

Introduction to  
Assignment Project Exam Help  
**Information Retrieval**  
<https://powcoder.com>

Add WeChat powcoder  
Lecture 1: Boolean retrieval

Introduction to  
Assignment Project Exam Help  
**Information Retrieval**  
<https://powcoder.com>

Add WeChat powcoder  
Term-document incidence matrices

# Unstructured data in 1620

---

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but NOT ***Calpurnia***?
- One could ~~Assignment Project Exam Help~~ **grep** all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?  
~~<https://powcoder.com>~~
- Why is that not the answer?  
~~Add WeChat powcoder~~
  - Slow (for large corpora)
  - NOT ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)
    - Later lectures

# Term-document incidence matrices

	Antony and Cleopatra	Julius Caesar	The Taming of the Shrew	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

*Brutus AND Caesar BUT NOT Calpurnia*

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

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.

Add WeChat powcoder

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

# Answers to query

- Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,

Assignment Project Exam Help

When Antony found Julius *Caesar* dead,

He cried almost to roaring; and he wept

When at Philippi he found *Brutus* slain.

Add WeChat powcoder

- Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius *Caesar* I was killed i' the

Capitol; *Brutus* killed me.



# Bigger collections

---

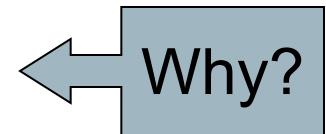
- Consider  $N = 1$  million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.  
<https://powcoder.com>
- Say there are  $M = 500K$  *distinct* terms among these.  
Add WeChat powcoder

# 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.

Assignment Project Exam Help

Add WeChat powcoder



Assignment Project Exam Help

# Information Retrieval

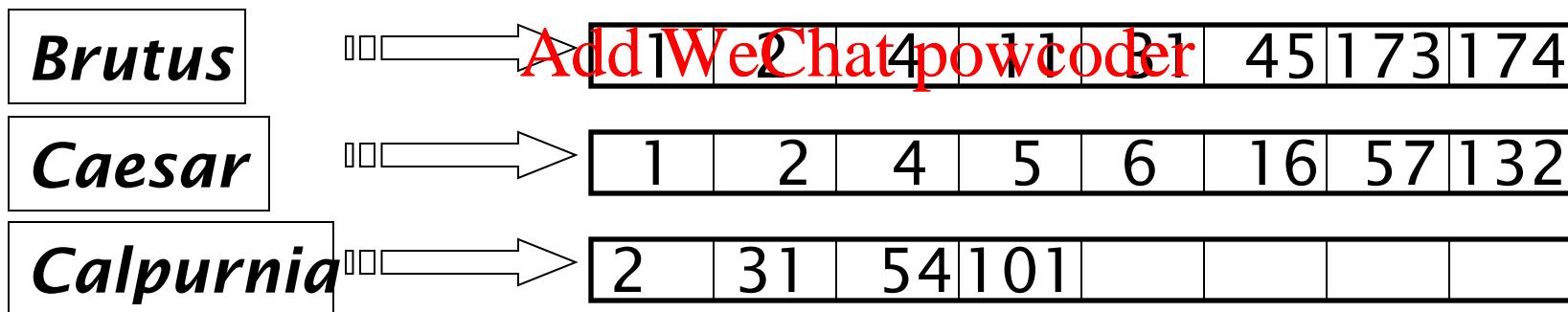
<https://powcoder.com>

Add WeChat powcoder  
The Inverted Index

The key data structure underlying modern IR

# 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?  
<https://powcoder.com>

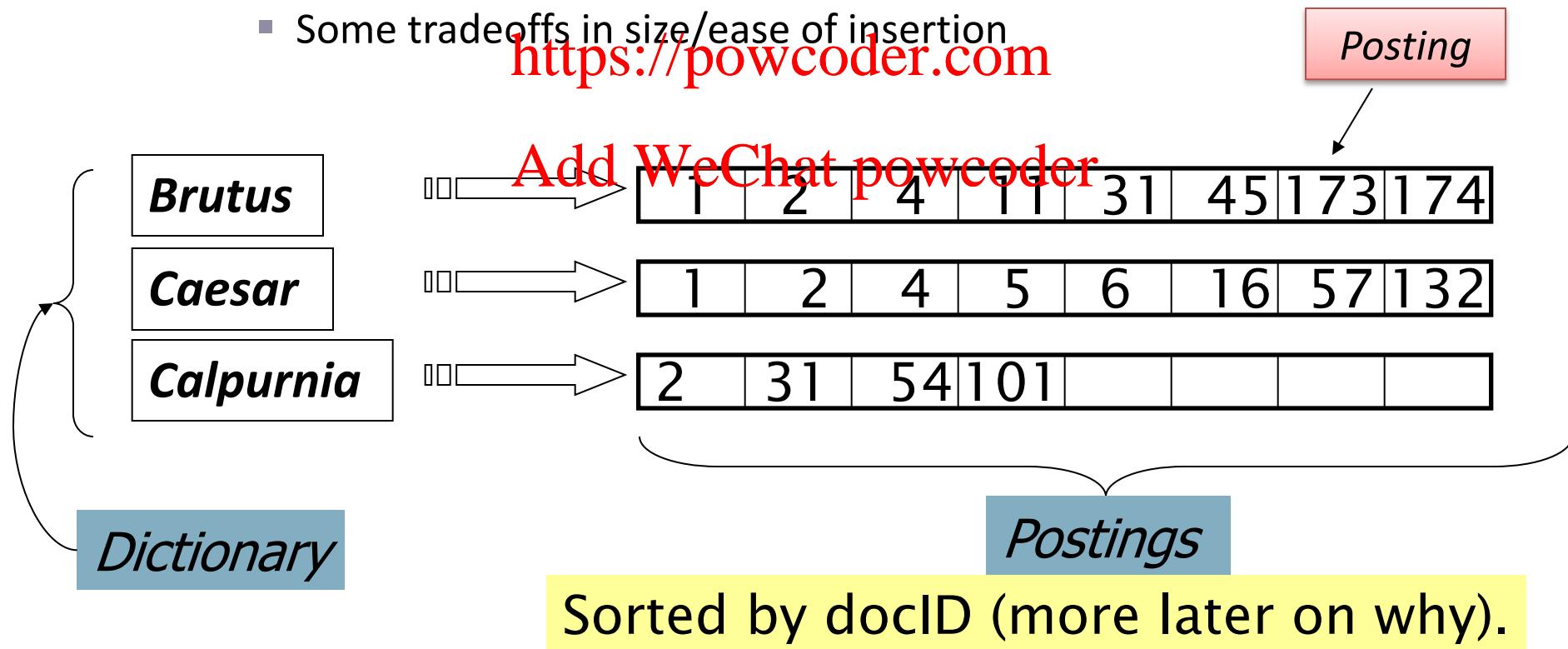


What happens if the word **Caesar** is added to document 14?

# 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

<https://powcoder.com>



# Inverted index construction

Documents to be indexed.



Friends, Romans, countrymen.

⋮

Assignment Project Exam Help

Tokenizer

https://powcoder.com

Token stream.

Friends

Romans

Countrymen

*More on these later.*

Add WeChat powcoder  
Linguistic modules

Modified tokens.

friend

roman

countryman

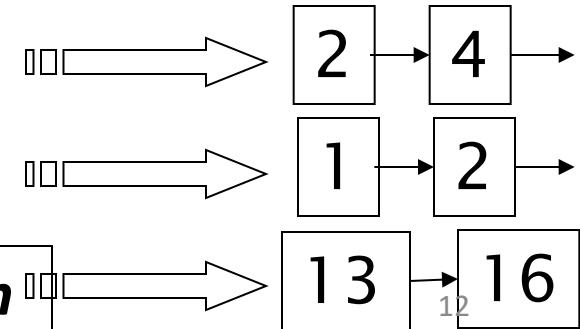
Indexer

*friend*

*roman*

*countryman*

Inverted index.



# Indexer steps: Token sequence

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

# Assignment Project Exam Help

<https://powcoder.com>

Doc 1

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

# Add WeChat powcoder

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 by terms

- And then docID

# Assignment Project Exam Help

<https://powcoder.com>

## Core indexing step

# Add WeChat powcoder

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.

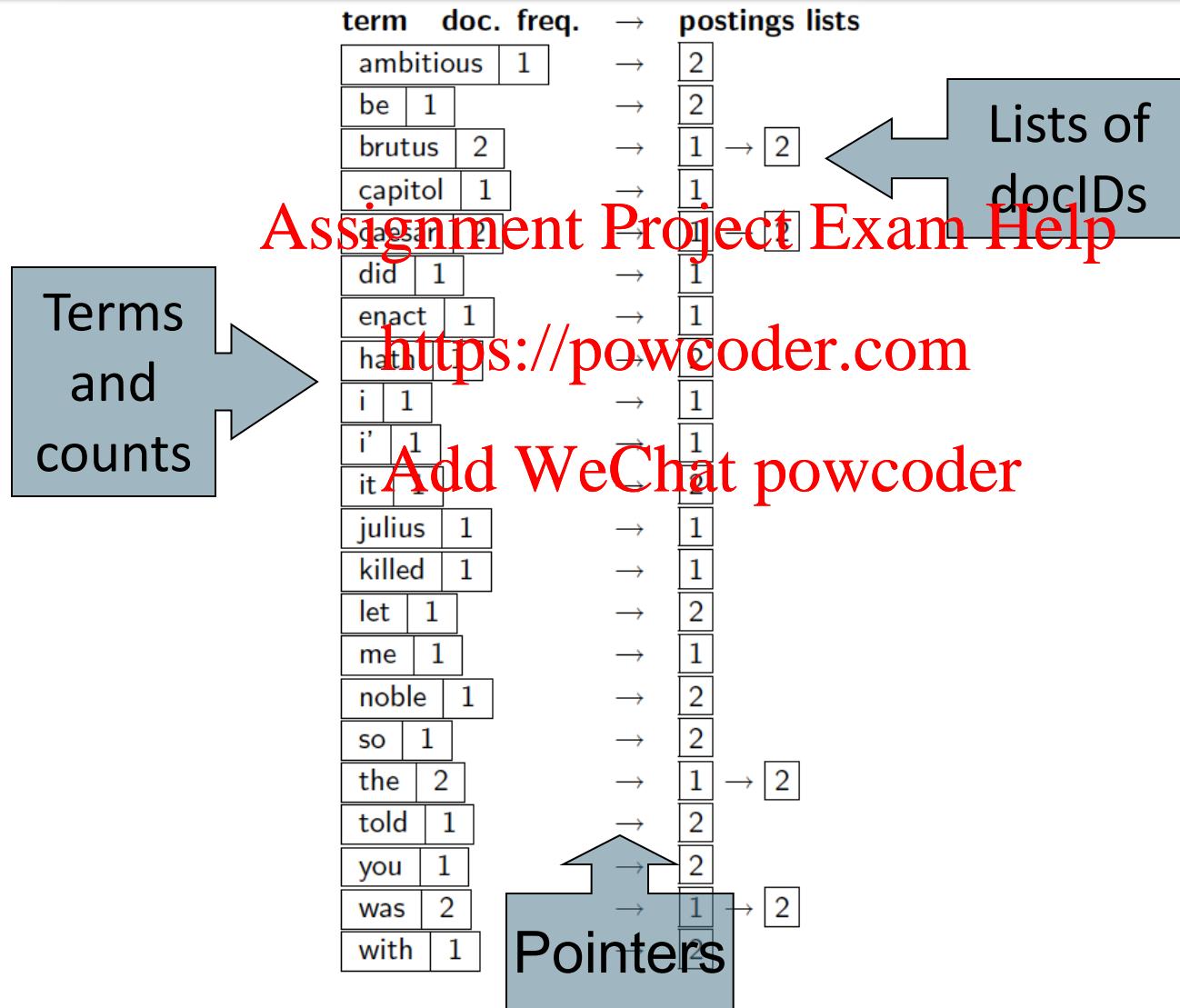
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Assignment Project Exam Help  
<https://powcoder.com>  
 Add WeChat powcoder



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?



Introduction to  
Assignment Project Exam Help  
**Information Retrieval**  
<https://powcoder.com>

Add WeChat powcoder  
Query processing with an inverted index

# The index we just built

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



Assignment Project Exam Help

<https://powcoder.com>

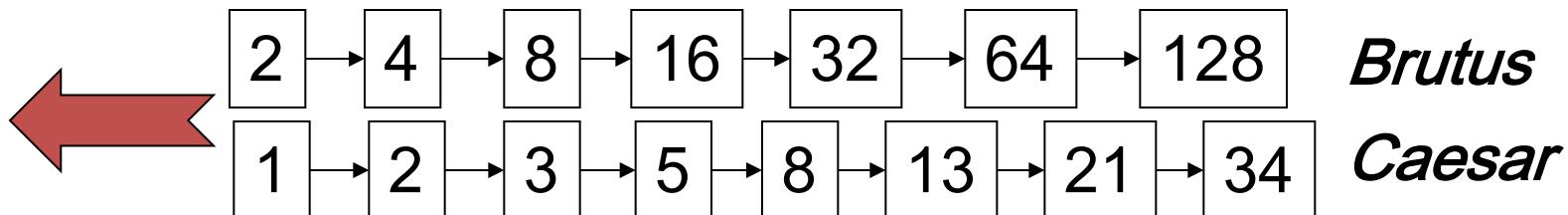
Add WeChat powcoder

# Query processing: AND

- Consider processing the query:

***Brutus AND Caesar***

- Locate *Brutus* in the Dictionary;
  - Retrieve its postings,  
<https://powcoder.com>
- Locate *Caesar* in the Dictionary;
  - Retrieve its postings.  
[Add WeChat powcoder](#)
- “Merge” the two postings:

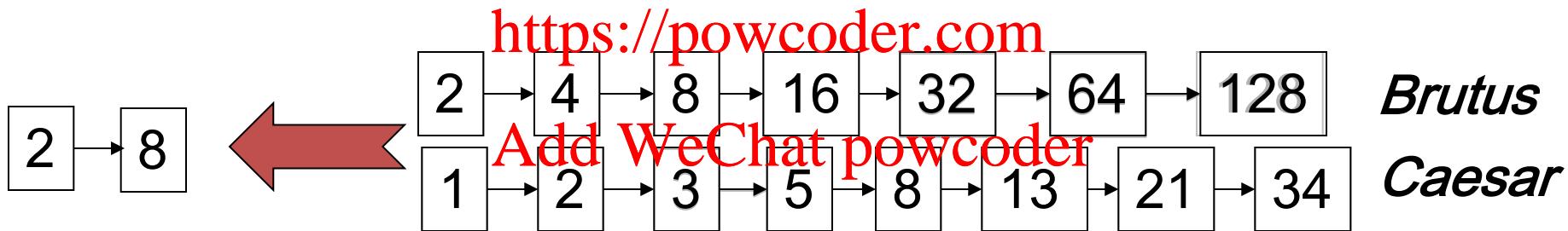


***Brutus***  
***Caesar***

# The merge

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

Assignment Project Exam Help



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

Crucial: postings sorted by docID.

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

INTERSECT( $p_1, p_2$ )

```
1  answer ← ⟨ ⟩  
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 ADD(answer,  $\text{docID}(p_1)$ )  
5          Add WeChat powcoder  
6           $p_1 \leftarrow \text{next}(p_1)$   
7           $p_2 \leftarrow \text{next}(p_2)$   
8      else if  $\text{docID}(p_1) < \text{docID}(p_2)$   
9          then  $p_1 \leftarrow \text{next}(p_1)$   
10         else  $p_2 \leftarrow \text{next}(p_2)$   
10 return answer
```

# Boolean queries: Exact match

---

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:  
~~Assignment Project Exam Help~~
  - Boolean Queries are queries using *AND*, *OR* and *NOT* to join query terms  
~~https://powcoder.com~~
    - 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; new federated search added 2010)  
~~Assignment Project Exam Help~~
- Tens of terabytes of data, ~700,000 users
- Majority of users still use boolean queries  
~~Add WeChat powcoder~~
- 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
    - foo! = foo\*, /3 = within 3 words, /S = in same sentence

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

---

- Another example query:
  - Requirements for disabled people to be able to access a workplace
  - disabl! /p address! /s work site workplace (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...

# Boolean queries:

## More general merges

---

- Exercise: Adapt the merge for the queries:

***Brutus AND NOT Caesar***

Assignment Project Exam Help

***Brutus OR NOT Caesar***

<https://powcoder.com>

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

What can we achieve?

# Merging

---

What about an arbitrary Boolean formula?

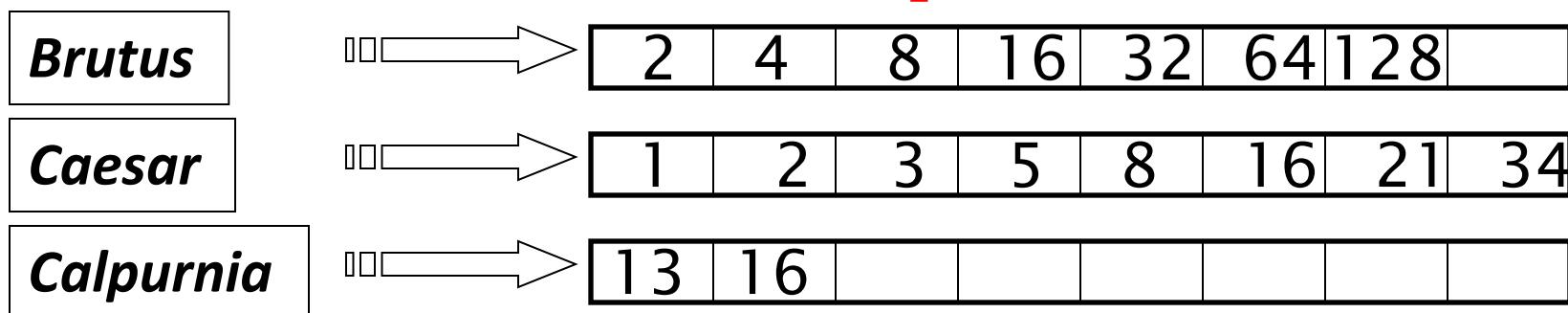
**(Brutus OR Caesar) AND NOT  
(Antony OR Cleopatra)**

- 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  $n$  terms.
- For each of the  $n$  terms, get its <https://powcoder.com> postings, then AND them together.

Add WeChat powcoder



Query: **Brutus AND Calpurnia AND Caesar**

# Query optimization example

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

Assignment Project Exam Help

This is why we keep  
<https://powcoder.com>  
document freq. in dictionary

Add WeChat powcoder

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

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

# More general optimization

- e.g., *(madding OR crowd) AND (ignoble OR strife) AND (light OR lord)*
- 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.

Q: When will the conservative estimate fail?

# Exercise

- Recommend a query processing order for

Assignment Project Exam Help

*(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

**Q:** Can we merge multiple lists (>2) simultaneously?

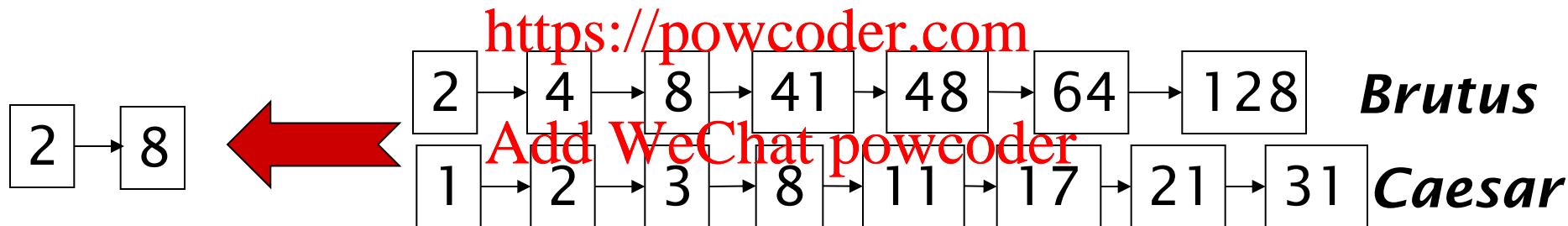
Introduction to  
Assignment Project Exam Help  
**Information Retrieval**  
<https://powcoder.com>

Add WeChat powcoder  
Faster postings merges. Skip pointers and  
Skip lists

# Recall basic merge

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

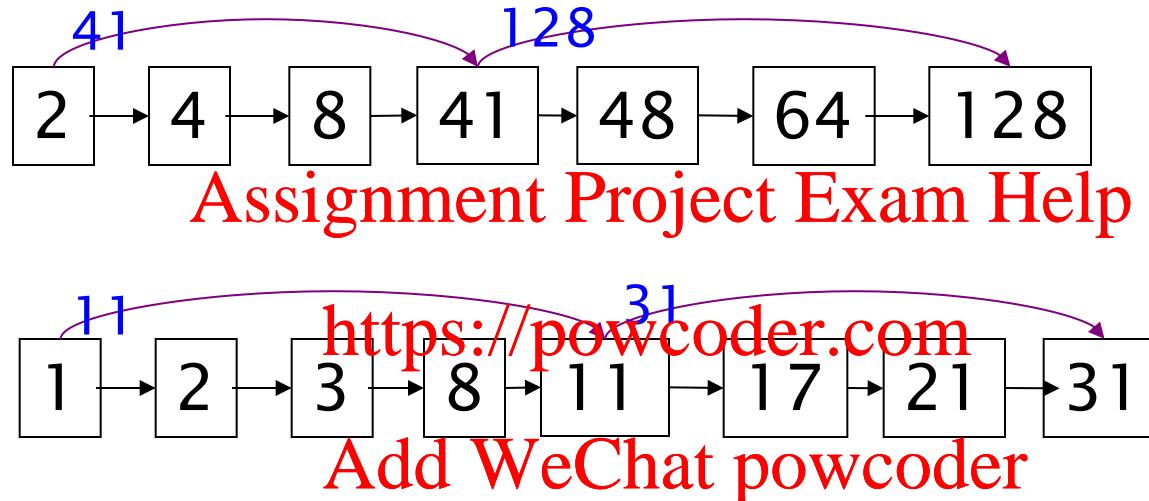
Assignment Project Exam Help



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

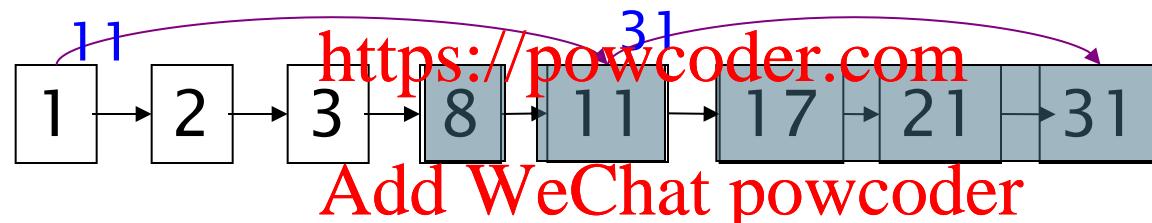
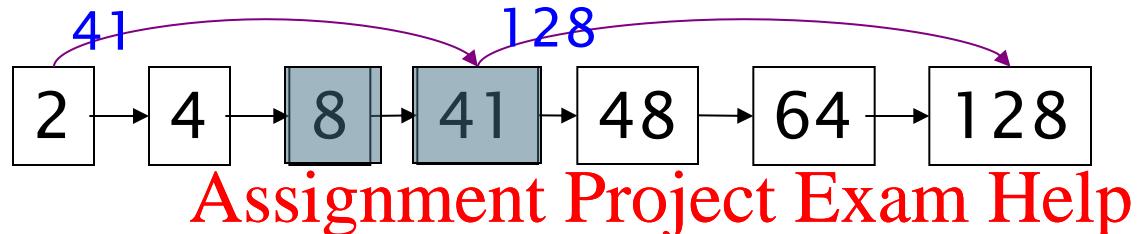
Can we do better?  
Yes (if index isn't changing too fast).

# Augment postings with skip pointers (at indexing time)



- Why?
- To skip postings that will not figure in the search results.
- How?
- Where do we place skip pointers?

# Query processing with skip pointers



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

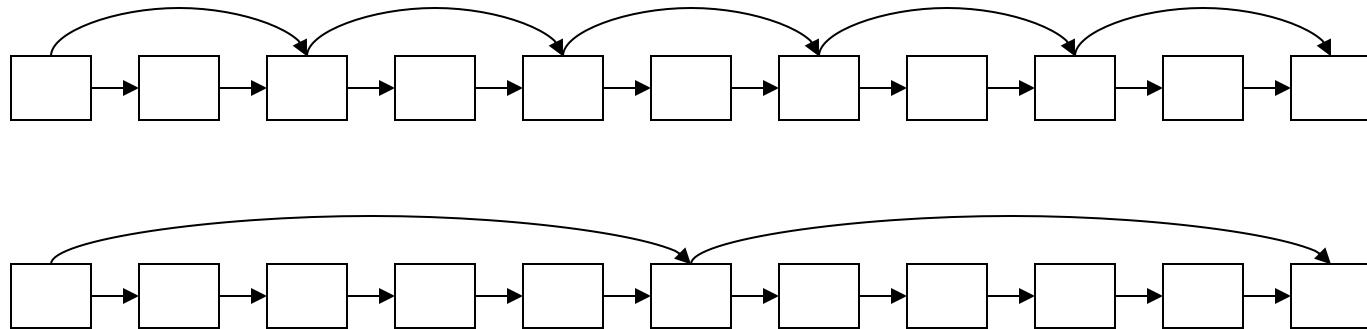
We then have 41 and 11 on the lower. 11 is smaller.

But 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.

Add WeChat powcoder



# Placing skips

---

- Simple heuristic: for postings of length  $L$ , use  $L^{1/2}$  evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relative static, harder if  $L$  keeps changing because of updates.  
*Assignment Project Exam Help  
https://powcoder.com  
Add WeChat powcoder*
- This definitely used to help; with modern hardware it may not (Bahle et al. 2002) unless you're memory-based
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

Introduction to  
Assignment Project Exam Help  
**Information Retrieval**  
<https://powcoder.com>

Add WeChat powcoder  
Phrase queries and positional indexes

# Phrase queries

---

- Want to be able to answer queries such as “*stanford university*” – as a phrase
- Thus the sentence “*I went to university at Stanford*” is not a match <https://powcoder.com>
  - The concept of phrase queries has proven easily understood by users, one of the few “advanced search” ideas that works
  - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only  $\langle \text{term} : \text{docs} \rangle$  entries

# Solution 1: Biword indexes

---

- Index every consecutive pair of terms in the text as a phrase
- For example, the text “Friends, Romans, Countrymen” would generate the biwords
  - *friends romans*
  - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

# Longer phrase queries

- Longer phrases are processed as we did with wild-cards:
- *stanford university palo alto* can be broken into the Boolean query on biwords:  
<https://powcoder.com>

*stanford university AND university palo AND palo alto*  
Add WeChat powcoder

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

# Issues for biword indexes

---

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them  
<https://powcoder.com>
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

## Solution 2: Positional indexes

---

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

Assignment Project Exam Help

<***term***, number of docs containing ***term***;

doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>

# Positional index example

<*be*: 993427;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, 430, 434;

5: 363, 367, ...

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Which of docs 1,2,4,5  
could contain “*to be*  
or *not to be*”?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

# Processing a phrase query

- Extract inverted index entries for each distinct term:  
*to, be, or, not.*
- Merge the ~~Assignment Project Exam Help~~ all positions with “**to be or not to be**”.  
<https://powcoder.com>
  - *to:*
    - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - *be:*
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Same general method for proximity searches

# Proximity queries

---

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here,  $/k$  means “within  $k$  words of”.
- Clearly, positional indexes can be used for such queries; biword indexes cannot.  
<https://powcoder.com>
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of  $k$ ?
  - This is a little tricky to do correctly and efficiently
  - See Figure 2.12 of IIR (Page 39)
  - There's likely to be a problem on it!

---

```

POSITIONALINTERSECT( $p_1, p_2, k$ )
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  $l \leftarrow \langle \rangle$ 
5      $pp_1 \leftarrow \text{positions}(p_1)$ 
6      $pp_2 \leftarrow \text{positions}(p_2)$ 
7     while  $pp_1 \neq \text{NIL}$ 
8     do while  $pp_2 \neq \text{NIL}$ 
9       do if  $|\text{pos}(pp_1) - \text{pos}(pp_2)| \leq k$ 
10      then ADD( $l, \text{pos}(pp_2)$ )
11      else if  $\text{pos}(pp_2) > \text{pos}(pp_1)$ 
12        then break
13         $pp_2 \leftarrow \text{next}(pp_2)$ 
14        while  $l \neq \langle \rangle$  and  $|l[0] - \text{pos}(pp_1)| > k$ 
15          do DELETE( $l[0]$ )
16          for each  $ps \in l$ 
17            do ADD(answer,  $\langle \text{docID}(p_1), \text{pos}(pp_1), ps \rangle$ )
18             $pp_1 \leftarrow \text{next}(pp_1)$ 
19             $p_1 \leftarrow \text{next}(p_1)$ 
20             $p_2 \leftarrow \text{next}(p_2)$ 
21        else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
22          then  $p_1 \leftarrow \text{next}(p_1)$ 
23          else  $p_2 \leftarrow \text{next}(p_2)$ 
24 return answer

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

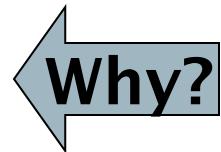
# Positional index size

---

- You can compress position values/offsets: we'll talk about that in lecture 5
- Nevertheless, a positional index expands postings storage substantially  
<https://powcoder.com>
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries ... whether used explicitly or implicitly in a ranking retrieval system.

# Positional index size

- Need an entry for **each occurrence**, not just once per document
- Index size depends on average document size
  - Average web page has ~1000 terms
  - SEC filings, books, even some epic poems ... easily 100,000 terms
- Consider a term with frequency 0.1%



Document size	Postings	Positional postings
1000	1	1
100,000	1	100

# Rules of thumb

---

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text  
<https://powcoder.com>
- Caveat: all of this holds for “English-like” languages  
[Add WeChat powcoder](#)

# Combination schemes

---

- These two approaches can be profitably combined
  - For particular phrases ("Michael Jackson", "Britney Spears") it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like "The Who"
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in  $\frac{1}{4}$  of the time of using just a positional index
  - It required 26% more space than having a positional index alone

# Resources for today's lecture

---

- *Introduction to Information Retrieval*, chapter 1
- Shakespeare:
  - [Assignment Project Exam Help](http://www.phyzone.com/shakespeare/)
  - Try the neat browser by keyword sequence feature!  
<https://powcoder.com>
- [Add WeChat powcoder](#)
- *Managing Gigabytes*, chapter 3.2
- *Modern Information Retrieval*, chapter 8.2

# Resources for today's lecture

---

- Skip Lists theory: Pugh (1990)
  - Multilevel skip lists give same  $O(\log n)$  efficiency as trees
- H.E. Williams, J. Zobel, and D. Bahle, 2004, “Fast Phrase Querying with Combined Indexes”, ACM Transactions on Information Systems  
**Assignment Project Exam Help**  
<https://powcoder.com>  
<http://www.seg.rmit.edu.au/research/research.php?author=4>
- D. Bahle, H. Williams and J. Zobel, Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.