

CMPE 493  
INTRODUCTION TO  
INFORMATION RETRIEVAL

Boolean Retrieval  
and the Inverted Index

Department of Computer Engineering, Boğaziçi University  
October 26, 2020

Boolean retrieval

- ▶ 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.**
- ▶ **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; Thousands of users
- ▶ Majority of users *still* use boolean queries
- ▶ Example query:
  - ▶ *Information need:* Cases about a host's responsibility for drunk guests
  - ▶ *Query:* host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest
  - ▶ /p = in same paragraph

3

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

- ▶ Another example query:
  - ▶ Requirements for disabled people to be able to access a workplace
  - ▶ disabl! /p access! /s work-site work-place (employment /3 place)
- ▶ Note that SPACE is disjunction, not conjunction!
- ▶ Long, precise queries; proximity operators; incrementally developed; not like web search
- ▶ Many professional searchers still like Boolean search
  - ▶ You know exactly what you are getting
- ▶ But that doesn't mean it actually works better....

4

## Does Google use the Boolean model?

- ▶ On Google, the default interpretation of a query  $[w_1 w_2 \dots w_n]$ 
  - ▶ is  $w_1$  AND  $w_2$  AND  $\dots$  AND  $w_n$
- ▶ Cases where you get hits that do not contain one of the  $w_i$  :
  - ▶ anchor text
  - ▶ page contains variant of  $w_i$  (morphology, spelling correction, synonym)
  - ▶ long queries ( $n$  large)
  - ▶ boolean expression generates very few hits
- ▶ Simple Boolean vs. Ranking of result set
  - ▶ Simple Boolean retrieval returns matching documents in no particular order.
  - ▶ Google (and most well designed search engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

## Unstructured data in 1680: Shakespeare

- ▶ Shakespeare's Collected Works (~ one million words)
- ▶ Which plays of Shakespeare contain the words **Brutus** AND **Caesar** but NOT **Calpurnia**?
- ▶ One could grep all of Shakespeare's plays for **Brutus** and **Caesar**, then strip out lines containing **Calpurnia**?
- ▶ Why is grep not the solution?
  - ▶ Slow (for large corpora)
  - ▶ Ranked retrieval (best documents to return)
    - ▶ Later lectures

## Binary 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

Shakespeare used  
around 32,000  
different words

**Brutus AND Caesar BUT NOT  
Calpurnia**

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.

	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

## 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.



9

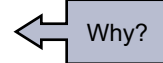
## Bigger collections

- ▶ Consider  $N = 1$  million documents, each with about 1000 words.
- ▶ Avg 6 bytes/word including spaces/punctuation
  - ▶ 6GB of data in the documents.
- ▶ Say there are  $M = 500,000$  *distinct* terms among these.

10

## Can't build the matrix

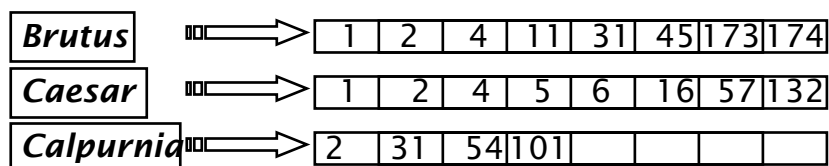
- ▶ 500K x 1M matrix has half-a-trillion 0's and 1's.
- ▶ But it has no more than one billion 1's.
  - ▶ matrix is extremely sparse.
- ▶ What's a better representation?
  - ▶ We only record the 1 positions.



11

## Inverted index

- ▶ For each term  $t$ , we must store a list of all documents that contain  $t$ .
  - ▶ Identify each by a **docID**, a document serial number
- ▶ Can we use fixed-size arrays for this?

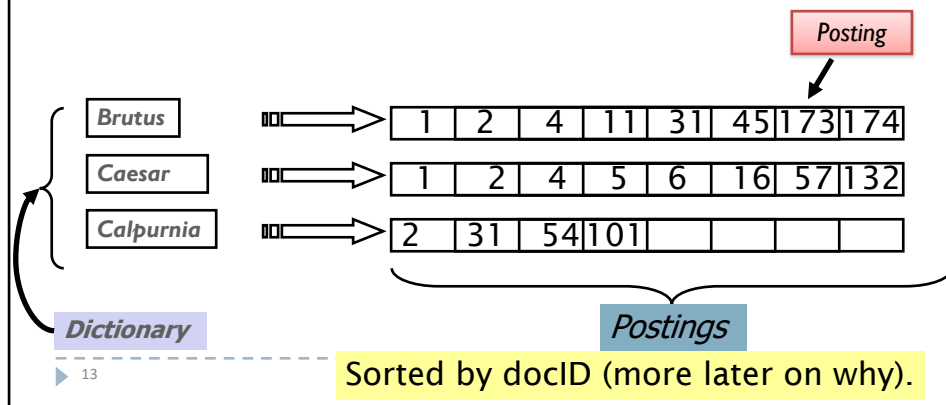


What happens if the word **Caesar** is added to document 14?

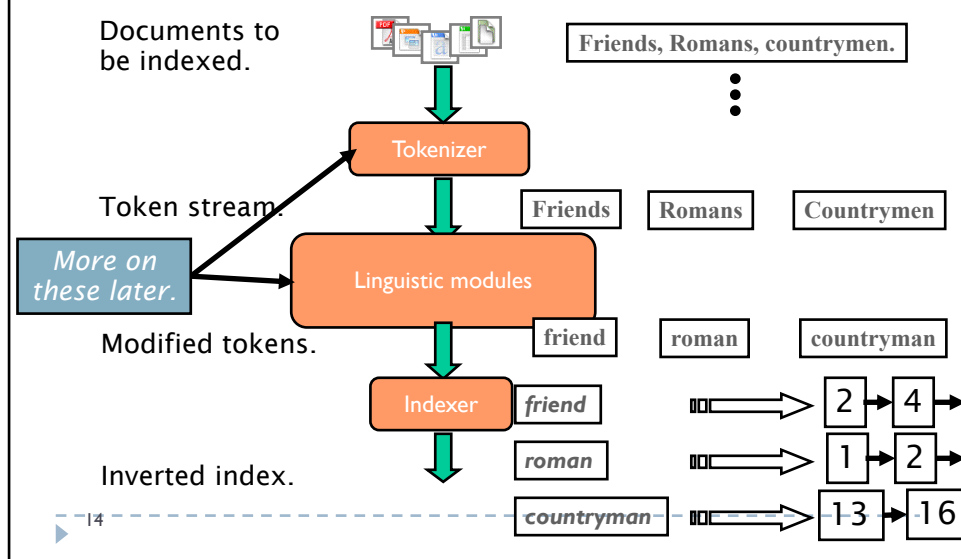
12

## 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



## Inverted index construction



## Indexer steps: Token sequence

- 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

15

## Indexer steps: Sort

- Sort by terms

And then docID

**Core indexing step**

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
it	1
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

16



## 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.

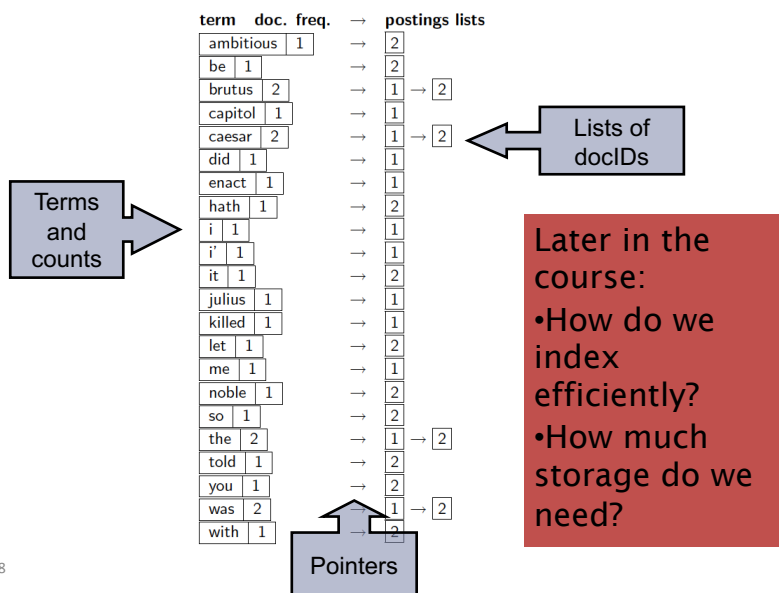
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



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

17

## Where do we pay in storage?



18

## The index we just built

- ▶ How do we process a query?

Today's focus

- ▶ Later - what kinds of queries can we process?

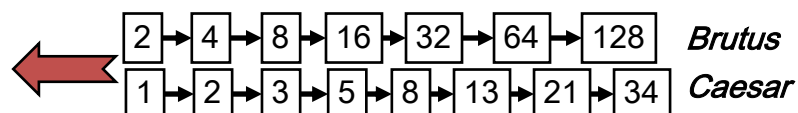
19

## Query processing: AND

- ▶ Consider processing the query:

***Brutus AND Caesar***

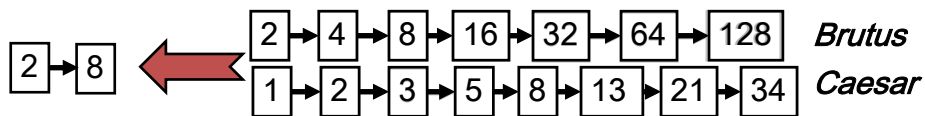
- ▶ Locate ***Brutus*** in the Dictionary;
    - ▶ Retrieve its postings.
  - ▶ Locate ***Caesar*** in the Dictionary;
    - ▶ Retrieve its postings.
  - ▶ “Merge” (Intersect) the two postings:



20

## 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.

21

## Intersecting two postings lists (a “merge” algorithm)

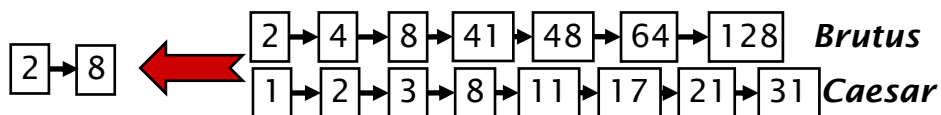
```
INTERSECT( $p_1, p_2$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(answer, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow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

22

## Faster postings merges: Skip pointers/Skip lists

### Recall basic merge

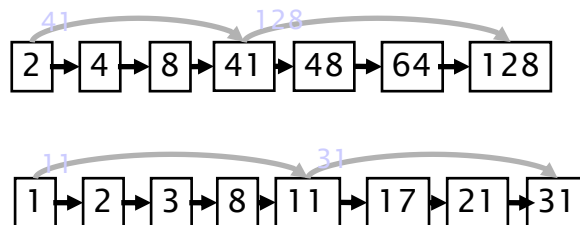
- ▶ Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

Can we do better?  
Yes (if index isn't changing too fast).

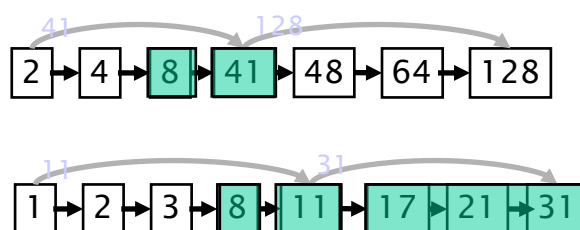
Augment postings with skip pointers (at indexing time)



► Why?

► To skip postings that will not figure in the search results.

Query processing with skip pointers



Suppose we've stepped through the lists until we process 8 on each list. We match it and advance.

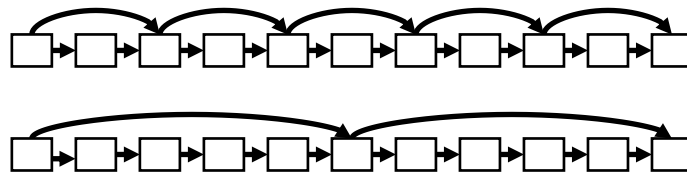
We then have 41 and 11 on the lower. 11 is smaller.

The skip successor of 11 on the lower list is 31, so we can skip ahead past the intervening postings.

## Where do we place skips?

### ► Tradeoff:

- More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
- Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.

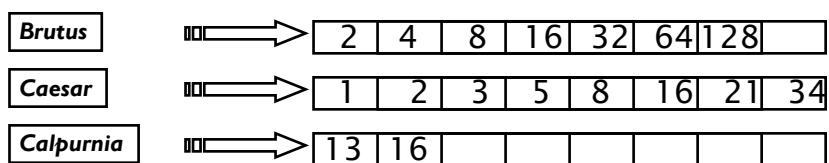


## Placing skips

- Simple heuristic: for postings of length  $L$ , use  $\sqrt{L}$  evenly-spaced skip pointers.
- Easy if the index is relatively static; harder if  $L$  keeps changing because of updates.
- This definitely used to help; with modern hardware it may not unless you're memory-based
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

## Query optimization

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



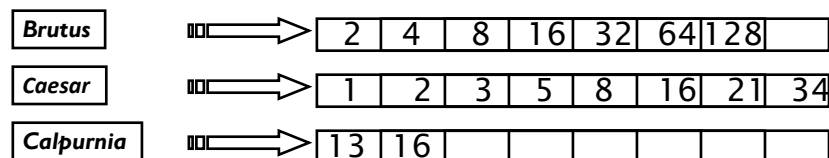
Query: **Brutus AND Caesar AND Calpurnia**

29

## Query optimization example

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

This is one of the reasons why we kept document freq. in dictionary



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

30

## More general optimization

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

31

## 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

32



## What's ahead in IR?

### Beyond term search

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

33

## Evidence accumulation

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

34

## Ranking search results

- ▶ Boolean queries give inclusion or exclusion of docs.
- ▶ Often we want to rank/group results
  - ▶ Need to measure proximity from query to each doc.

35

## 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 process queries with wild-card?

36

## Clustering, classification and ranking

- ▶ **Clustering:** Given a set of docs, group them into clusters based on their contents.
- ▶ **Classification:** Given a set of topics, plus a new doc  $D$ , decide which topic(s)  $D$  belongs to.
- ▶ **Ranking:** Can we learn how to best order a set of documents, e.g., a set of search results

37

## 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 ...
- ▶ How do search engines work? And how can we make them better?

38

## More sophisticated *information* retrieval

- ▶ Cross-language information retrieval
- ▶ Question answering
- ▶ Summarization
- ▶ Text mining
- ▶ ...

39

## Resources for today's lecture

- ▶ *Introduction to Information Retrieval*, chapter 1
- ▶ Content adapted from the IR book's web site.
- ▶ Shakespeare:
  - ▶ <http://www.rhymezone.com/shakespeare/>

40