Boolean Model

- Based on set theory and Boolean algebra
- Reality much more a **data retrieval** model

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

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

Incidence vectors

- So we have a 0/1 vector for each term.
- Take the vectors:
  - ***Brutus*** 110100
  - ***aesar*** 110111
  - ***Calpurnia*** 010000

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

- To answer query:
  - 110100 *AND* 110111 *AND* 101111 = **100100**

Bigger corpora(语料库)

- Consider
  - ***N*** = 1 000 000 documents
  - each with about 1000 terms.
  - Avg **6 bytes/term**
    - include spaces/punctuation(空间/标点符号)
  - **6GB** of data in the documents.
- Say there are
  - ***m*** = 500 000
    - distinct terms among these.

11

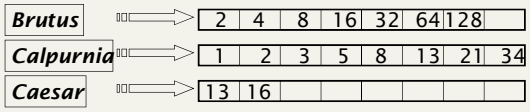
Can't build the matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
  - trillion(万亿)
  - M(百万 1 000 000)
  - K(千 1 000)
- But it has no more than one billion 1's.
  - matrix is extremely sparse(稀疏).
- What's a better representation?
  - We only record the 1 positions.

12

Inverted index(倒排索引)

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

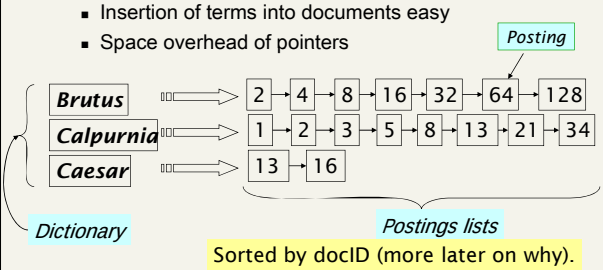


What happens if the word *Caesar* is added to document 14?

13

Inverted index

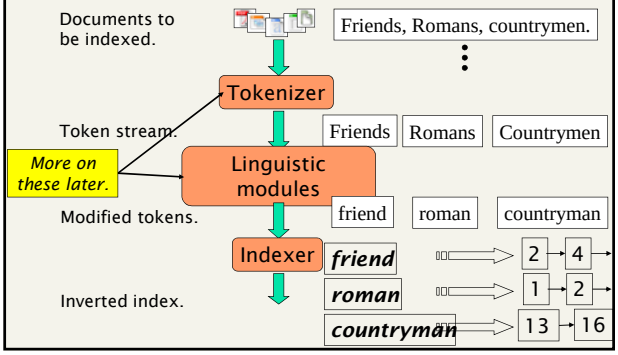
- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers



Sorted by docID (more later on why).

14

Inverted index construction



15

Indexer steps

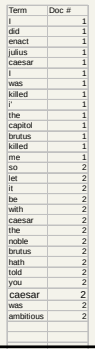
- Sequence of **(Modified token, Document ID)** pairs.

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

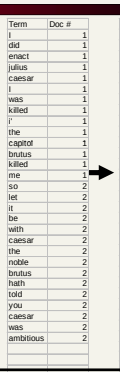


Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

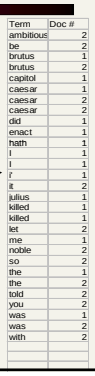
16

- 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



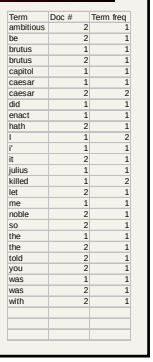
Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

17

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

Why frequency? Will discuss later.

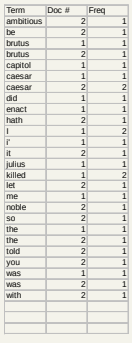
Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Term	Doc #	Term 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

18

- 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	Coll 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
1	1
2	1
2	1
1	1
2	1
2	1

21

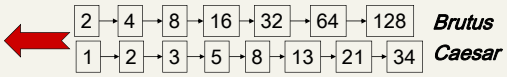
The index we just built

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

22

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 postings:



23

Algorithm : Intersect(p1,p2)

Algorithm for the intersection of two postings lists p1 and p2

```
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
```

25

Boolean queries: Exact match(精确匹配)

- The Boolean Retrieval model is being able to ask a query
  - Boolean Queries that is a Boolean expression:
  - using ***AND***, *OR* and *NOT* to join query terms
    - Views each document as a **set** of words
    - Is precise : document matches condition or not
- Professional searchers (e.g.,) still like Boolean queries:
  - Can know exactly what you're getting

26

Boolean queries: More general merges

Exercise:

- Adapt the merge for the queries:

  Brutus** AND NOT **Caesar

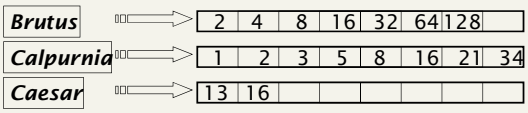
  Brutus** OR NOT **Caesar

Can we still run through the merge in time O(*x+y*) or what can we achieve?

29

Query optimization(查询优化)

- 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* them together.



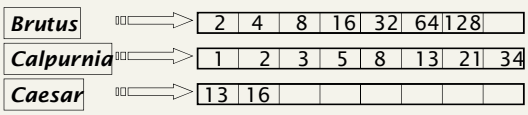
Query: ***Brutus** AND **Calpurnia** AND **Caesar***

30

Query optimization example

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

This is why we kept frequency in dictionary



Executed query : (***Caesar** AND **Brutus**) AND **Calpurnia***

Algorithm : Intersect(<t_1,t_2,…,t_n>)

Algorithm for conjunctive queries that returns the set of documents containing each term in the input list of terms

```
INTERSECT(⟨t1, ..., tn⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t1, ..., tn⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

32

Exercise

If the query is:

friends AND romans AND (NOT countrymen)

- 对比：百度、搜狗、Bing 的检索结果
- 对比：Baidu学术、Bing学术、Google学术 的检索结果

33

More general optimization

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

34

Exercise

- Query: ***(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

(tangerine** OR **trees) *46653+316812 = 363465*
(marmalade** OR **skies) *107913+271658 = 379571*
(kaleidoscope** OR **eyes) *87009+213312 = 300321*