Ricerca dell’Informazione nel Web

Aris Anagnostopoulos

Docenti

- **Dr. Aris Anagnostopoulos**

  http://aris.me

  Stanza B118

  Ricevimento: Inviate email a: **aris@cs.brown.edu**

- Laboratorio:

  Dr.ssa Ilaria Bordino (Yahoo! Barcelona)

  Ing. Ida Mele (DIS)

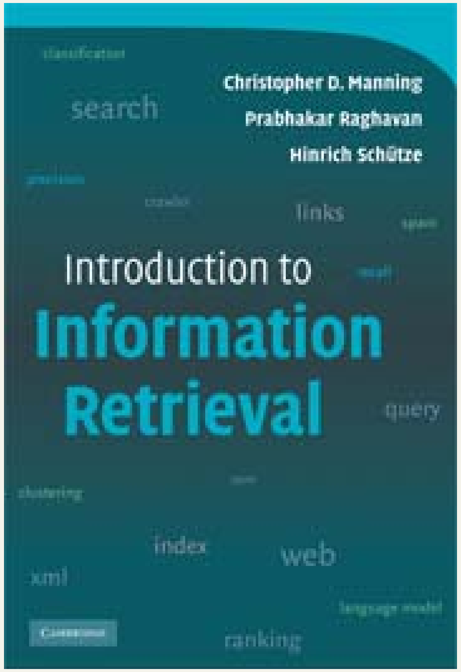
Program

1. Information Retrieval: Indexing and Querying of document databases
2. Vector space model
3. Search Engines: Architecture, Crawling, Ranking e Compression
4. Classification and Clustering
5. Projects (lab)

Materiale didattico

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schueze, **Introduction to Information Retrieval**, Cambridge University Press, 2007.

http://nlp.stanford.edu/IR-book/



Exam

- L'**esame** prevede lo svolgimento di una **prova scritta** sui temi affrontati nel corso e di un **progetto** a scelta del candidato.
  Il progetto deve essere consegnato in occasione della prova scritta ad eccezione che per gli studenti che sostengono il primo appello del corso per cui la consegna e' possibile anche in occasione del secondo appello.

Web page

- **http://aris.me**

and follow the link about teaching

- Slides and other class material

- **Announcements:**

  We will be posting announcements about changes etc. at the web page. Please check it often!

Web Information Retrieval

Lecture 1

Introduction

Query

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- Could grep all of Shakespeare’s plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the phrase ***Romans and countrymen***) not feasible

Term-document incidence

	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 contains word, 0 otherwise

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100.

Answers to query

- ## Antony and Cleopatra, Act III, Scene ii
- *Agrippa* [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
- When Antony found Julius ***Caesar*** dead,
- He cried almost to roaring; and he wept
- When at Philippi he found ***Brutus*** slain.

- ## Hamlet, Act III, Scene ii
- *Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the
- Capitol; ***Brutus*** killed me.

Bigger corpora

- Consider *n* = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
    - 6GB of data in the documents.
- Say there are *m* = 500K *distinct* terms among these.

Can’t build the matrix

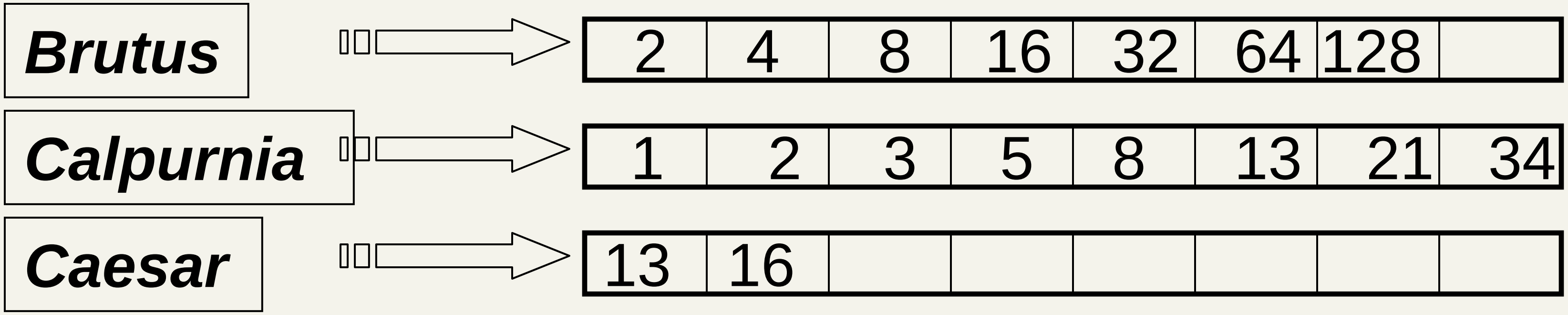
- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s.
  - matrix is extremely sparse.

Why?

- What’s a better representation?
  - We only record the 1 positions.

Inverted index

- For each term *T*, must store a list of all documents that contain *T*.
- Do we use an array or a list for this?

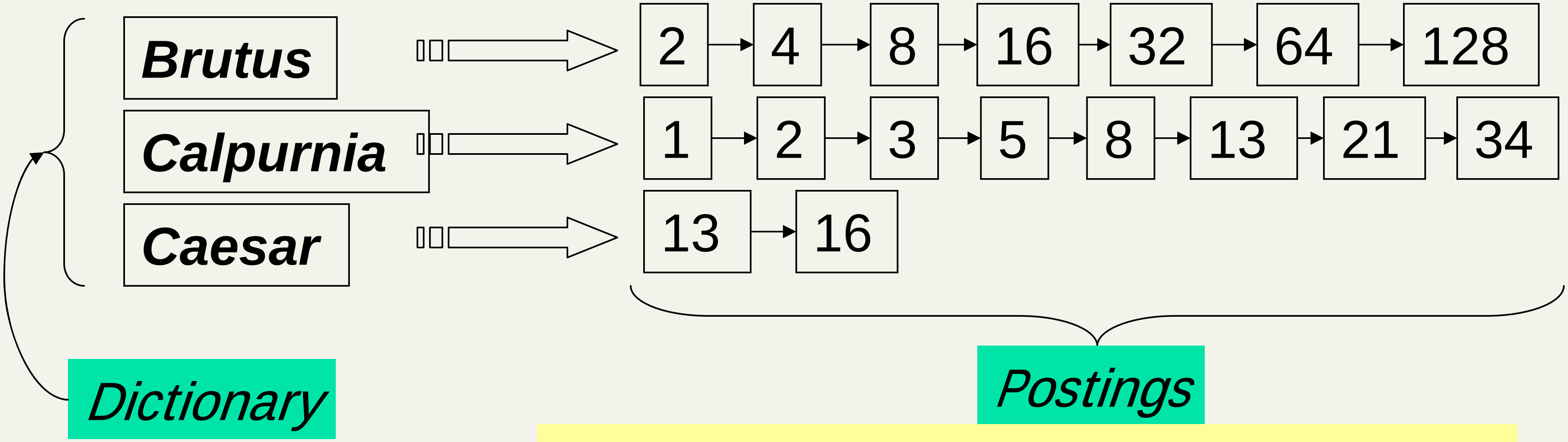


Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

What happens if the word ***Caesar*** is added to document 14?

Inverted index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers



Brutus → 2 → 4 → 8 → 16 → 32 → 64 → 128

Calpurnia → 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

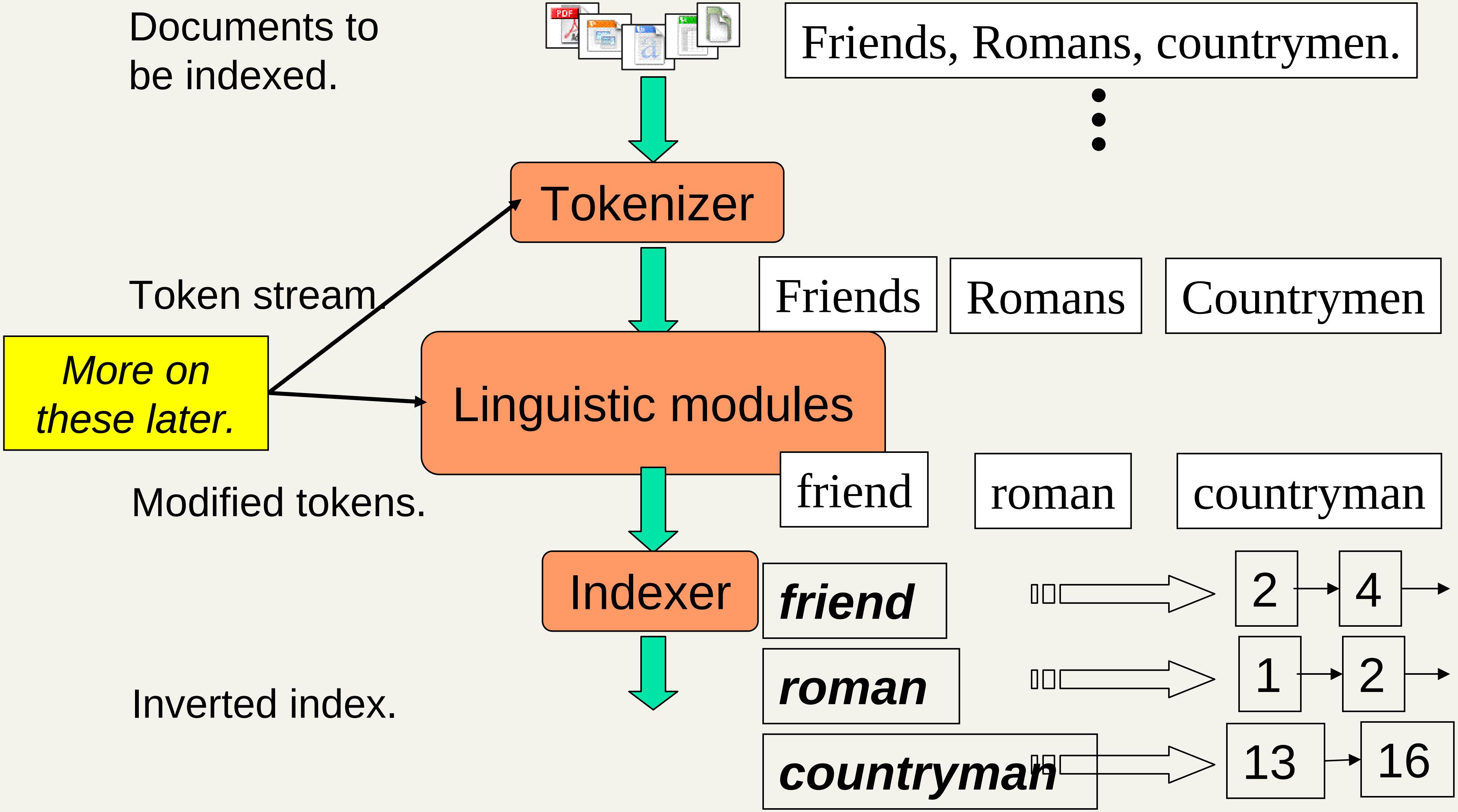
Caesar → 13 → 16

Dictionary

Postings

Sorted by docID (more later on why).

Inverted index construction



Documents to be indexed.

Friends, Romans, countrymen.

Tokenizer

Token stream.

Friends Romans Countrymen

More on these later.

Linguistic modules

Modified tokens.

friend roman countryman

Indexer

Inverted index.

friend → 2 → 4

roman → 1 → 2

countryman → 13 → 16

Indexer steps

- 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	Doc #
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

- Sort by terms.

Core indexing step.

Term	Doc #
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	Doc #
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

- Multiple term entries in a single document are merged.
- Frequency information is added.

Why frequency?
Will discuss later.

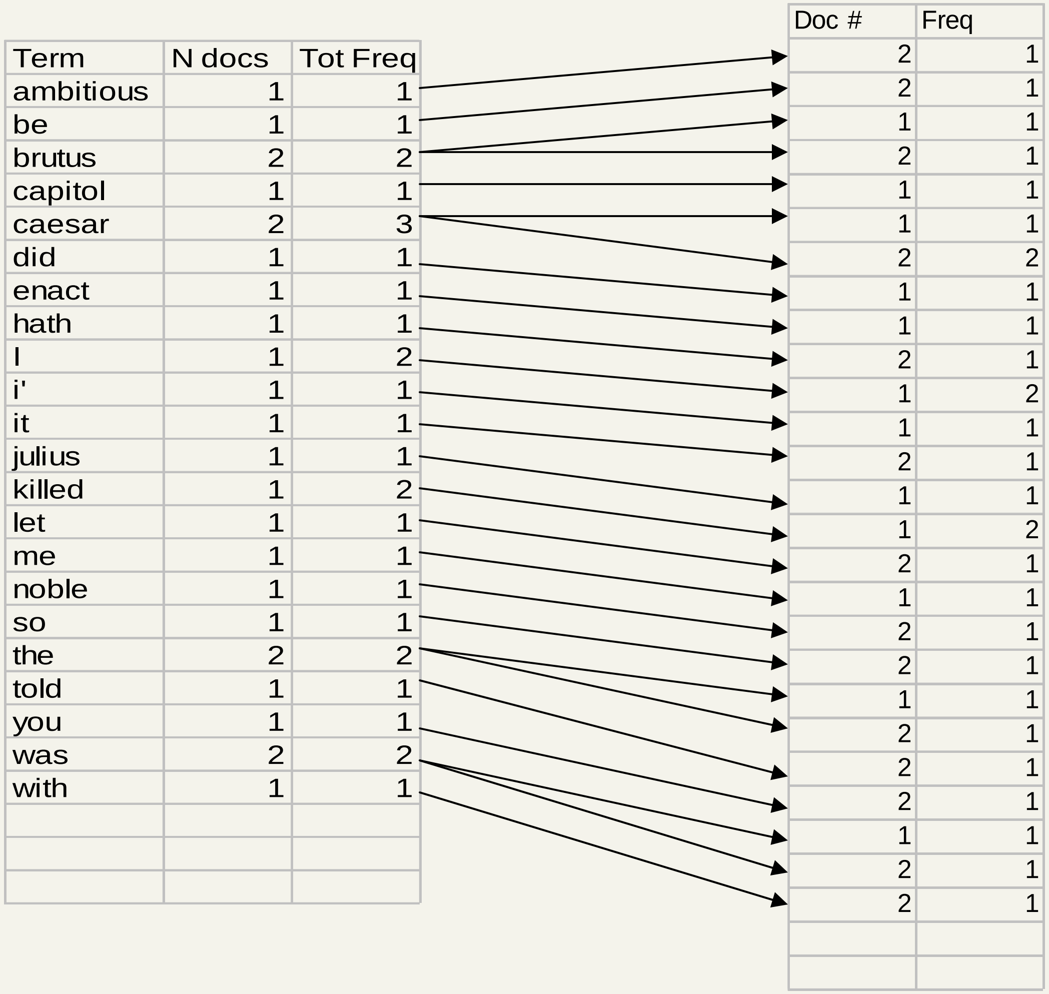
Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

→

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

- The result is split into a *Dictionary* file and a *Postings* file.

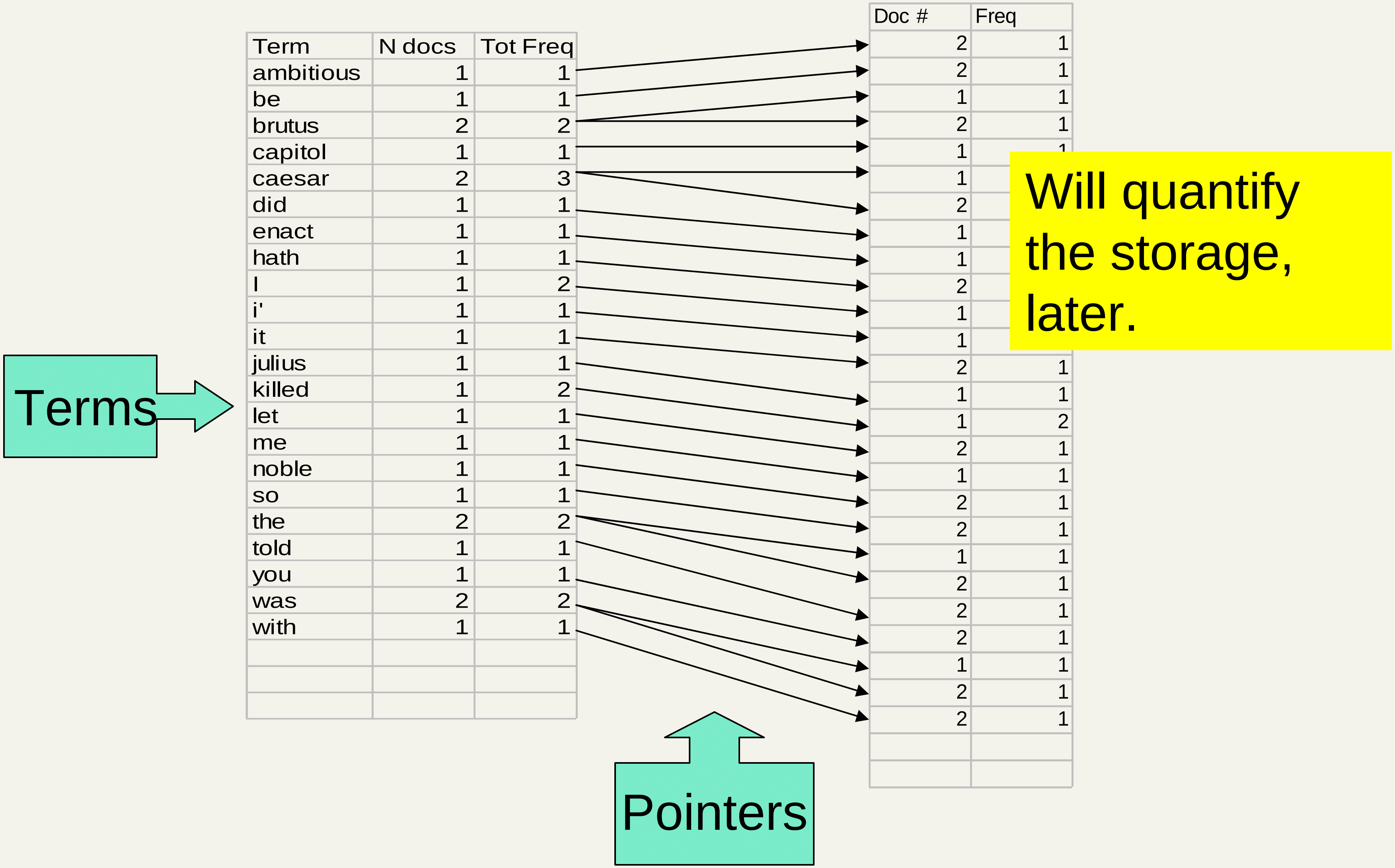
Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1



Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

- Where do we pay in storage?



Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	
2	
1	
1	
2	
1	
1	
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Terms

Pointers

Will quantify the storage, later.

The index we just built

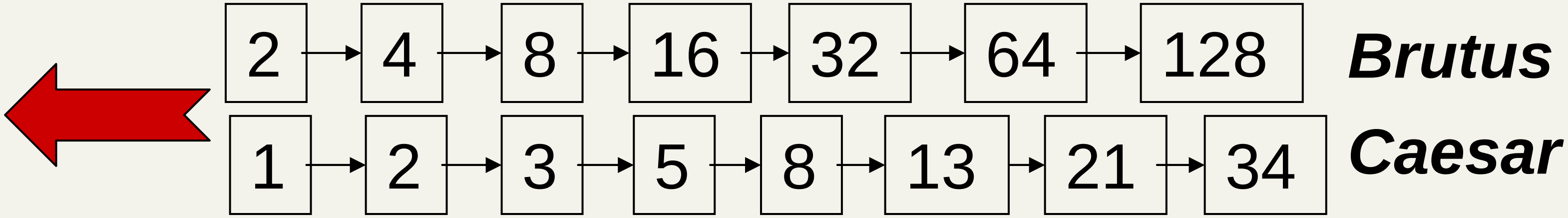
- How do we process a query?
  - What kinds of queries can we process?

Today's focus

- Which terms in a doc do we index?
  - All words or only "important" ones?
- Stopword list: terms that are so common that they're ignored for indexing.
  - *e.g.*, ***the, a, an, of, to*** …
  - language-specific.

Query processing

- 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 postings:

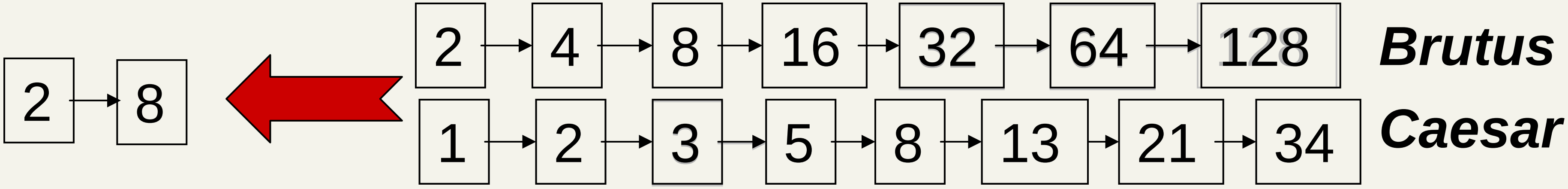


2 → 4 → 8 → 16 → 32 → 64 → 128 ***Brutus***

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 ***Caesar***

The merge

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



If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

Crucial: postings sorted by docID.

Merge algorithm

- Ex: $Term_0$ AND $Term_1$
- Index i_0 traverse $Post_0[0,\ldots,length_0-1]$
- Index i_1 traverse $Post_1[0,\ldots,length_1-1]$

```
$i_0=i_1=0$
Do While $i_0<length_0$ and $i_1<length_1${
    If $Post_1(i_1) = Post_0(i_0)$
        then hit!;  $i_0=i_0+1$; $i_1=i_1+1$
        else If  $Post_1(i_1) < Post_0(i_0)$ then $i_1=i_1+1$
                                             else  $i_0=i_0+1$
    }
```

Boolean queries: Exact match

- Queries using *AND, OR* and *NOT* together with query terms
  - Views each document as a set of words
  - Is precise: document matches condition or not.
- Primary commercial retrieval tool for 3 decades.
- Professional searchers (e.g., Lawyers) still like Boolean queries:
  - You know exactly what you’re getting.

More general merges

- What about the following queries:

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

  Brutus *OR NOT* ***Caesar***

Can we still run through the merge in time O(m+n)?

Ex: Term$_0$ AND NOT Term$_1$

- Index i_0 traverse $Post_0[0,\ldots,length_0-1]$
- Index i_1 traverse $Post_1[0,\ldots,length_1-1]$

```
$i_0=i_1=0$
Do While $i_0<length_0$ and $i_1<length_1$
    If $Post_1(i_1) > Post_0(i_0)$ then hit $Post_0(i_0)$! ;  $i_0=i_0+1$
        else If $Post_1(i_1) = Post_0(i_0)$ then  $i_0=i_0+1$; $i_1=i_1+1$
                                       else   $i_1=i_1+1$
    }
Do While $i_0<length_0$  hit $Post_0(i_0)$ ! ;  $i_0=i_0+1$
```

Ex: Term$_0$ OR NOT Term$_1$

- Index i_0 traverse $Post_0[0,\ldots,length_0-1]$
- Index i_1 traverse $Post_1[0,\ldots,length_1-1]$

```
$i_0=i_1=0$
Do While $i_0<length_0$ and $i_1<length_1$
    If $Post_1(i_1) > Post_0(i_0)$ then $i_0=i_0+1$;
          else if $Post_1(i_1) = Post_0(i_0)$ then
                    hit $(Post_1(i_1-1), Post_1(i_1)]$ ! $i_0=i_0+1$; $i_1=i_1+1$
                    else hit $(Post_1(i_1-1), Post_1(i_1))$! ; $i_1=i_1+1$
    }
Do While $i_1<length_1$ hit $(Post_1(i_1-1), Post_1(i_1))$! ; $i_1=i_1+1$
hit$(Post_1(length_1-1), maxdocid)$!;
```

Merging

What about an arbitrary Boolean formula?

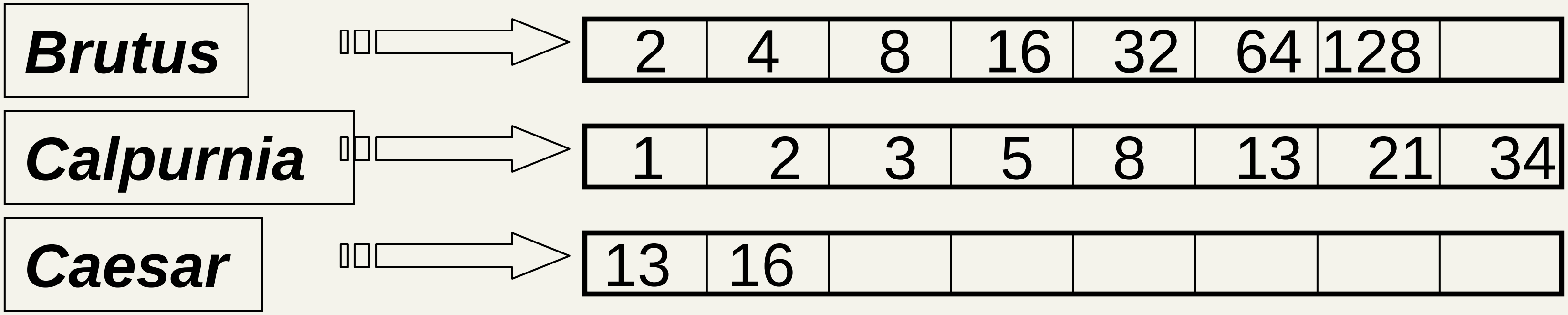
(Brutus *OR* ***Caesar)*** *AND NOT*

(Antony *OR* ***Cleopatra)***

- Can we always merge in "linear" time?
- Can we do better?

Query optimization

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



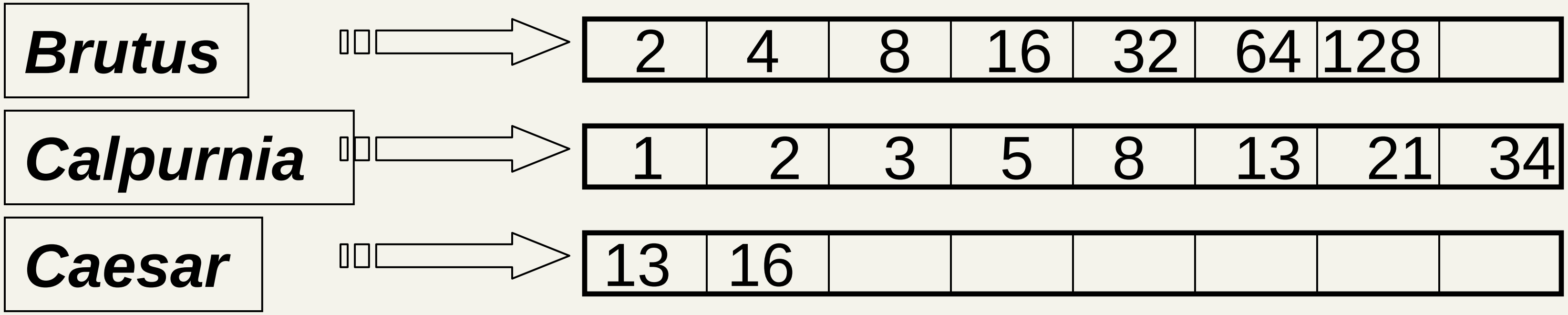
Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

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

Query optimization example

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

This is why we kept freq in dictionary



Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

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

More general optimization

- e.g., *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get freq’s for all terms.
- Estimate the size of each *OR* by the sum of its freq’s (conservative).
- Process in increasing order of *OR* sizes.

Exercise

- Recommend a query processing order for

(tangerine *OR* ***trees)*** *AND*
(marmalade *OR* ***skies)*** *AND*
(kaleidoscope *OR* ***eyes)***

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

Query processing exercises

- If the query is ***friends*** *AND* ***romans*** *AND (NOT* ***countrymen****)*, how could we use the freq of ***countrymen***?
- Exercise: Extend the merge to an arbitrary Boolean query. Can we always guaranteee execution in time linear in the total postings size? (Think of Conjunctive normal form)
- Hint: Begin with the case of a Boolean *formula* query: each query term appears only once in the query.

Query processing Excercise

- Can you process the query with only one traversal if all posting lists are in main memory?
- Ex: Term_0 AND Term_1 …. AND Term_{n-1}
- Index $i_k$traverse $\text{Post}_k[0,\ldots,\text{length}_k-1]$

```
$I_k=0$, $k=1,..,n$
$k=1$
Do While $i_{k-1 \text{mod } n} < \text{length}_{k-1 \text{mod } n}$
Do While $\text{Post}(i_k) < \text{Post}(i_{k-1 \text{ mod } n})$ $i_k=i_k+1$
 If $\text{Post}_k(i_k) = \text{Post}_{k-1}(i_{k-1 \text{ mod } n}) = \ldots\ldots = \text{Post}_{k-n+1 \text{mod } n} (i_{k-n+1 \text{ mod } n})$
          then hit! $i_k=i_k+1$, $k=1,..,n$
          else $k=k+1$ mod n
```

Query processing exercises

Process in linear time a CNF formula:

$(C_{11}$OR C_{12}... OR $C_{1k1})$ AND …..AND
$(C_{n1}$OR C_{n2}… OR $C_{nkn})$

Algorithm:

- If C_{ij}= NOT Term then use the Doc id intervals not containing Term while traversing the posting list of Term
- For each $(C_{i1}$OR C_{i2}... OR $C_{iki})$ implicitely consider the *posting interval list* I_i union of the intervals for every Term C_{ij} while traversing the posting lists
- Find Doc ids contained in all intervals I_1,….,I_n

Need all posting lists in main memory at the same time.

Digression: food for thought

- What if a doc consisted of *components*
  - Each component has its own *access control list.*
- Your search should get a doc only if your query meets one of its components that you have access to.
- More generally: doc assembled from *computations* on components
  - e.g., in Lotus databases or in content management systems
- Welcome to the real world … more later.

Beyond term search

- What about phrases?
- Proximity: Find ***Gates*** *NEAR* ***Microsoft***.
  - Need index to capture position information in docs. More later.
- Zones in documents: Find documents with (*author* = ***Ullman***) *AND* (text contains ***automata***).

Evidence accumulation

- 1 vs. 0 occurrence of a search term
    - 2 vs. 1 occurrence
    - 3 vs. 2 occurrences, etc.
- Need term frequency information in docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Need to measure proximity from query to each doc.
- Whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Structured vs unstructured data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows numerical range and exact match (for text) queries, e.g.,
Salary < 60000 AND Manager = Smith.

Unstructured data

- Typically refers to free text
- Allows
  - Keyword queries including operators
  - More sophisticated “concept” queries e.g.,
    - find all web pages dealing with *drug abuse*
- Classic model for searching text documents

Semi-structured data

- But in fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
    - *Title* contains data AND *Bullets* contain search

More sophisticated semi-structured search

- *Title* is about Object Oriented Programming AND *Author* something like stro*rup
- where * is the wild-card operator
- Issues:
  - how do you process “about”?
  - how do you rank results?
- The focus of XML search.

Clustering and classification

- Given a set of docs, group them into clusters based on their contents.
- Given a set of topics, plus a new doc *D*, decide which topic(s) *D* belongs to.

The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
- Beyond terms, exploit ideas from social networks
  - link analysis, clickstreams ...

Resources for today’s lecture

- IIR Chapter 1
- Shakespeare: http://www.rhymezone.com/shakespeare/
- Try the neat browse by keyword sequence feature!