’s review the assignment

For Question 2, you are allowed to use python set!
With set, operations such as and/or/not are easy to be deployed!