



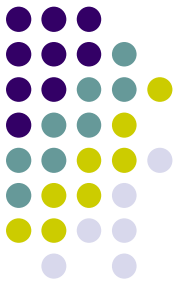
Advanced topics in Computer Science

Jiaheng Lu

Department of Computer Science

Renmin University of China

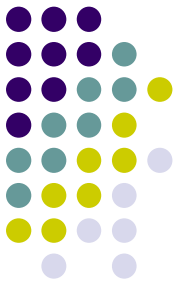
www.jiahenglu.net



Review:

Measuring information retrieval system by users' happiness

Precision, recall and F-measure

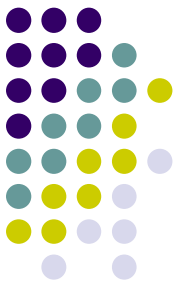


Precision and Recall

- **Precision:** fraction of retrieved docs that are relevant = $P(\text{relevant}|\text{retrieved})$
- **Recall:** fraction of relevant docs that are retrieved = $P(\text{retrieved}|\text{relevant})$

	Relevant	Not Relevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision $P = \text{tp}/(\text{tp} + \text{fp})$
- Recall $R = \text{tp}/(\text{tp} + \text{fn})$

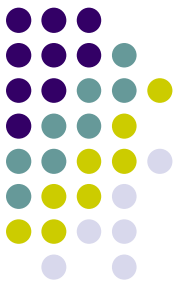


A combined measure: F

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

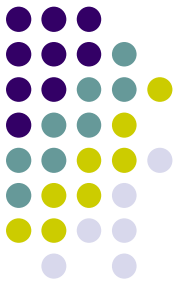
- People usually use balanced F_1 measure
 - i.e., with $\beta = 1$ or $\alpha = \frac{1}{2}$



Online text book:

Introduction to Information Retrieval

<http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html>



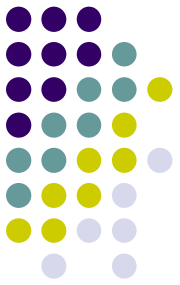
IT news:

New direction in teaching computer science emphasizes activity, interaction, critique

<http://news-info.wustl.edu/tips/page/normal/13517.html>

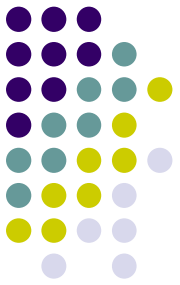
"At the heart of active learning is the hallmark of interactive face time and students taking a more active role and not just repeating what a professor wants to hear,"

"Students learn the art of critiquing substantively without ruffling feathers," she said. "they also want them to be familiar with teamwork, explaining their ideas, and being comfortable with the critique. We haven't taught that as much until just recently."



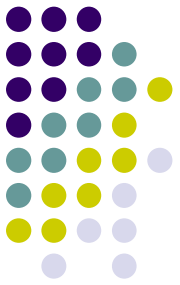
Introduction to Information Retrieval System

Query



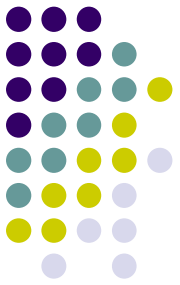
- Which plays of Shakespeare contain the words ***Brutus AND Caesar*** but ***NOT Calpurnia***?
- Could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?
 - Slow (for large corpora)
 - *NOT Calpurnia* is non-trivial
 - Other operations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

Term-document incidence



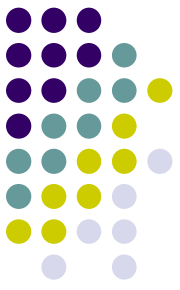
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

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*.
- $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$.



Answers to query

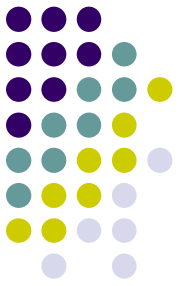
- Antony and Cleopatra, Act III, Scene ii

- *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.

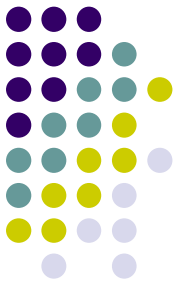
- Hamlet, Act III, Scene ii

- *Lord Polonius*: I did enact Julius **Caesar** I was killed i' the Capitol; **Brutus** killed me.

Bigger document collections

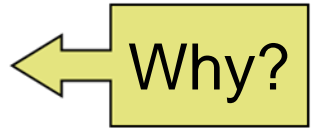


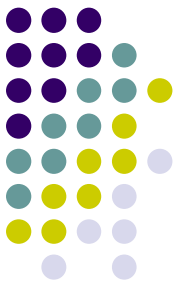
- Consider $N = 1$ million documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
 - 6GB of data in the documents.
- Say there are $M = 500K$ distinct terms among these.



Can't build the matrix

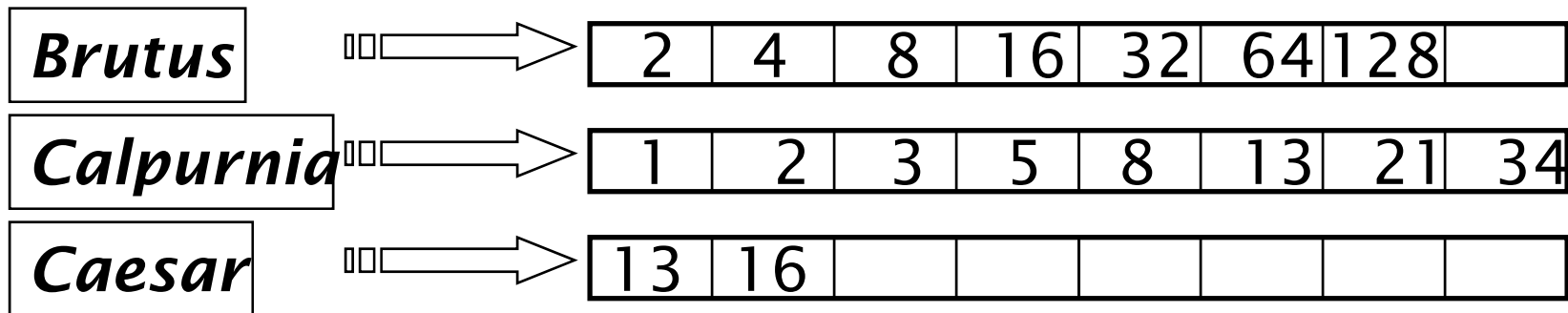
- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's.
 - matrix is extremely sparse.
- What's a better representation?
 - We only record the 1 positions.



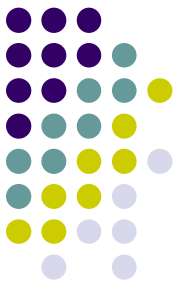


Inverted index

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

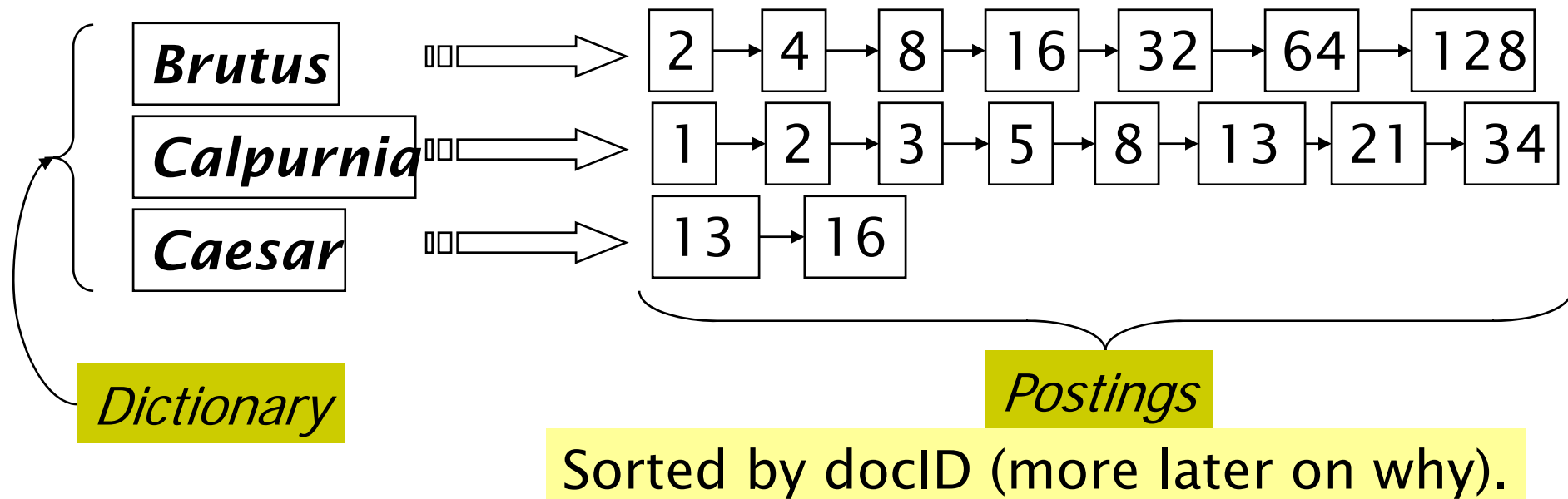


What happens if the word *Caesar* is added to document 14?

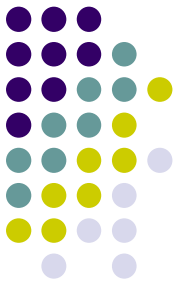


Inverted index

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



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

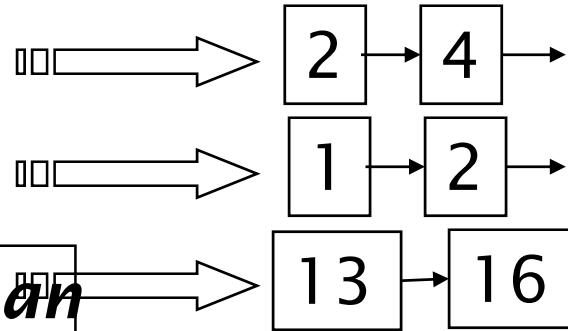
Indexer

Inverted index.

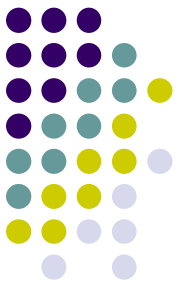
friend

roman

countryman



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

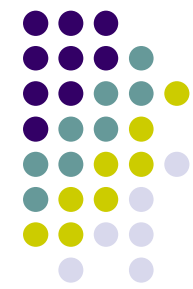
- Sort by terms.

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

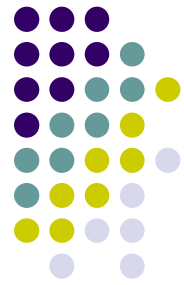


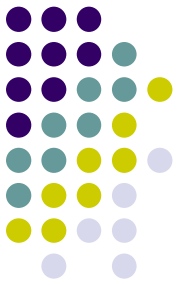
- Multiple term entries in a single document are merged.
- Frequency information is added.

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





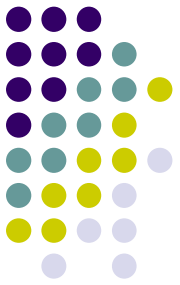
- The result is split into a *Dictionary* file and a *Postings* file.

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	1
2	1
2	1
1	1
2	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?

Terms →

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

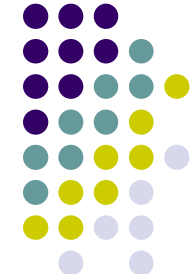
↑
Pointers

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

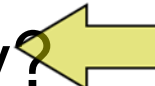
Will quantify the storage, later.

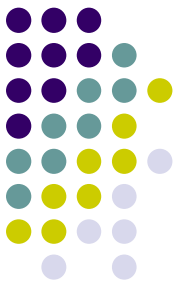
The index we just built

- How do we process a Boolean query?
 - Later - what kinds of queries can we process?



Today's
focus



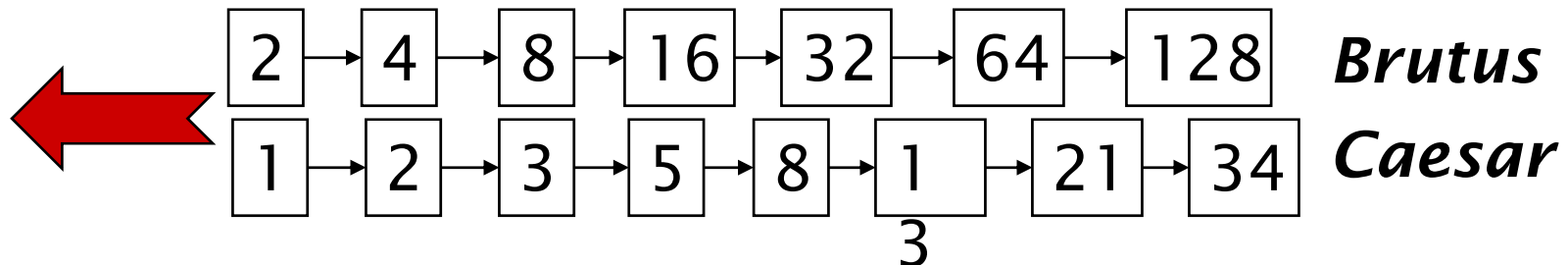


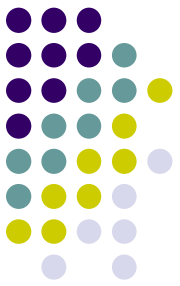
Query processing

- Consider processing the query:

Brutus AND Caesar

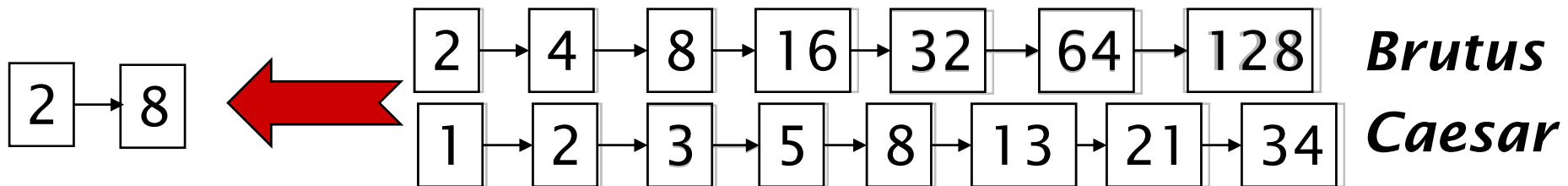
- Locate ***Brutus*** in the Dictionary;
 - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
 - Retrieve its postings.
- “Merge” the two postings:





The merge

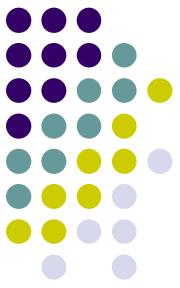
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are x and y , the merge takes $O(x+y)$ operations.

Crucial: postings sorted by docID.

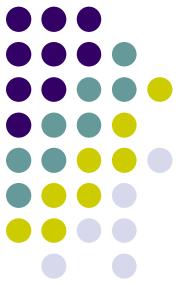
Basic postings intersection



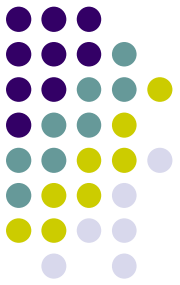
```
INTERSECT( $p_1, p_2$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $docID(p_1) = docID(p_2)$ 
4      then  $\text{ADD}(answer, docID(p_1))$ 
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7  else if  $docID(p_1) < docID(p_2)$ 
8      then  $p_1 \leftarrow next(p_1)$ 
9      else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 
```

► **Figure 1.7** Algorithm for the intersection of two postings lists p_1 and p_2 .

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.



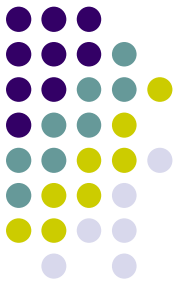
More general merges

- Exercise: Adapt the merge for the queries:

Brutus AND NOT Caesar

Brutus OR NOT Caesar

Can we still run through the merge in time $O(x+y)$?



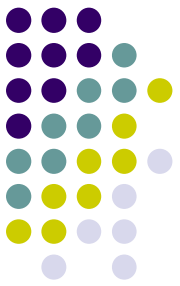
Merging

What about an arbitrary Boolean formula?

(Brutus OR Caesar) AND NOT

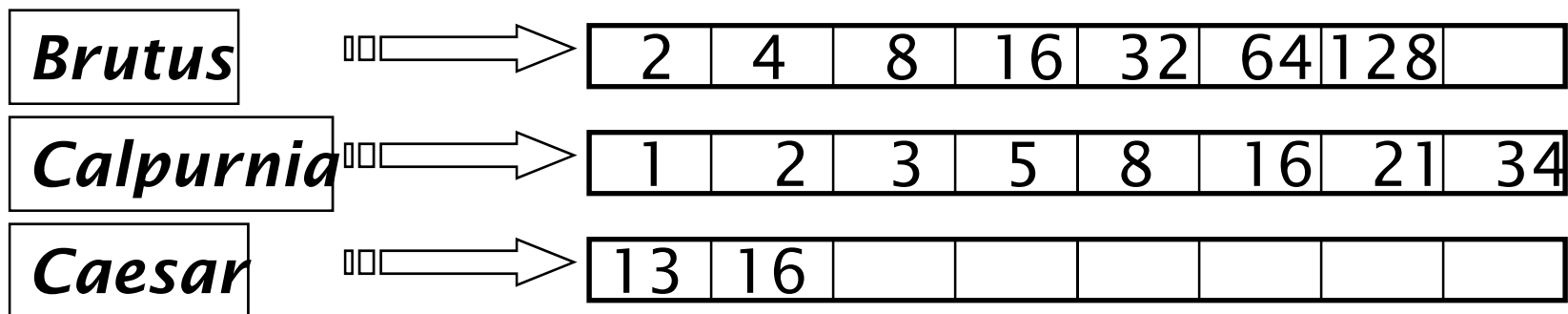
(Antony OR Cleopatra)

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

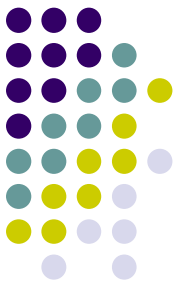


Query optimization

- 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.



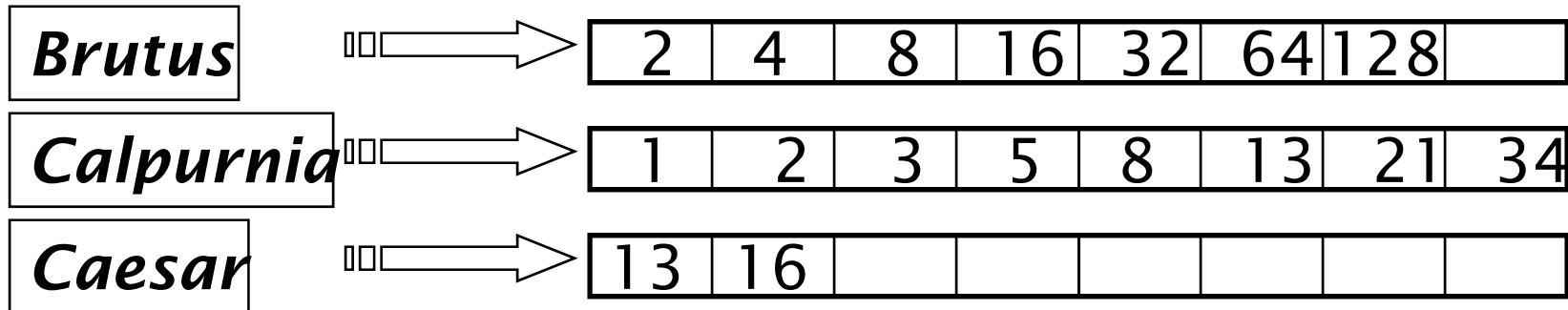
Query: **Brutus AND Calpurnia AND Caesar**



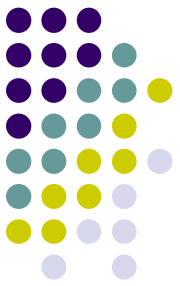
Query optimization example

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

This is why we kept
freq in dictionary



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



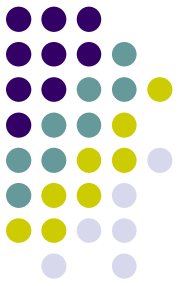
Exercise

- Recommend a query processing order for

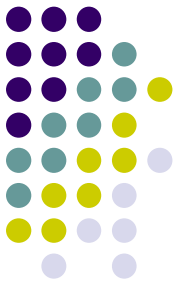
*(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)*

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*?



Greedy optimization

- (Process in increasing order of term frequency):
- Is this always guaranteed to be optimal?