

Multimedia Information Retrieval and Technology

L2. Query Processing

By : Laura Liu

Room: EE314

Tel. no. 7756



Xi'an Jiaotong-Liverpool University
西交利物浦大学

Exercise I (a)

- (a) What is the entropy (η) of the image below, where numbers (0, 20, 50, 99) denote the gray-level intensities?

99	99	99	99	99	99	99	99
20	20	20	20	20	20	20	20
0	0	0	0	0	0	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	50	50	50	50	0	0
0	0	0	0	0	0	0	0



Solution

$$P_{20} = P_{99} = 1/8, P_{50} = 1/4, P_0 = 1/2.$$

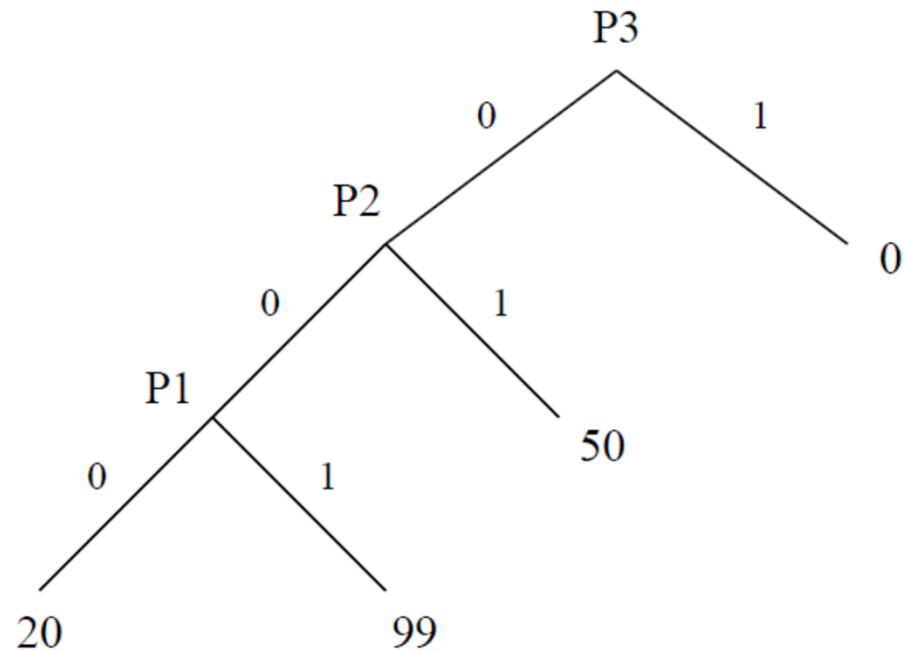
$$\eta = 2 \times \frac{1}{8} \log_2 8 + \frac{1}{4} \log_2 4 + \frac{1}{2} \log_2 2 = \frac{3}{4} + \frac{1}{2} + \frac{1}{2} = 1.75$$



Exercise I (b,c)

- (b) Show step by step how to construct the Huffman tree to encode the above four intensity values in this image. Show the resulting code for each intensity value.
- (c) What is the average number of bits needed for each pixel, using your Huffman code? How does it compare to η ?

(b) Only the final tree is shown below. Resulting code: 0: “1”, 50: “01”, 20: “000”, 99: “001”



(c) Average number of bits = $0.5 \times 1 + 0.25 \times 2 + 2 \times 0.125 \times 3 = 1.75$.

This happens to be identical to η — it only happens when all probabilities are 2^{-k} where k is an integer. Otherwise, this number will be larger than η .



Exercise 2

Assume that Adaptive Huffman Coding is used to code an information source S with a vocabulary of four letters (a, b, c, d). Before any transmission, the initial coding is $a = 00$, $b = 01$, $c = 10$, $d = 11$. A special symbol NEW will be sent before any letter if it is to be sent the first time.

Figure 7.11 is the Adaptive Huffman tree after sending letters *aabb*. After that, the additional bitstream received by the decoder for the next few letters is 01010010101.

- What are the additional letters received?
- Draw the adaptive Huffman trees after each of the additional letters is received.

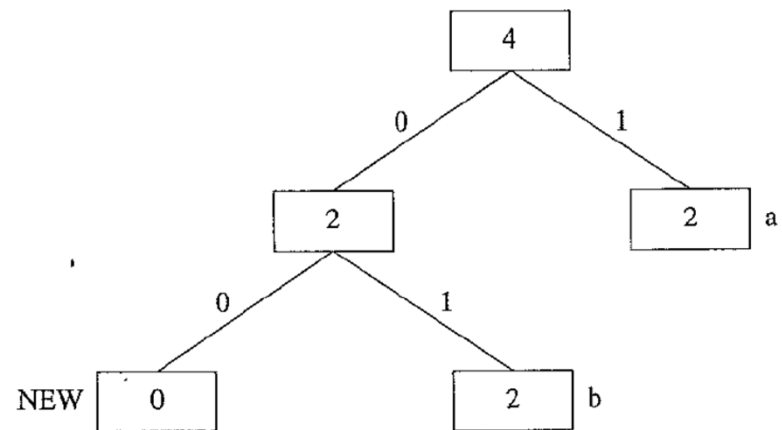
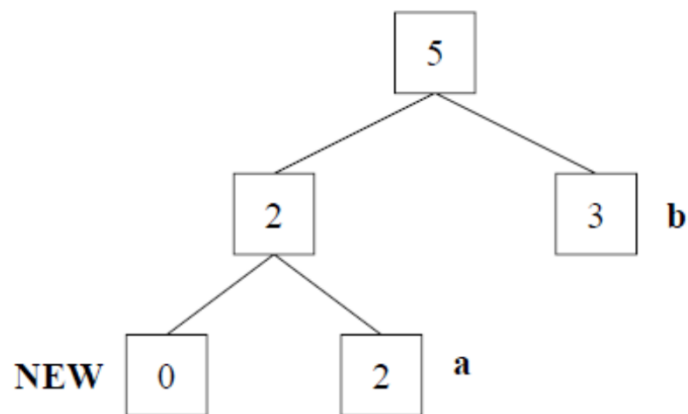


FIGURE 7.11: Adaptive Huffman tree.

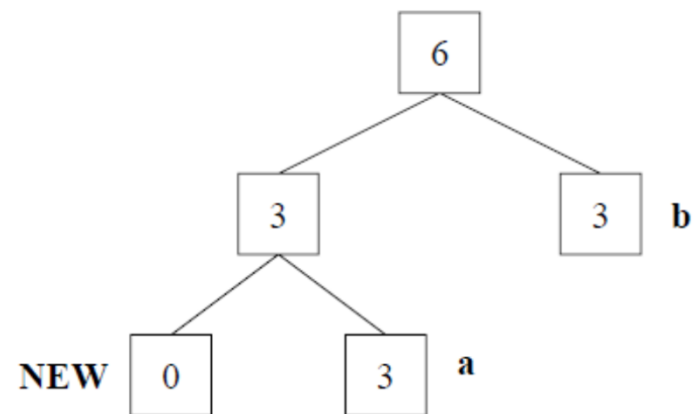
Solution

- (i) The additional letters received are “b (01) a (01) c (00 10) c (101)”.
- (ii) The trees are as below.

After another "b"



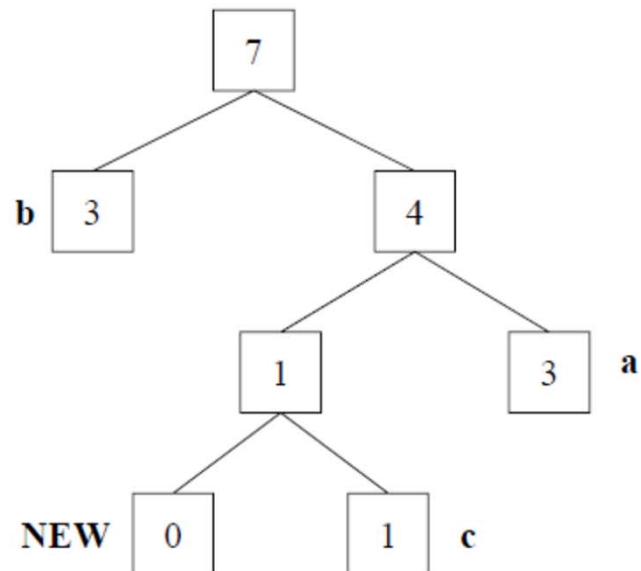
After another "a"



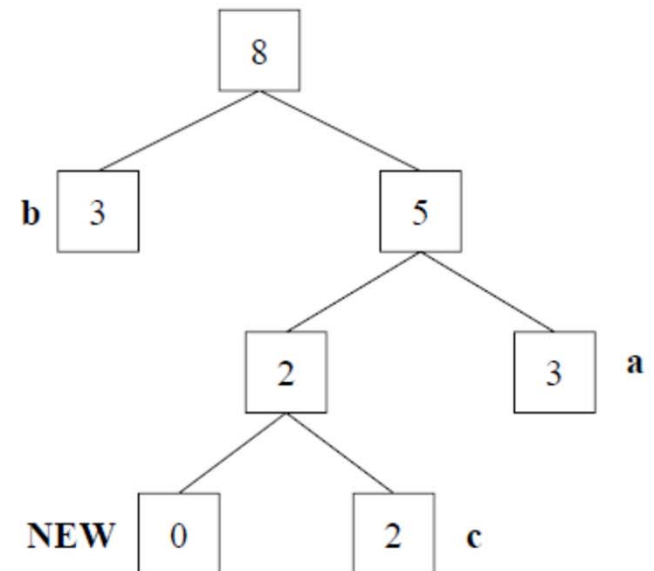
Xi'an Jiaotong-Liverpool University

西交利物浦大學

After "c"



After another "c"



Processing Boolean queries

- I. Query processing with an inverted index
- II. Query optimization
- III. The Extended Boolean Models
- IV. Faster posting list intersection



The index we just built

How do we process a query?

AND operation

The *intersection* operation is the crucial one: we need to efficiently intersect postings lists so as to be able to quickly find documents that contain both terms.

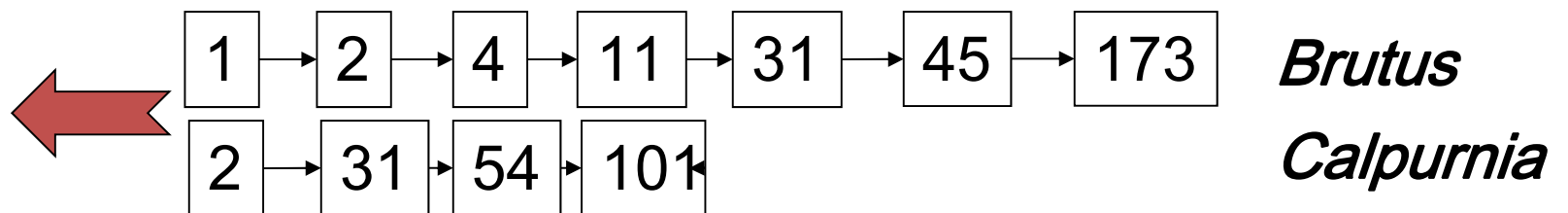


Query processing: AND

Consider processing the query:

Brutus AND Calpurnia

1. Locate ***Brutus*** in the Dictionary; Retrieve its postings.
2. Locate ***Calpurnia*** in the Dictionary; Retrieve its postings.
3. “Merge” the two postings (intersect the document sets):

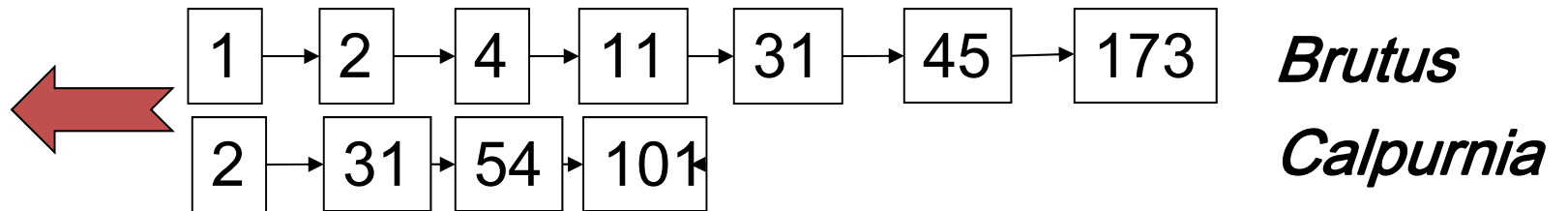


Intersecting two postings lists (a “merge” algorithm)

```
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  $docID(p_1) = docID(p_2)$ 
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  $answer$ 
```

The merge

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



Intersecting two postings lists

BRUTUS \longrightarrow $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA \longrightarrow $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection \implies

Intersecting two postings lists

BRUTUS \longrightarrow 1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 11 \longrightarrow 31 \longrightarrow 45 \longrightarrow 173 \longrightarrow 174

CALPURNIA \longrightarrow 2 \longrightarrow 31 \longrightarrow 54 \longrightarrow 101

Intersection \implies

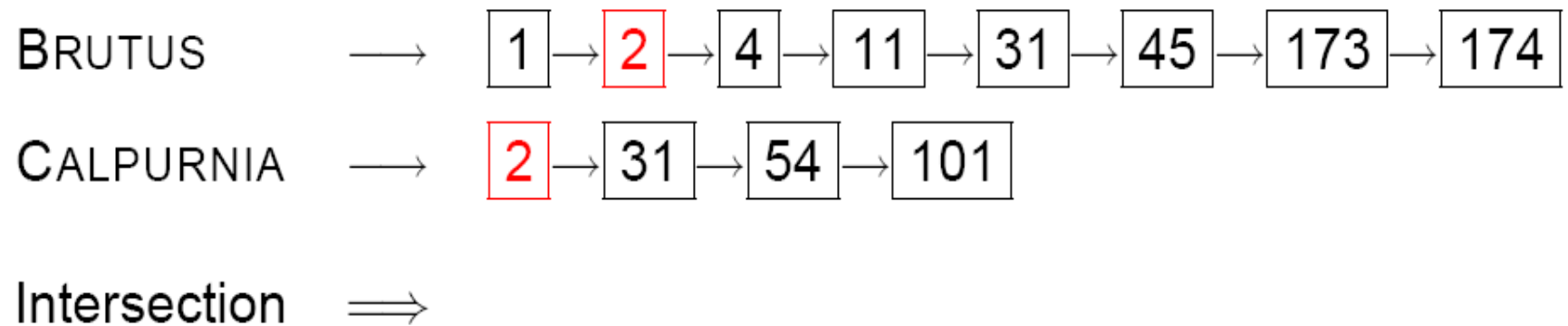
Intersecting two postings lists

BRUTUS \longrightarrow 1 \longrightarrow 2 \longrightarrow 4 \longrightarrow 11 \longrightarrow 31 \longrightarrow 45 \longrightarrow 173 \longrightarrow 174

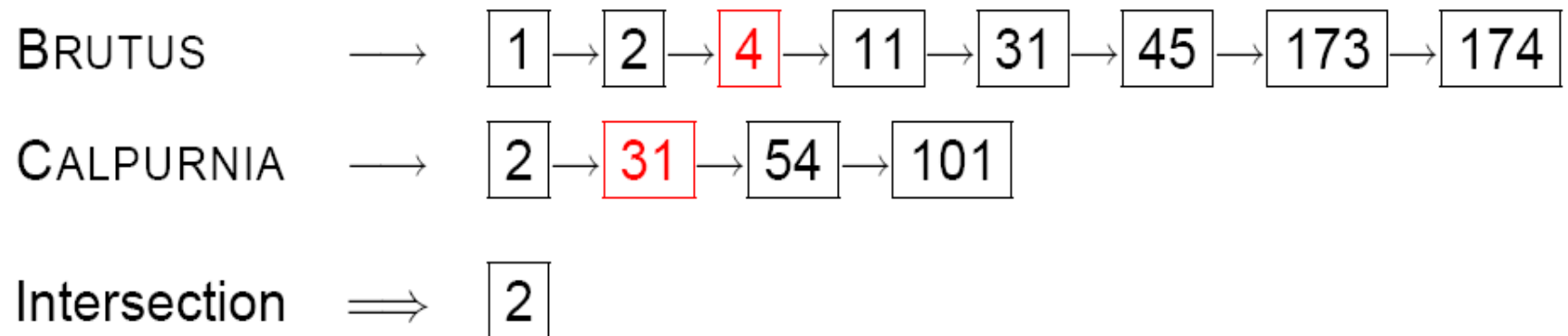
CALPURNIA \longrightarrow 2 \longrightarrow 31 \longrightarrow 54 \longrightarrow 101

Intersection \implies

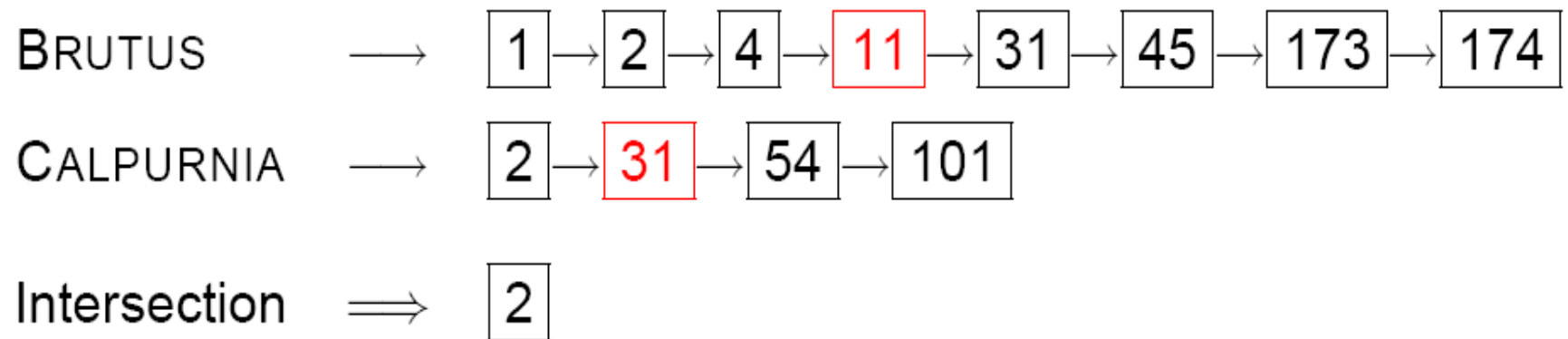
Intersecting two postings lists



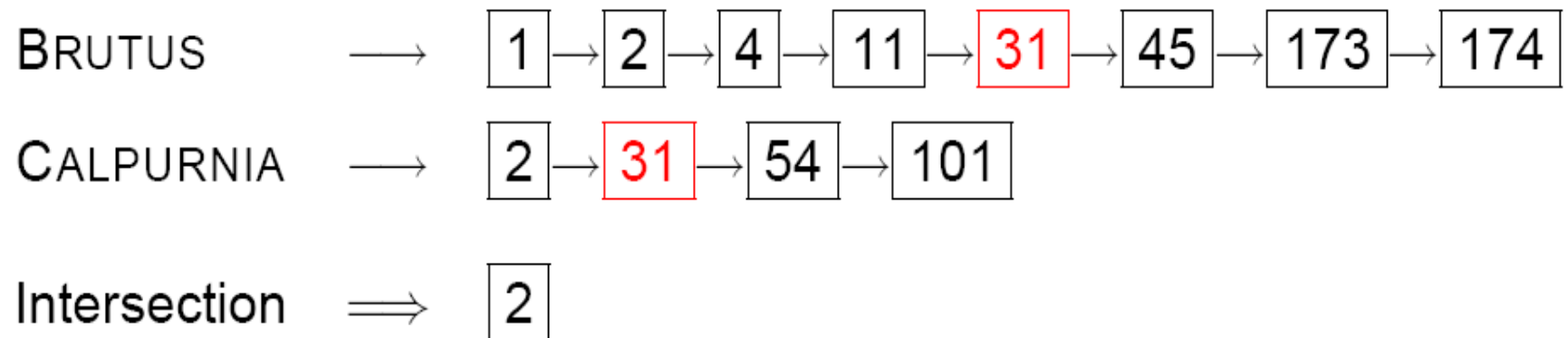
Intersecting two postings lists



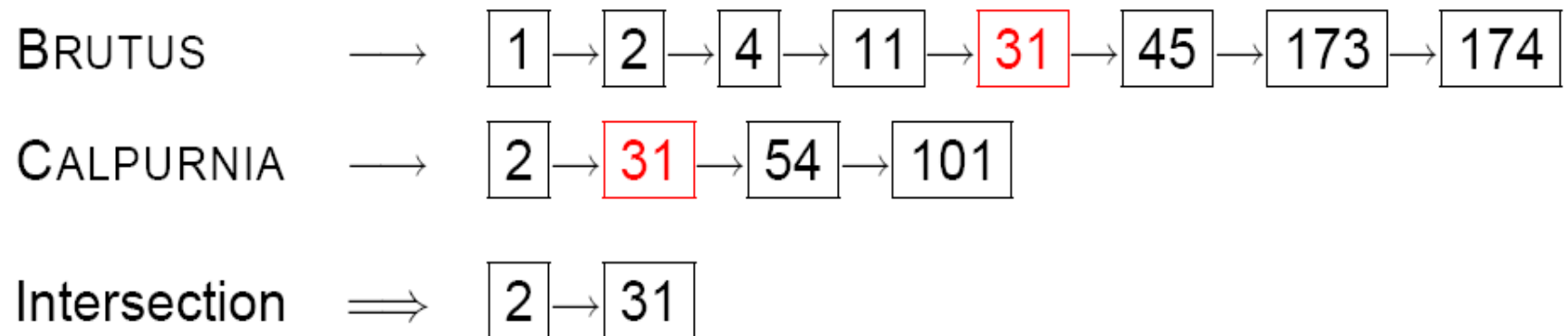
Intersecting two postings lists



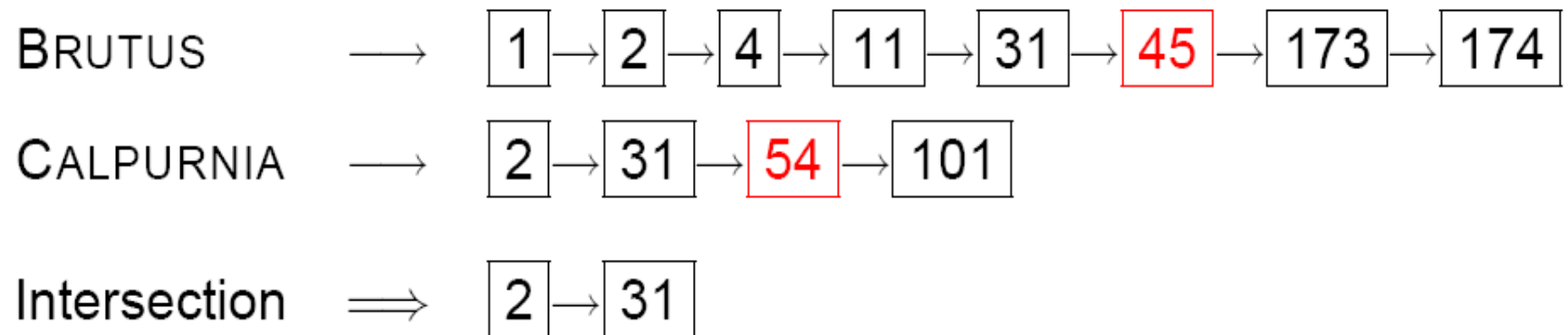
Intersecting two postings lists



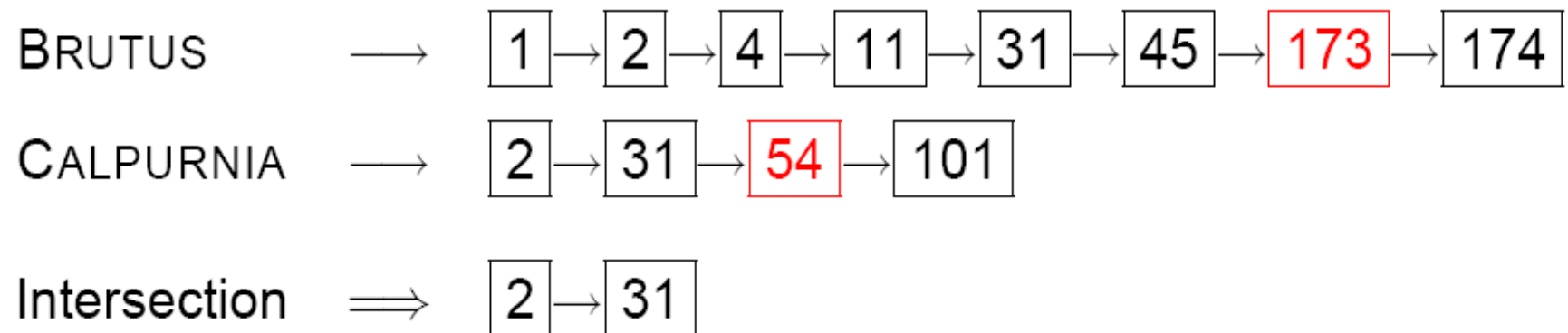
Intersecting two postings lists



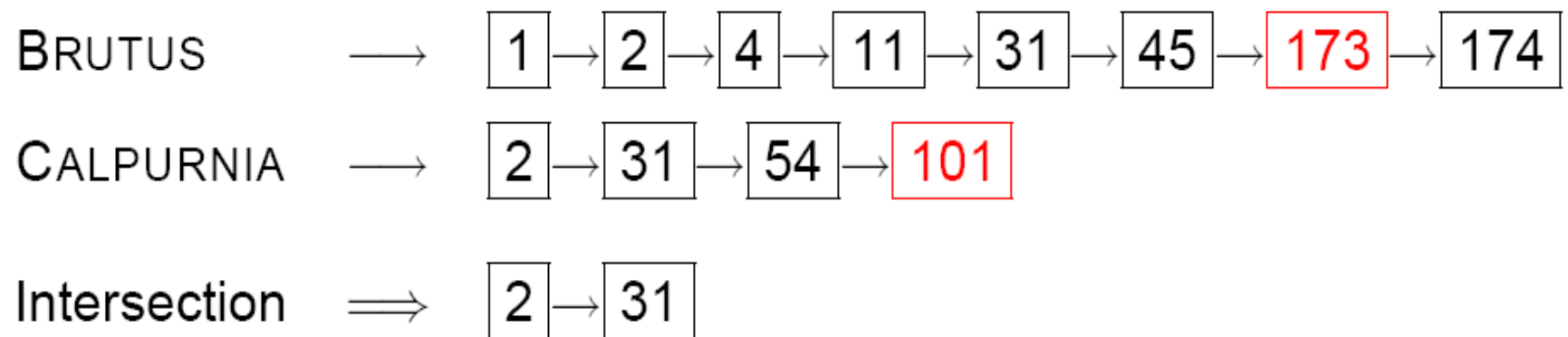
Intersecting two postings lists



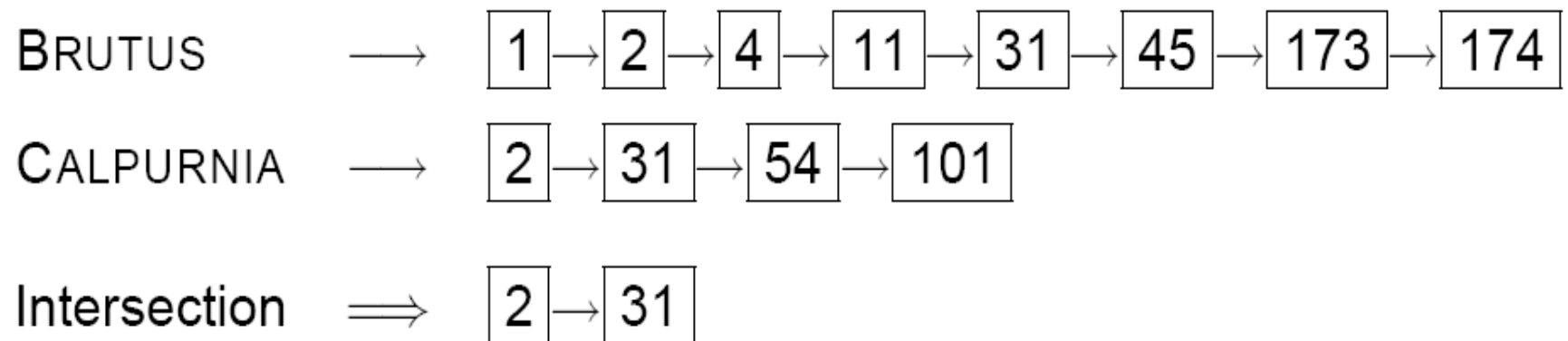
Intersecting two postings lists



Intersecting two postings lists



Intersecting two postings lists



Intersecting two postings lists

BRUTUS \longrightarrow $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA \longrightarrow $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection \implies $\boxed{2} \rightarrow \boxed{31}$

- This is linear in the length of the postings lists.
- This only works if postings lists are sorted.

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

Crucial: postings sorted by docID.

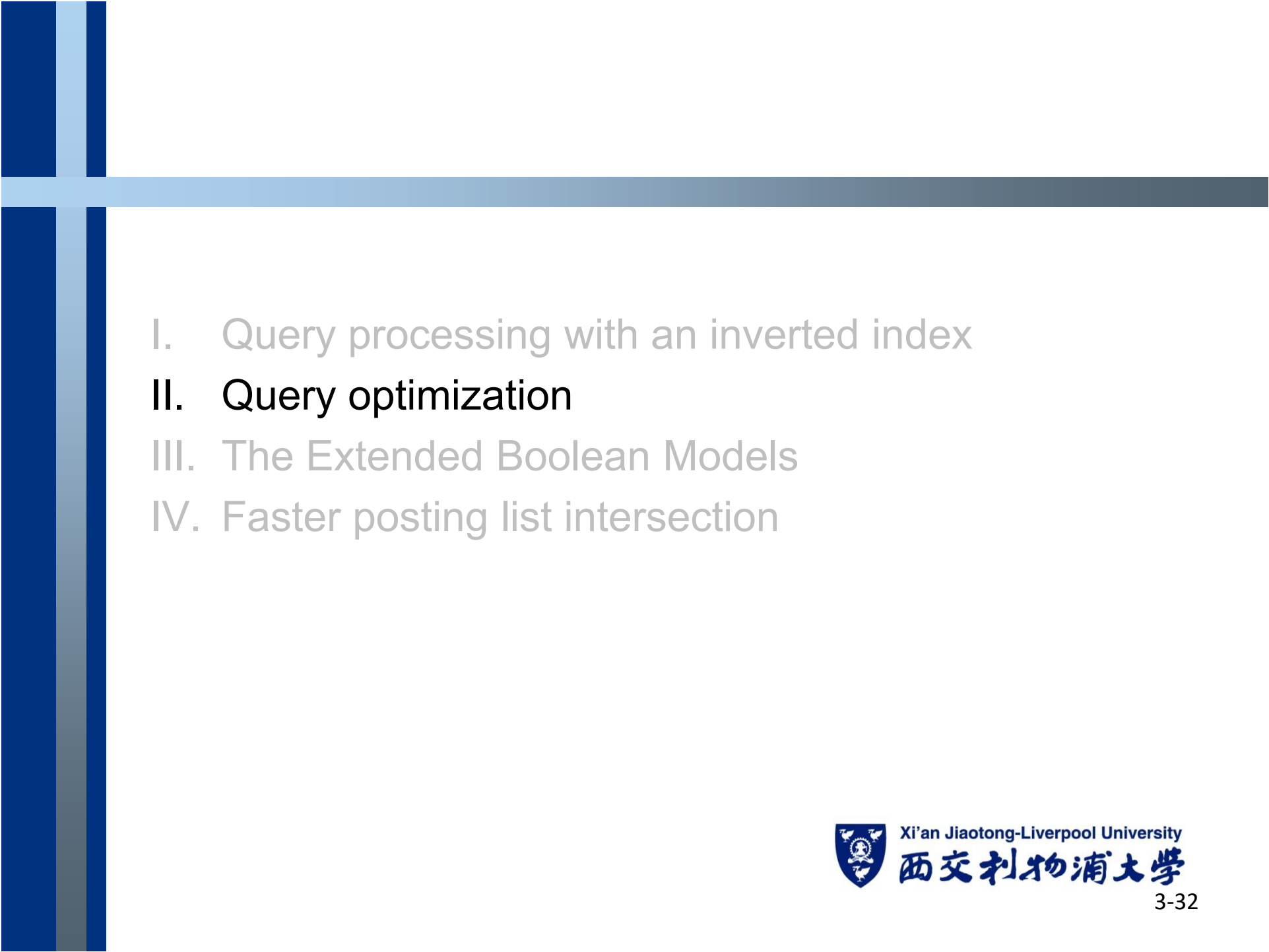
Exercise:

Write out a postings merge algorithm for an x or y query



Xi'an Jiaotong-Liverpool University

西交利物浦大學

- 
- I. Query processing with an inverted index
 - II. Query optimization**
 - III. The Extended Boolean Models
 - IV. Faster posting list intersection



Xi'an Jiaotong-Liverpool University

西交利物浦大學

Query Optimization

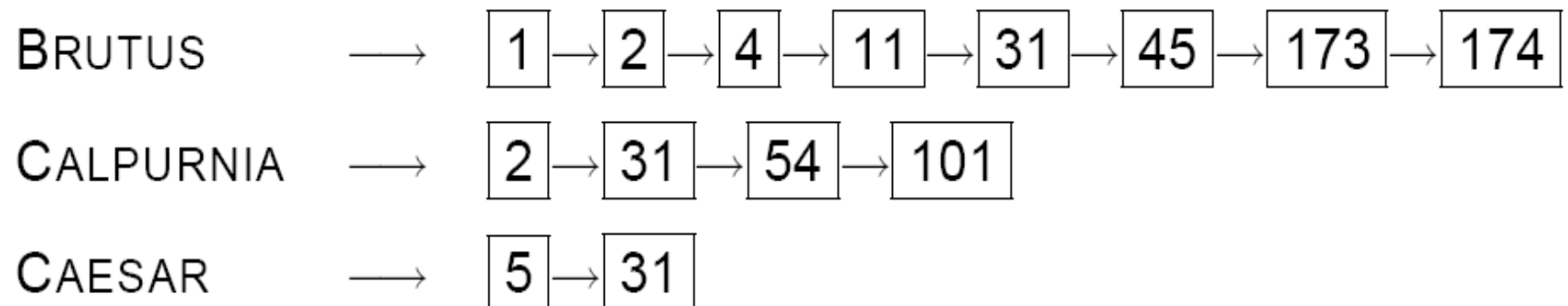
The process of selecting **how to organize** the work of answering a query so that **the least amount of work** needs to be done by the system.

Query Optimization

- What is the best order for query processing?
- Consider a query that is an AND of n terms, $n > 2$:
Brutus and Calpurnia and Caesar

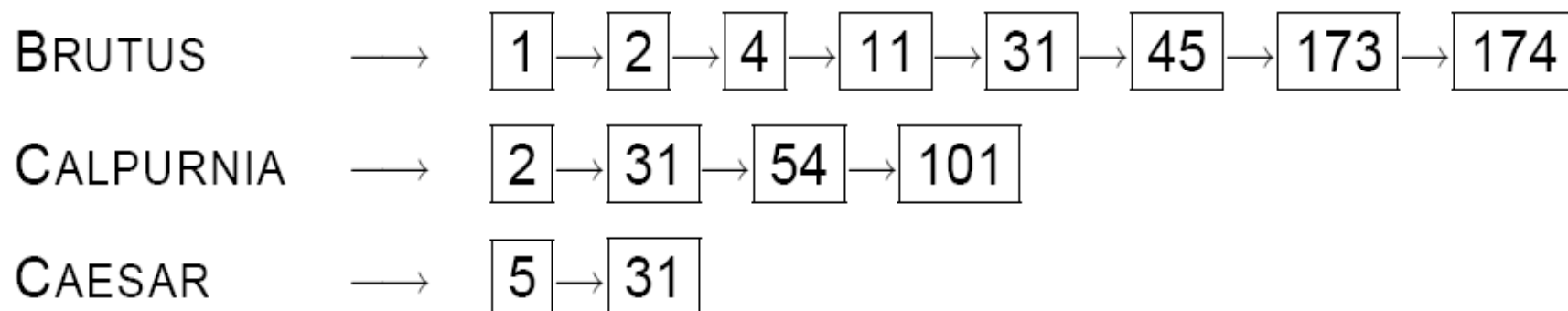
Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR



Query optimization

- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: **Process in order of increasing frequency**
- Start with the shortest postings list, then keep cutting further
- In this example, first CAESAR, then CALPURNIA, then BRUTUS



Optimized intersection algorithm for conjunctive queries

```
INTERSECT( $\langle t_1, \dots, t_n \rangle$ )  
1  terms  $\leftarrow$  SORTBYINCREASINGFREQUENCY( $\langle t_1, \dots, t_n \rangle$ )  
2  result  $\leftarrow$  postings(first(terms))  
3  terms  $\leftarrow$  rest(terms)  
4  while terms  $\neq$  NIL and result  $\neq$  NIL  
5  do result  $\leftarrow$  INTERSECT(result, postings(first(terms)))  
6     terms  $\leftarrow$  rest(terms)  
7  return result
```

Query Optimization

This is the first justification for keeping the frequency of terms in the dictionary, it allows us to make this ordering decision based on **in-memory data** before accessing any postings list.



Xi'an Jiaotong-Liverpool University

西交利物浦大學

- I. Query processing with an inverted index
- II. Query optimization
- III. The Extended Boolean Models**
- IV. Faster posting list intersection



Boolean queries: Exact match

- 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
 - Boolean retrieval is precise: document matches condition or not. (No ranking)
 - Perhaps the simplest model to build an IR system on



Boolean model VS ranked retrieval

Boolean retrieval model:

- Primary commercial retrieval tool for 3 decades.
- Many search systems still in use are Boolean:
 - Email, library catalog, Mac OS X Spotlight

In ranked retrieval models, users largely use **free text queries**, that is, just typing one or more words rather than using a precise language with operators for building up **query expressions**, and the system decides which documents best satisfy the query.



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

Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)

Tens of terabytes of data; ~700,000 users

Majority of users still use boolean queries



Example: WestLaw

Operators:

- & is AND
- /s, /p, /k ask for matches in the same sentence, same paragraph or within k words respectively.
- The exclamation mark (!) gives a trailing wildcard query.
 - disab! matches all words starting with disab.



Example: WestLaw

Information need: What is the statute of limitations in cases involving the federal tort claims act?

Query: LIMIT! /3 STATUTE ACTION /S
FEDERAL /2 TORT /3 CLAIM

/3 = within 3 words,

/S = in same sentence



Example: WestLaw

- 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

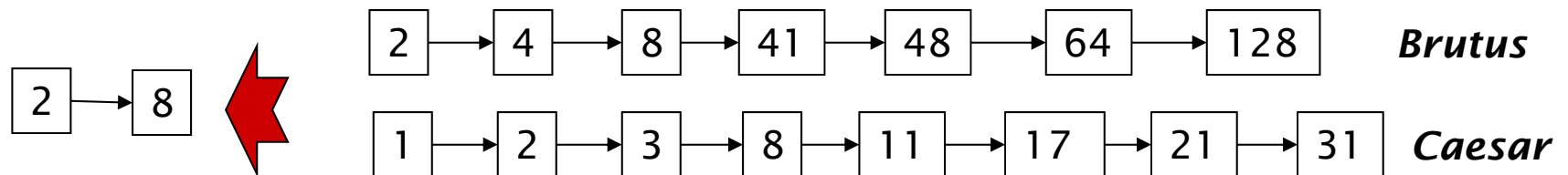


- I. Query processing with an inverted index
- II. Query optimization
- III. The Extended Boolean Models
- IV. Faster posting list intersection



Recall basic merge

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



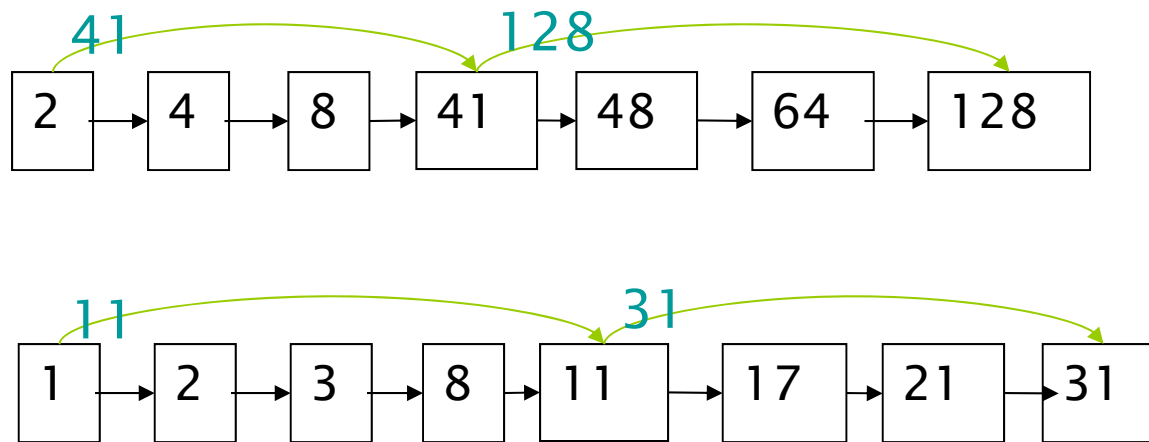
If the list lengths are m and n , the merge takes $O(m+n)$ operations.

Can we do better?

Yes (if the index isn't changing too fast).



Augment postings with skip pointers (at indexing time)



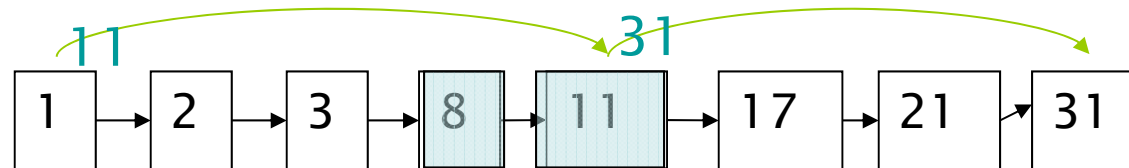
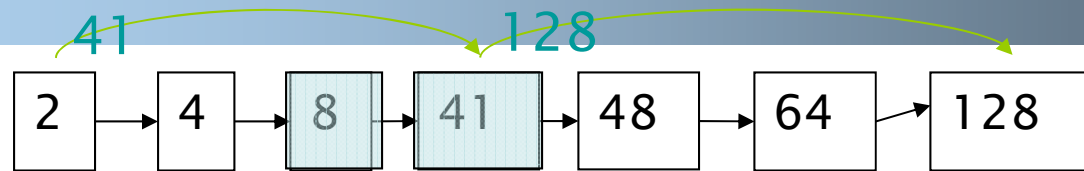
Why?

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

Where do we place skip pointers? How to do efficient merging using skip pointers.



Query processing with skip pointers



1. We process **8** on each list
2. We match it and advance.
3. We then have **41** and **11**, **11** is smaller. Rather than advancing the lower pointer, we first check the skip pointer, and note that **31** is also smaller than 41.
4. we can skip ahead past the intervening postings and advance the lower pointer to **31**.



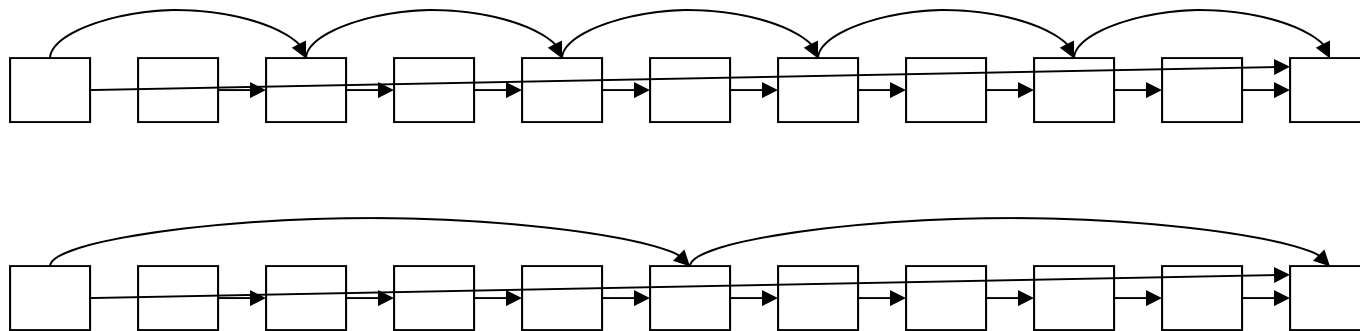
Query processing with skip pointers

Skip pointers will only be available for the original postings lists. (The skip pointer is put at index construction time when we do sorting.)

The presence of skip pointers only helps for AND queries, not for OR queries.



Where do we place skips?

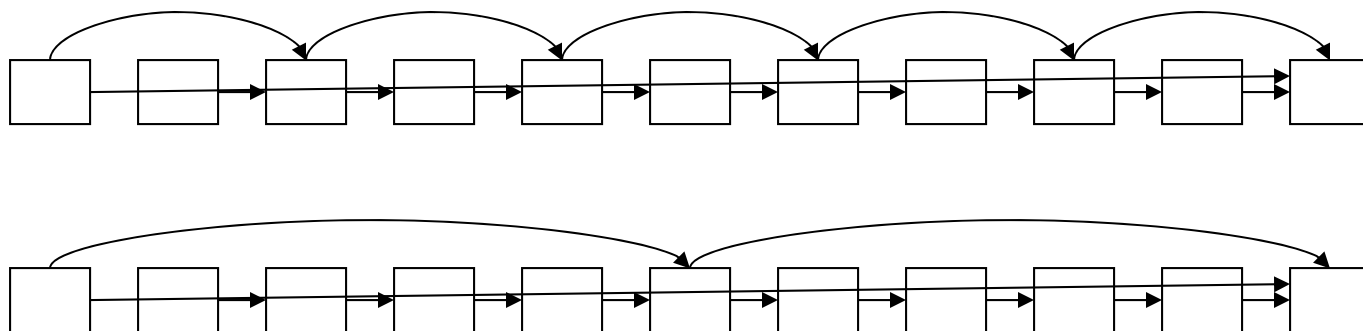


Where do we place skips?

Tradeoff:

More skips \rightarrow shorter skip spans \Rightarrow more likely to skip. But lots of comparisons to skip pointers.

Fewer skips \rightarrow few pointer comparison, but then long skip spans \Rightarrow few successful skips.



Placing skips

Simple heuristic: for postings of length L , use \sqrt{L} evenly-spaced skip pointers

This ignores the distribution of query terms.

Easy if the index is relatively static; harder if L keeps changing because of updates.

