INFORMATION RETRIEVAL AND SEMANTIC WEB 16B1NCI648

Lecture 5

CONTENTS TO BE COVERED

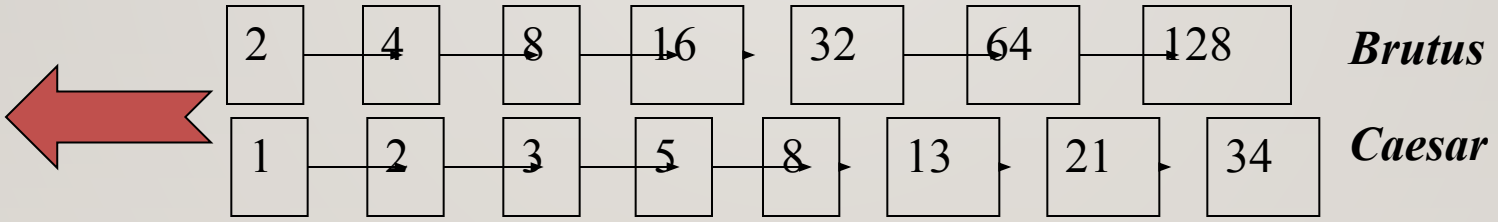
Query processing on Inverted Index

- Query processing
  - Boolean queries
    - Merge algorithm
  - Query optimization
  - Phrase queries
    - Bi-word Indexes
    - Positional Indexes
  - Proximities queries

Query Processing

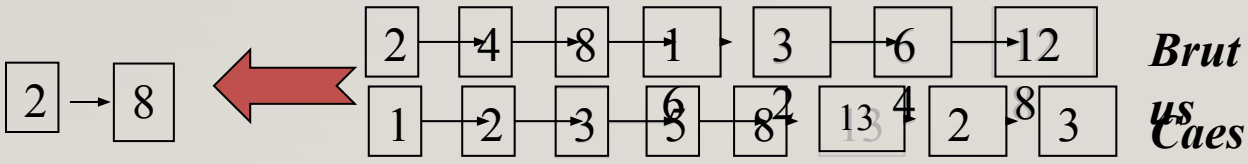
BOOLEAN QUERY PROCESSING: **AND**

- Consider processing the query: ***Brutus*** *AND* ***Caesar***
  - Locate ***Brutus*** in the Dictionary and Retrieve its postings.
  - Locate ***Caesar*** in the Dictionary and Retrieve its postings.
  - **Merge** the two postings:



INTERSECTING TWO POSTINGS LISTS "MERGE" ALGORITHM

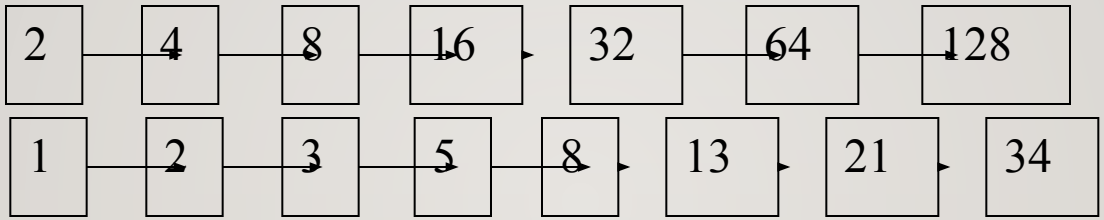
```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```



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

BOOLEAN QUERY PROCESSING: **OR**

- Consider processing the query: ***Brutus*** *OR* ***Caesar***

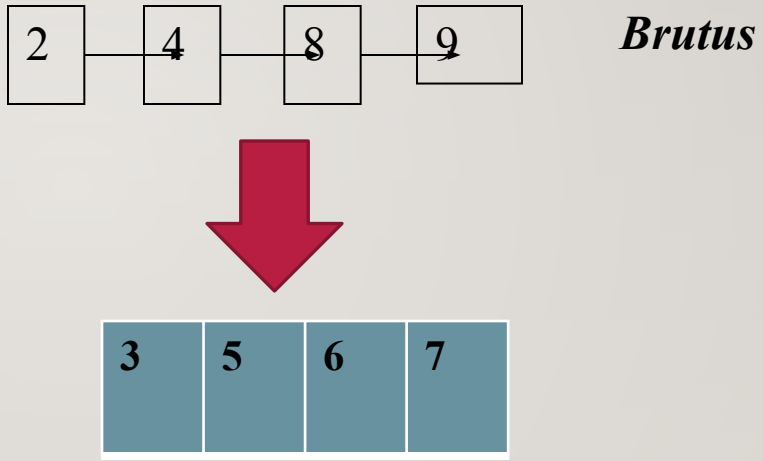


Performs Union Operation

1	2	3	4	5	8	13	16	21	32	34	64	128

BOOLEAN QUERY PROCESSING: **NOT**

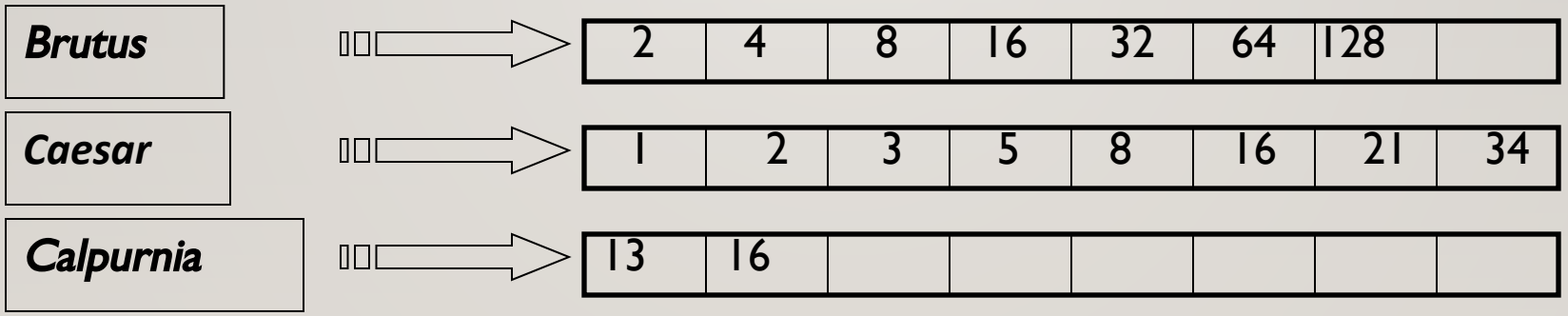
- Consider processing the query: NOT ***Brutus***



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.



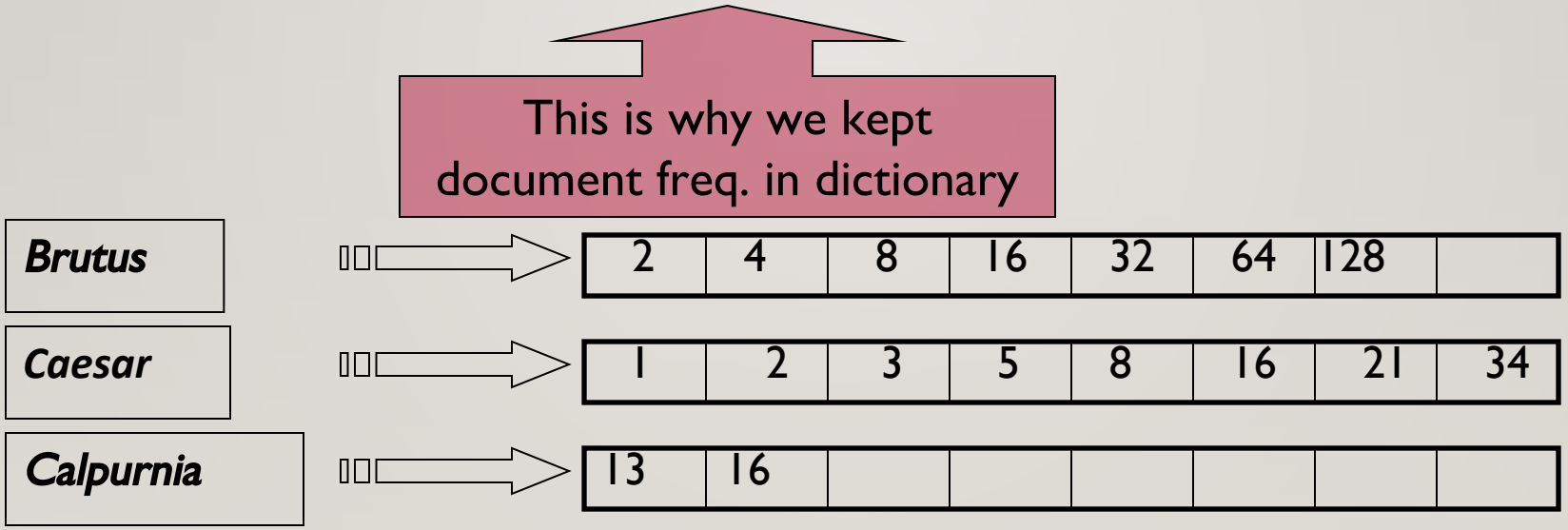
Query: ***Brutus*** *AND* ***Caesar*** *AND* ***Calpurnia***

QUERY OPTIMIZATION

- **The standard heuristic is to process terms in order of increasing document frequency.**
- If we start by intersecting the two smallest posting lists then all intermediate results must be no bigger than the smallest posting list and we are therefore likely to do the least amount of work.

QUERY OPTIMIZATION EXAMPLE

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



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

MORE GENERAL OPTIMIZATION

Consider an example :

*(**madding** OR **crowd**) AND (**ignoble** OR **strife**) AND (**killed** OR **slain**)*

The steps to process the query will be

- 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.

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

PHRASE QUERIES

- Many complex or technical concepts and many organization and product names are multiword compounds or phrases.
- If we wish to answer queries such as "**JIIT University**" – as a phrase . In this case, the sentence "**I went to JIIT which is great university**" is not a match.
- 10% of web queries are phrase queries and many more queries are implicit phrase queries without double quotes

PHRASE QUERIES

Biword Indexes: One approach to handling phrases is to consider every pair of consecutive terms in a document as a phrase.

For eg, ***Friends, Romans, Countrymen*** *would generate the biwords as:*

friends romans

romans countrymen

- In this model, we treat each of these biwords as **a vocabulary term**.
- The concept of a **biword index** can be extended to longer sequences of words, and if the index includes variable length word sequences, it is generally referred to as a ***phrase index***.

LONGER PHRASES QUERIES

- Longer phrases are processed as Boolean biword queries:
- For eg: **JIIT University Noida UP** can be written in the form in biwords as

  JIIT University AND University Noida AND Noida UP

- Without examining the documents, we cannot tell that the documents matching the boolean query actually contain above 4 word phrase.

BIWORD INDEXES ISSUES

- In this method occasionally we get false positive
- Index blowup due to bigger dictionary – Infeasible for more than biwords, big even for them
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy.

POSITIONAL INDEX

- Another solution is **Positional Index**, which popularly used.
- In this postings(Docid) store, for each term the position(s) in which tokens of it appear:
- (**term,** total documents that contain this **term;**

  Doc 1: pos1,pos2….;

  Doc2: pos1, pos3, pos10 …..;

)

POSITIONAL INDEX

- e.g.

to, 993427:

(1, 6: (7, 18, 33, 72, 86, 231);

2, 5: (1, 17, 74, 222, 255);

4, 5: (8, 16, 190, 429, 433);

5, 2: (363, 367);

7, 3: (13, 23, 191); ....)

be, 178239:

(1, 2: (17, 25);

4, 5: (17, 191, 291, 430, 434);

5, 3: (14, 19, 101); . . . ..)

POSITIONAL INDEX

- To process a phrase query, we still need to access the inverted index entries for each distinct term.
- In the merge operation, the same general technique is used as before, but rather than simply checking that both terms are in a document, we also need to check that their positions of appearance in the document are compatible with the phrase query being evaluated.

Example: Satisfying phrase queries

- Suppose the phrase query is **"to be or not to be".**
- Here we will examine intersecting the postings lists for '**to**' and '**be**'.
- We first look for documents that contain both terms.
- Then, we look for places in the lists where there is an occurrence of *'be'* with a token index one higher than a position of *'to',* .

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); …)

PROXIMITY QUERIES (EXTENDED BOOLEAN MODEL)

- The same general method is applied for within word proximity searches:
- employment /3 place
- Here, / means ``within k words of (on either side)".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of k?

PROXIMITY OPERATOR

- A proximity operator is a way of specifying that two terms in a query must occur close to each other in a document where closeness may be measured by limiting the allowed number of intervening words or reference to structural unit such as sentence or paragraph

PROXIMITY QUERIES

- Create term indexes just like inverted index.
- It stores terms positions in the documents instead of counts.
- Positions are embedded in the inverted list.

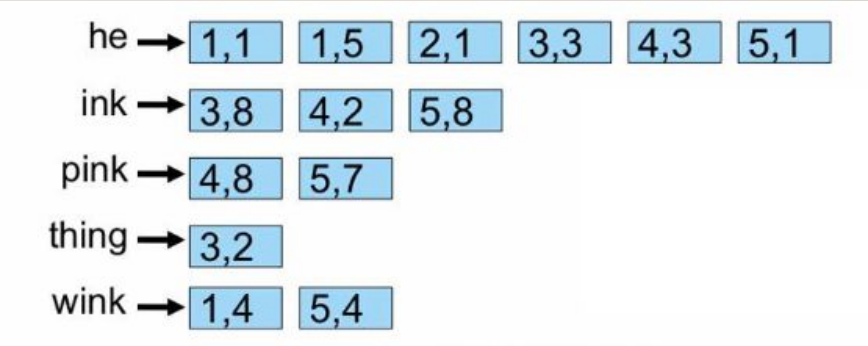
D1: He likes to wink, he likes to drink.

D2: He likes to drink, and drink, and drink.

D3: The thing he likes to drink is ink.

D4: The ink he likes to drink is pink.

D5: He likes to wink and drink pink ink.



Source: Victor Levranko, Indexing 4: phrases and proximity, url: https://www.youtube.com/watch?v=A6y7RAXxBCI

QUERY HANDLING USING PROXIMITY INDEXES

- Lets say we want to find "**Great JIIT**"
- Create postings of the terms JIIT and Great
- Compare the document ids of both terms
- If found a match- compute pos(JIIT)-pos(Great)=1

Here JIIT(5,7) and Great(5,6) is the right match

POSITIONING INDEXES

- A positional index expands postings storage substantially
- Nevertheless, a positional index is now standard used because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.
- A positional index is 2–4 times large as a non positional index
- Positional index size 35–50% of volume of original text

COMBINATION SCHEMES

- Biword index and Positional index ideas can be combined.
- Use biword index or common phrases (such as Taj Mahal).
  - Avoids merging postings lists.
- Use positional index for other phrases (such as JIIT Noida).

Consider the following fragment of a positional index with the format:

word: document: ⟨position, position, . . .⟩; document: ⟩ position, . . .⟩
. . .

Gates: 1: ⟨3⟩; 2: ⟨6⟩; 3: ⟨2,17⟩; 4: ⟨1⟩;
IBM: 4: ⟨3⟩; 7: ⟨14⟩;
Microsoft: 1: ⟨1⟩; 2: ⟨1,21⟩; 3: ⟨3⟩; 5: ⟨16,22,51⟩;

The /*k* operator, word1 /*k* word2 finds occurrences of word1 within *k* words of word2 (on either side), where *k* is a positive integer argument. Thus $k = 1$ demands that word1 be adjacent to word2.

a. Describe the set of documents that satisfy the query Gates /2 Microsoft.

b. Describe each set of values for *k* for which the query Gates /*k* Microsoft returns a different set of documents as the answer.

Q2: Consider the following documents,

Doc1: Data analysis deals with data sets

Doc2: Data mining in various fields

Doc3: Data analysis techniques and algorithms

Doc4: Data sets with temporal attributes

Construct Inverted Index and execute the query and find the resultant document that will be retrieved.

a. data AND (sets OR analysis)
b. data AND NOT (sets OR analysis)

REFERENCES

- IIR Chapter 2: http://nlp.stanford.edu/IRbook/html/htmledition/positional-postingsand-phrase-queries-1.html.
- D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.