Information Retrieval

Lecture 1: Boolean retrieval

Today

- Creating a system for Boolean retrieval
  - Creating an index (offline)
  - Evaluating a query (online)
- Performance criteria
  - Efficiency: Processing speed
    - Complexity, required resources
  - Effectiveness
    - System search specificity, UX
- Reading Materials: Introduction to IR, Chapter 1

Information Retrieval

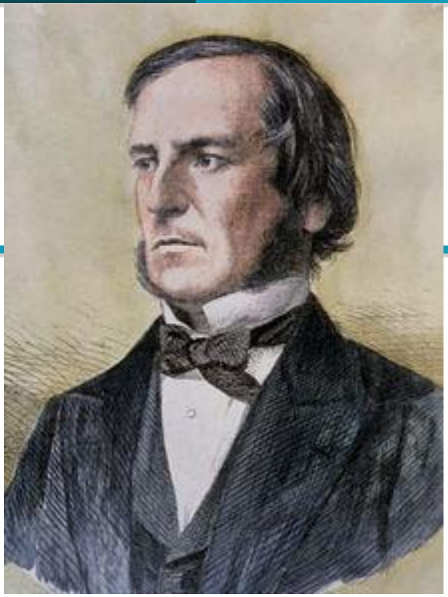
- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) from within large collections (usually stored on computers), given a specific information need .
  - These days we frequently think first of web search, but there are many other cases:
    - E-mail search
    - Searching your laptop
    - Corporate knowledge bases
    - Legal information retrieval

IR In a nutshell

RETRIEVAL FROM A DIGITAL INDEX USING BOOLEAN CONNECTIVES

Refreshing propositional logic

George Boole (1815-1864)



- English Mathematician
- 1847: Mathematical analysis of Logic
- Developed an algebra for Aristotle's logic
  - Propositional variables *x,y* valued true (1) or false (0)
  - Operators and arithmetic equivalent
    - *x* AND *y* ➔ *xy*
    - *x* OR *y* ➔ *x + y - xy*
    - NOT *x* ➔ *1-x* (unary operator)

Logical Conjunction

p	*q*	*p* ∧ *q*
T	T	T
T	F	F
F	T	F
F	F	F

Logical connectives in natural language

- Natural language expressions are full of logical connectives:
  - AND: and, and then, however, but
  - OR: or
  - NOT: not, nobody, none
  - XOR: either … or …
  - $\leftrightarrow \Leftrightarrow \equiv \Longleftrightarrow$: if and only if (*material equivalence)*
  - ➔ if .. Then (*material implication*)
- But natural language is also ambiguous, vague, implicit, sensitive for framing etc.

Unstructured data in 1680 *(running example from IIR book)*

Query: Brutus and Caesar and not Calpurnia

- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
  - grep Brutus Shakespeare | grep Caesar | grep –v Calpurnia
- Why is that not the answer?
  - Slow (for large corpora)
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)
    - Later lectures
- Solution: inverted data structures

Term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

1 if play is about term,
0 otherwise

Brutus** AND **Caesar but ***NOT Calpurnia***

conflicting positions

documents without conflict

Incidence vectors

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

1 if play is about term,
0 otherwise

- So we have a 0/1 vector for each term:
  - 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

Brutus** AND **Caesar** but NOT **Calpurnia

Brutus** AND **Caesar** AND NOT **Calpurnia

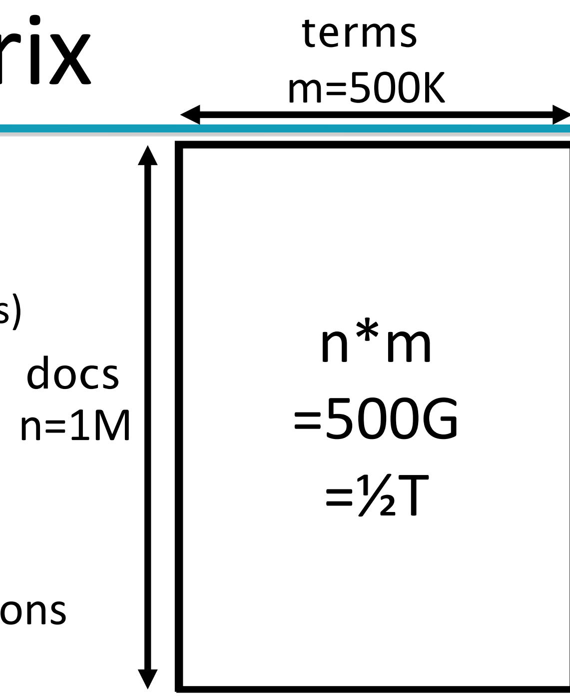
- To answer query: take the vectors for
  - Brutus 1 1 0 1 0 0
  - Caesar 1 1 0 1 1 1
  - Calpurnia **not** 010000 = 1 0 1 1 1 1 (complemented!)

(AND)

→ bitwise AND: 1 0 0 1 0 0

INDEX CONSTRUCTION

Building the incidence matrix



#ones: n*50 = 50M

- Suppose we have:
  - n = 1,000,000 = 1M documents,
  - average document size d = 1000 = 1K terms (incl duplicates)
  - dictionary size m = 500,000 = 500K unique terms
  - u = 50 unique terms per document
- Then the incidence matrix has size:
  - n * m = 1M * 500K = 500G = 500,000,000,000 positions (half a trillion)
- But it has no more than
  n * u = 1M * 50 = 50M = 50,000,000 ‘ones’!
  - So: matrix is extremely sparse.
    (density = n*u / n*m = u / m = 50/500K = 1/10,000 = 0.1 ‰)
- What is a better representation?
- We only record the non-zero positions.
- How can we implement the boolean operators for that representation?

Inverted index / inverted file

- For each term *t*, we must store a list of all documents that contain *t* (= the list of all documents about *t*)
  - Identify each by a **docID**, a document serial number
- Can we use fixed-size arrays for this?

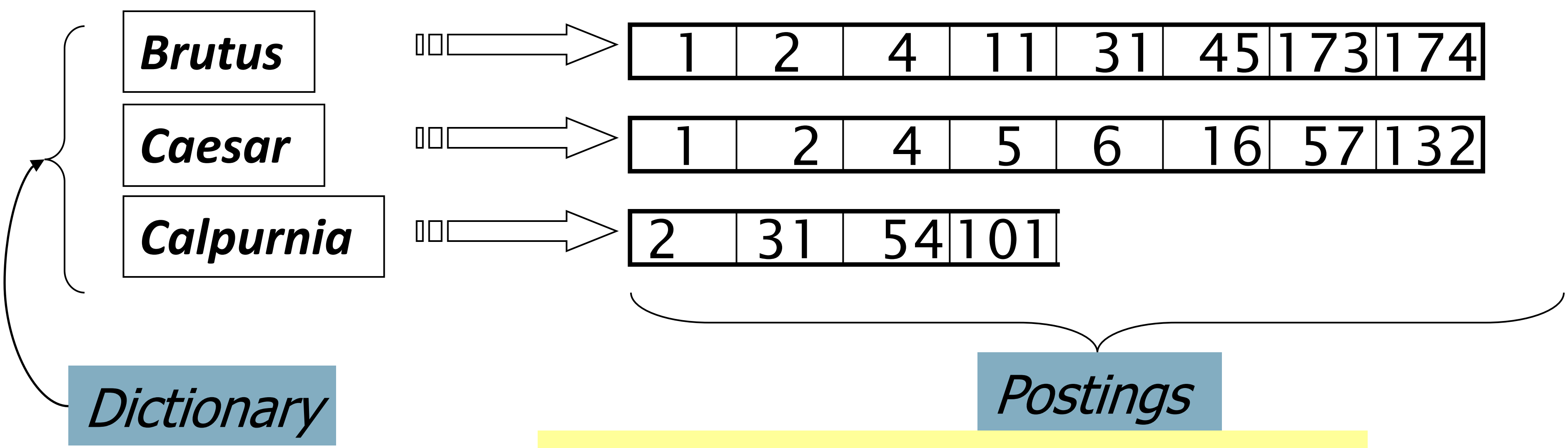
These are called 'postings'

Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

What happens if the word ***Caesar*** is added to document 14 (e.g. recrawled Wikipedia page)?

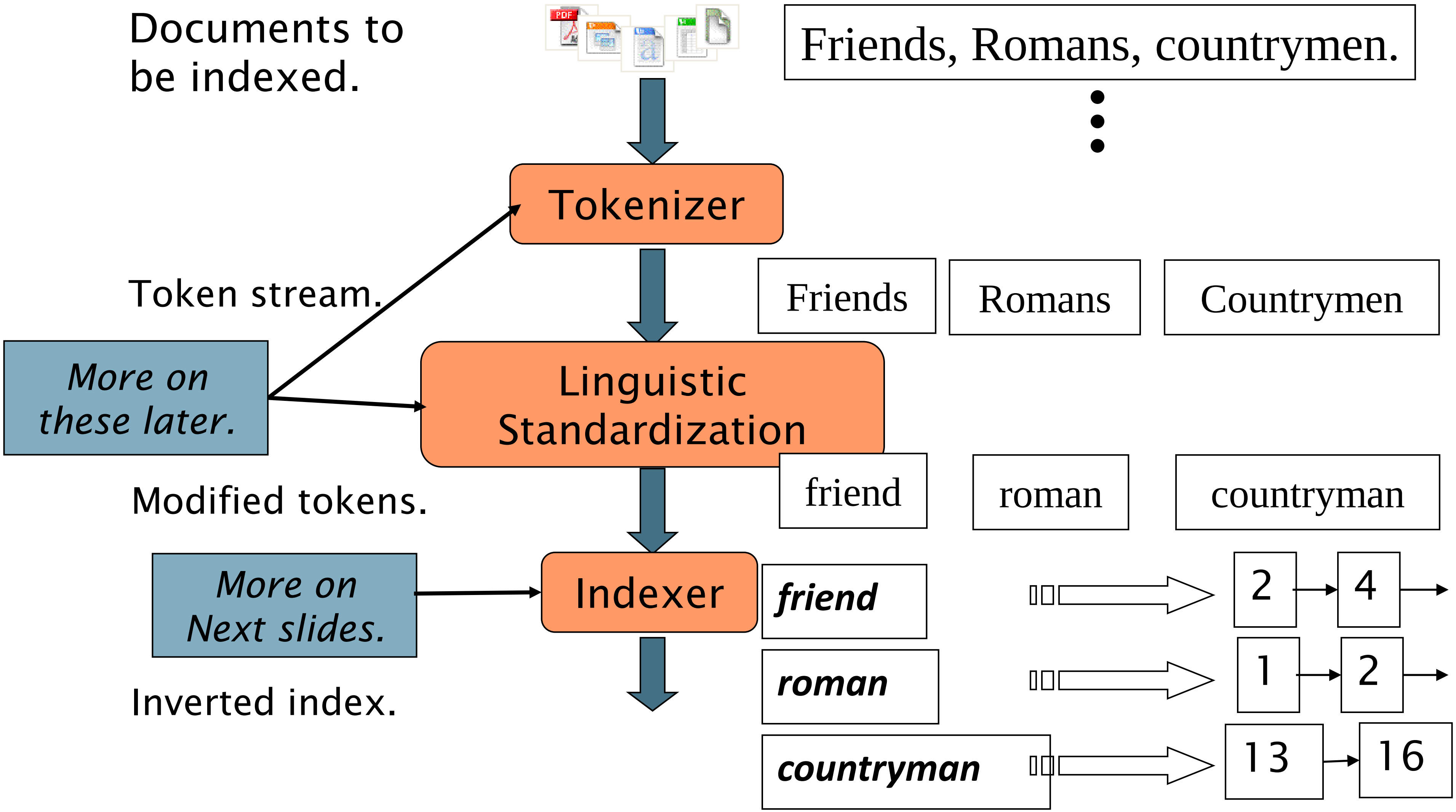
Inverted index

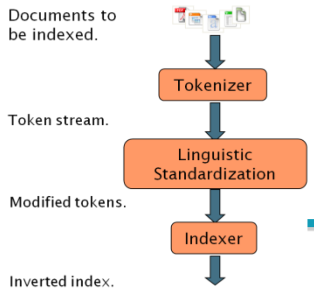
- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion



Sorted by docID (more later on why).

Inverted index construction





Indexer steps: Token sequence

- First the modified token streams are converted to a single sequence of

  (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort the sequence of (Term, DocID) pairs
  - by term
  - and then docID

Core indexing step

- Terms will occur more than once in the list

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency? Will discuss later.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Where do we pay in storage?

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Terms and counts

Lists of docIDs

Pointers

Later in the course:
- How do we index efficiently?
- How much storage do we need?

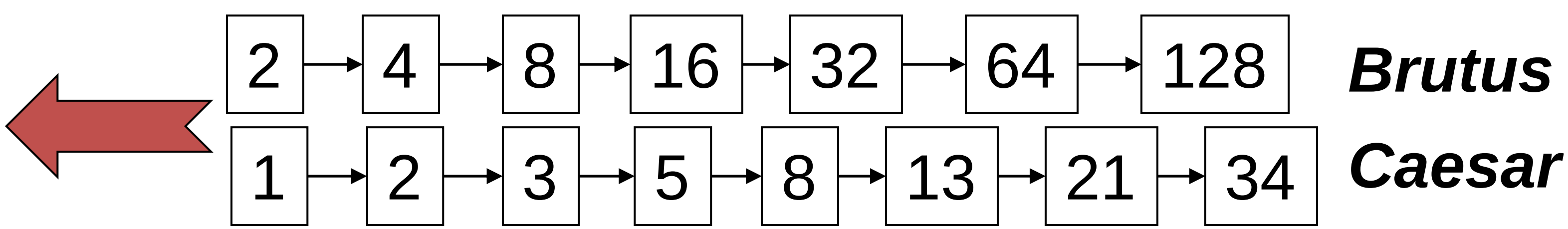
QUERY EVALUATION

Query processing: AND

- Consider processing the query:

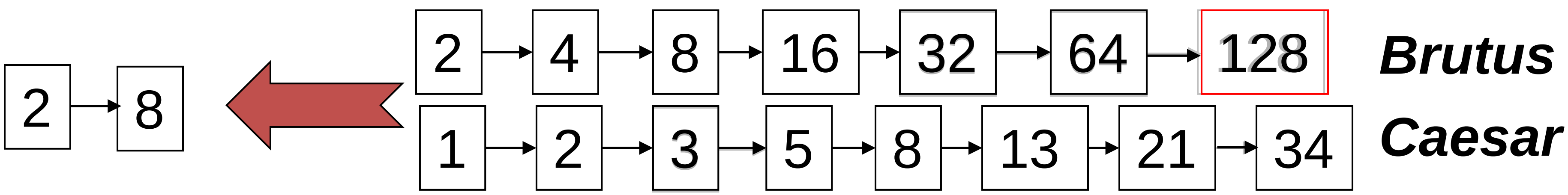
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - “Merge” the two posting lists:



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *n* and *m*, the merge takes n+m-1 (maximum) posting comparisons, and thus O(*n*+*m*) time complexity.
What is the minimum?

Crucial: postings sorted by docID.
Otherwise O (n*m) operations! (why?)

Intersecting two postings lists (a “merge” algorithm)

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
             then p1 ← next(p1)
             else p2 ← next(p2)
return answer
```

More general merges

- Group Exercise: Adapt the merge for the OR query:
  - Brutus OR Caesar

- Can we still run the merge in $O(n+m)$?

Junction of two postings lists (a "merge" algorithm)

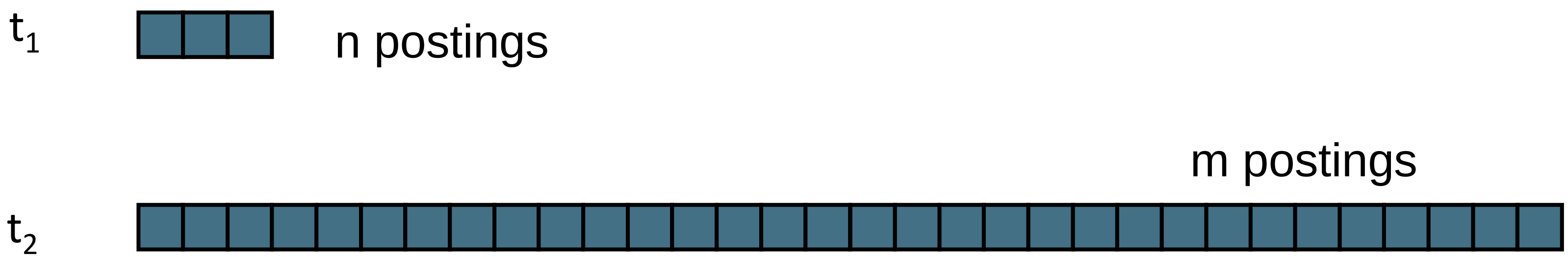
```
UNION(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

Add(answer,docID(p1);

Add(answer,docID(p2);

Merging

- Can we always merge in “linear” time?
- Can we do better?



- Use binary search!
- Cost: O (n * ^{2}log (m))
- Example: n = 3, m = 1024 = 2^{10}
  Then n+m = 1027, and n ^{2}log (m) = 3*10 = 30
- But n = 1024 , m = 1024?

QUERY PROCESSING OPTIMIZATION

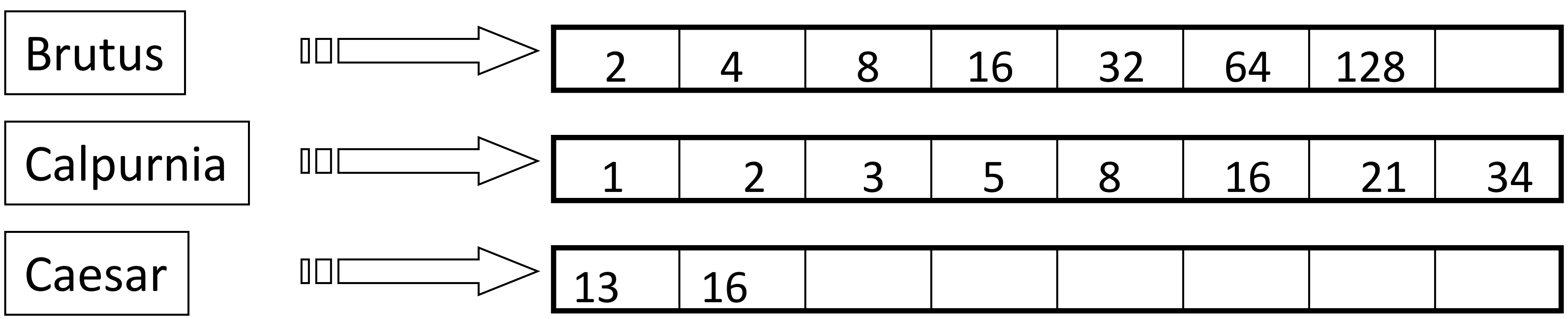
Refresher: Rewriting Boolean expressions

- Associativity: order of evaluation not relevant
  - $(x \wedge y) \wedge z = x \wedge (y \wedge z)$
  - $(x \vee y) \vee z = x \vee (y \vee z)$
- Commutativity: operands may be swapped
  - $x \wedge y = y \wedge x$ $\quad$ $x \vee y = y \vee x$
- Distributivity of $\wedge$ over $\vee$, $\vee$ over $\wedge$, $\neg$ over $\wedge$ and $\neg$ over $\vee$
  - $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$,
  - $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
  - $\neg (a \wedge b) = \neg a \vee \neg b$
  - $\neg (a \vee b) = \neg a \wedge \neg b$

de Morgan's laws

Query processing optimization

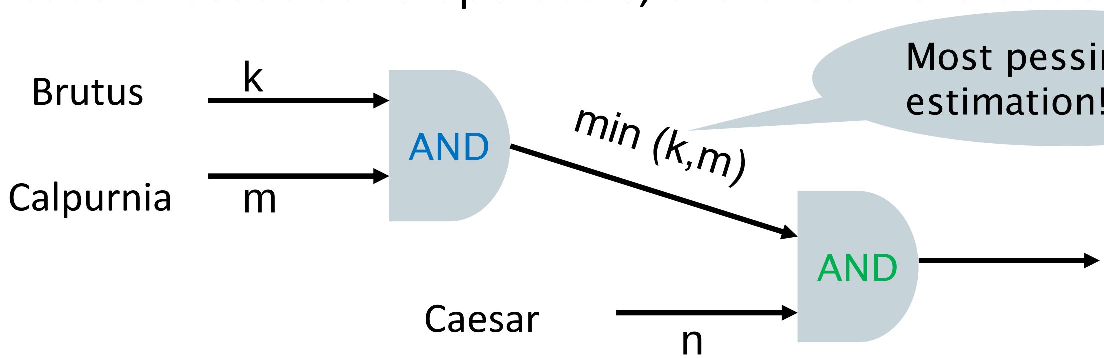
- What is the best order for query processing?
- Consider a query that is an AND of k terms.
- For each of the k terms, get its postings, then AND together.



Query: **Brutus** AND **Calpurnia** AND **Caesar**

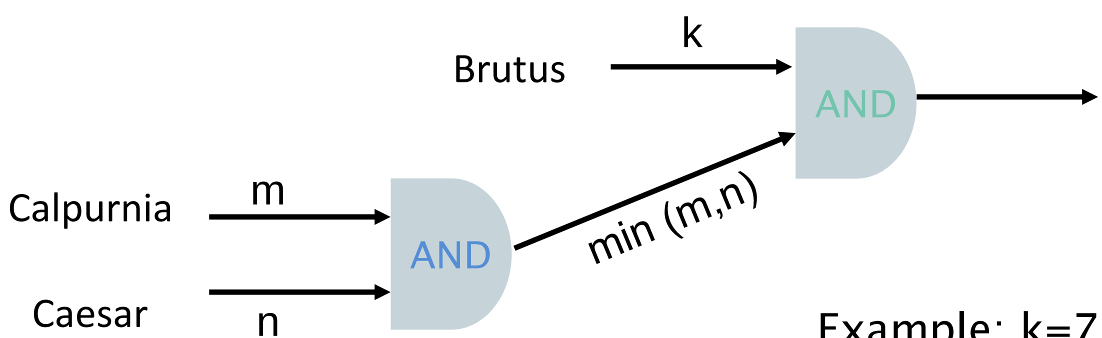
Associative operators

- In case of associative operators, there is an evaluation choice:



Cost1:
[k + m] + [min(k,m) + n]
= k + m + n + min(k,m)

(Brutus∧Calpurnia)∧Caesar



Cost2:
[m + n] + [k + min(m,n)]
= k + m + n + min(m,n)

Example: k=7, m=8, n=2
Cost1 = 17+7 = 24
Cost2 = 17+2 = 19

Brutus∧(Calpurnia∧Caesar)

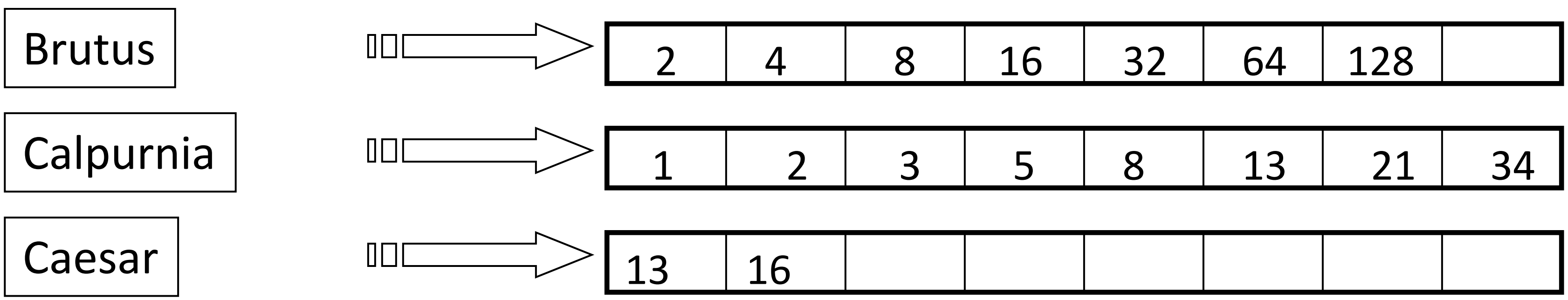
Commutative operators

- For a commutative operator we may even exchange the operands:

  - Brutus AND Calpurnia AND Caesar
    = (Brutus AND Calpurnia) AND Caesar
    = Brutus AND (Calpurnia AND Caesar)
    = *(Brutus AND Caesar) AND Calpurnia* (new)

Query optimization example

- Process in order of increasing freq:
  - start with smallest set, then keep cutting further.



Execute the query as (**Caesar** AND **Brutus)** AND **Calpurnia**.

Query transformation

- Consider the query:
  - (eyes or skies) and tangerine

OR is inefficient

- The estimated cost:
  - cost or: 213,312 + 271,658 = 484,970
  - cost and: 484,970 + 46,653 = 531,623
  - total cost: 1,016,593
- Transform into (distributivity):
  - (eyes and tangerine) or (skies and tangerine)
- Estimated cost:
  - cost and: 213,312 + 46,653 = 259,965
  - cost and: 271,658 + 46,653 = 318,311
  - cost or: 46,653 + 46,653 = 93,306
  - total cost: 671,582

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

So a more complex query may require less computational effort!

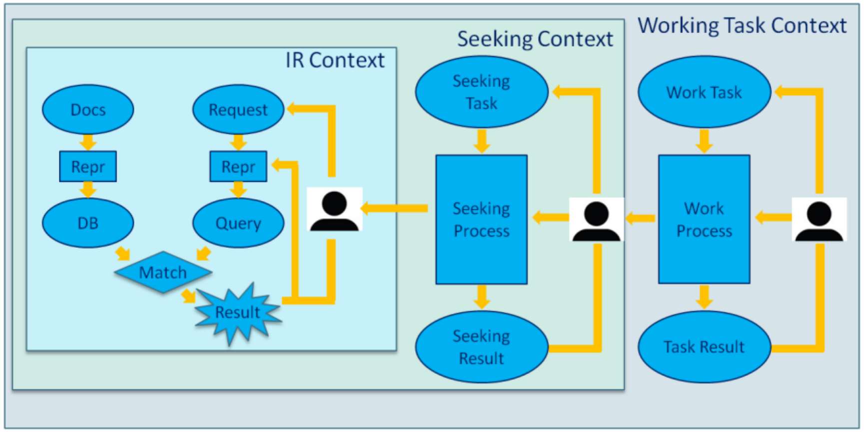
More general optimization

- Strategy:
  - Get freq's for all terms.
  - Estimate the size of each OR by the sum of its freq's (conservative).
  - Estimate the size of each AND by the minimum of its freq's.
  - Delay evaluation of OR if possible (transformation)
  - Process in increasing order of OR sizes (or AND sizes).

BOOLEAN SYSTEMS APPRECIATION & USE CASE

Basic assumptions of Information Retrieval

- Collection: Fixed set of documents
- Goal: Retrieve documents
  - with information that is relevant to the user's information need
  - and helps the user complete a task



How good are the retrieved docs?

- *Effectiveness of a system?*
- *Precision* : Fraction of retrieved docs that are relevant to user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved
- These are ***set based*** metrics (intuitive)
- More precise definitions and measurements to follow in later lectures

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades (60s-80s).
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

Example: WestLaw http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Tens of terabytes of data; 700,000 users
- Majority of users *still* use Boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
    - /3 = within 3 words
    - /S = in same sentence
    - ! = wildcard

Example pubmed

- https://www.ncbi.nlm.nih.gov/pubmed/advanced
- Tutorial: http://www.youtube.com/watch?v=dncRQ1cobdc&feature=relmfu
- Exercise: build query for Information Need:
  - breast cancer treatment options other than chemotherapy
  - How to add related terms?

Disadvantages of Boolean Retrieval

- Retrieval based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression which most users find awkward
  - AND/OR in natural language are different from logical AND/OR
  - NOT operator is hard for humans
  - The Boolean queries formulated by the users are most often too simplistic
    - So: either too few or too many documents in response to a user query

Further reading & assignment

- IIR book: Chapter 1
- For library science and historic perspective
  - Introduction chapter to Readings in Information Retrieval (Sparck Jones and Willett, 1997)
    - https://books.google.nl/books?id=Nt5nDTYQ0okC&pg=PA1&source=gbs_toc_r&cad=4#v=onepage&q&f=false
- A (dated) view on Google's engineering efforts
  - Video Jeff Dean: Chief engineer Google http://videolectures.net/wsdm09_dean_cblirs/
    - Note this requires a Flash player.
- Assignment week1!
  - on Brightspace, submit answers on Brightspace