Text Technologies for Data Science

INFR11145

Indexing

Instructor:
Youssef Al Hariri

04-Oct-2023

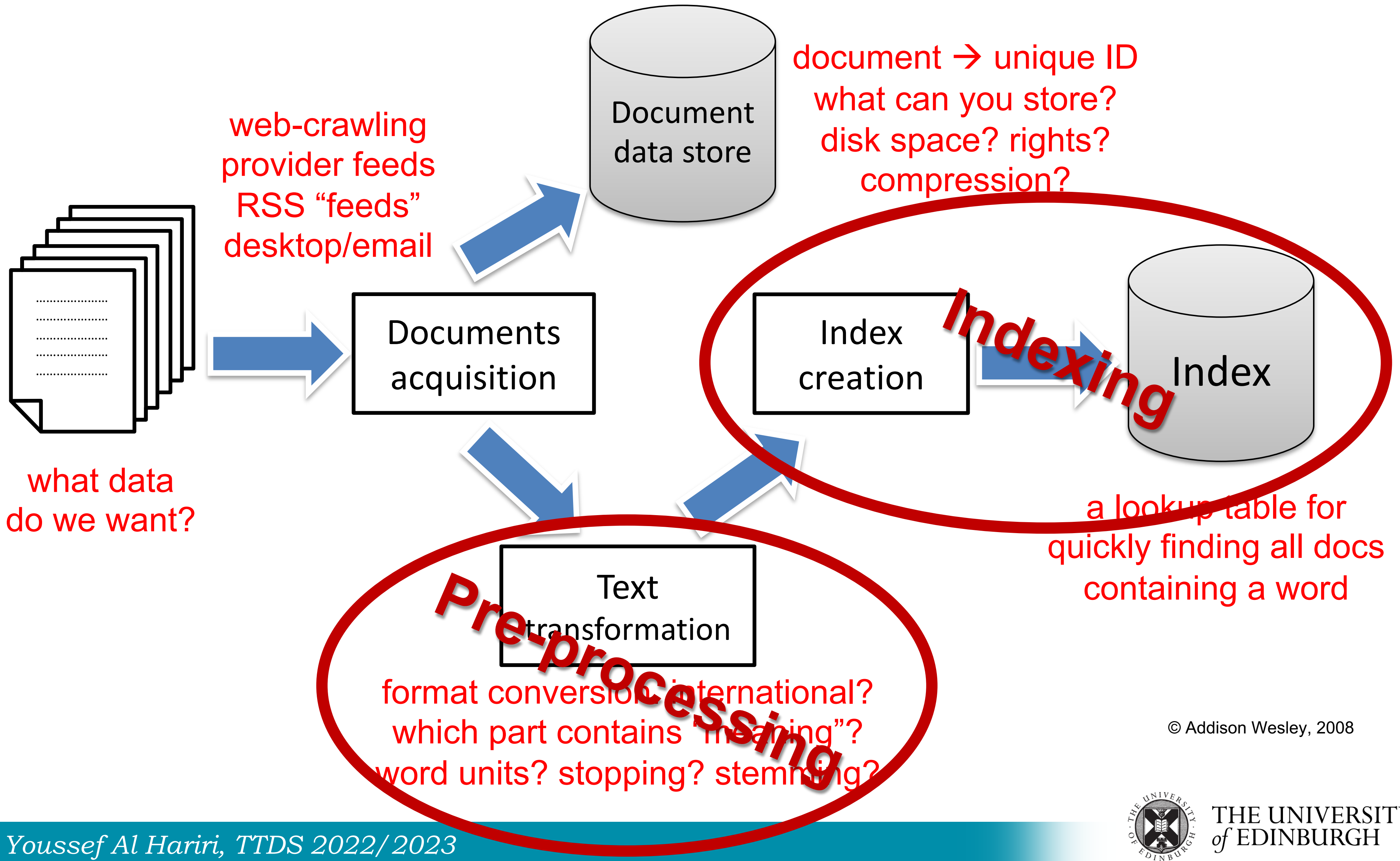
Pre-lecture

- Lectures 1-4 → warmup
  - Now, it is getting more serious!
- Lab 1 → slightly different results
  - Tokenisation, stopping, stemming
- Today: two lectures on Indexing
  - expect some knowledge in binary numbers “001011101”
- Announcement of CW1 (Friday, October 6th)
- Labs → Do it On Time
- Piazza!!!!

Lecture Objectives

- Learn about and implement
- Boolean search
- Inverted index
- Positional index

Indexing Process



THE UNIVERSITY of EDINBURGH

Pre-processing output

This is an example sentence of how the pre-processing is applied to text in information retrieval. It includes: Tokenization, Stop Words Removal, and Stemming

exampl sentenc pre process appli text inform retriev includ token stop word remov stem

- Add processed terms to the index
- What is "index"?

Index

- How to match your term in non-linear time?
- Find/Grep:
  Sequential search for term
- Index:
  Find term locations immediately

Book Index

Index

THE UNIVERSITY of EDINBURGH

Indexing

- Search engines vs PDF find or grep?
  - Infeasible to scan large collection of text for every “search”
  - Find section that has: “UK and Scotland and Money”?!
- Book Index
  - For each word, list of “relevant” pages
  - Find topic in sub-linear time
- IR Index:
  - Data structure for fast finding terms
  - Additional optimisations could be applied

Document Vectors

- Represent documents as vectors
  - Vector → document, cell → term
  - Values: term frequency or binary (0/1)
  - All documents → collection matrix

number of occurrence of
a term in a document

Inverted Index

- Represent terms as vectors
  - Vector → term, cell → document
  - Transpose of the collection matrix
  - Vector: inverted list

he	drink	ink	likes	pink	thing	wink	
2	1	0	2	0	0	1	← **D1:** He likes to wink, he likes to drink
1	3	0	1	0	0	0	← **D2:** He likes to drink, and drink, and drink
1	1	1	1	0	1	0	← **D3:** The thing he likes to drink is ink
1	1	1	1	1	0	0	← **D4:** The ink he likes to drink is pink
1	1	1	1	1	0	1	← **D5:** He likes to wink, and drink pink ink

Boolean Search

- Boolean: exist / not-exist
- Multiword search: logical operators (AND, OR, NOT)
- Example
  - Collection: search Shakespeare's Collected Works
  - Boolean query: Brutus AND Caesar AND NOT Calpurnia

- Build a **Term-Document Incidence Matrix**
  - Which term appears in which document
  - Rows are terms
  - Columns are documents

Collection Matrix

Documents

	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

Terms

1 if ***document*** contains ***term***, 0 otherwise

Query: Brutus AND Caesar AND NOT Calpurnia

Apply on rows: **110100** AND **110111** AND !(**010000**) = **100100**

Bigger collections?

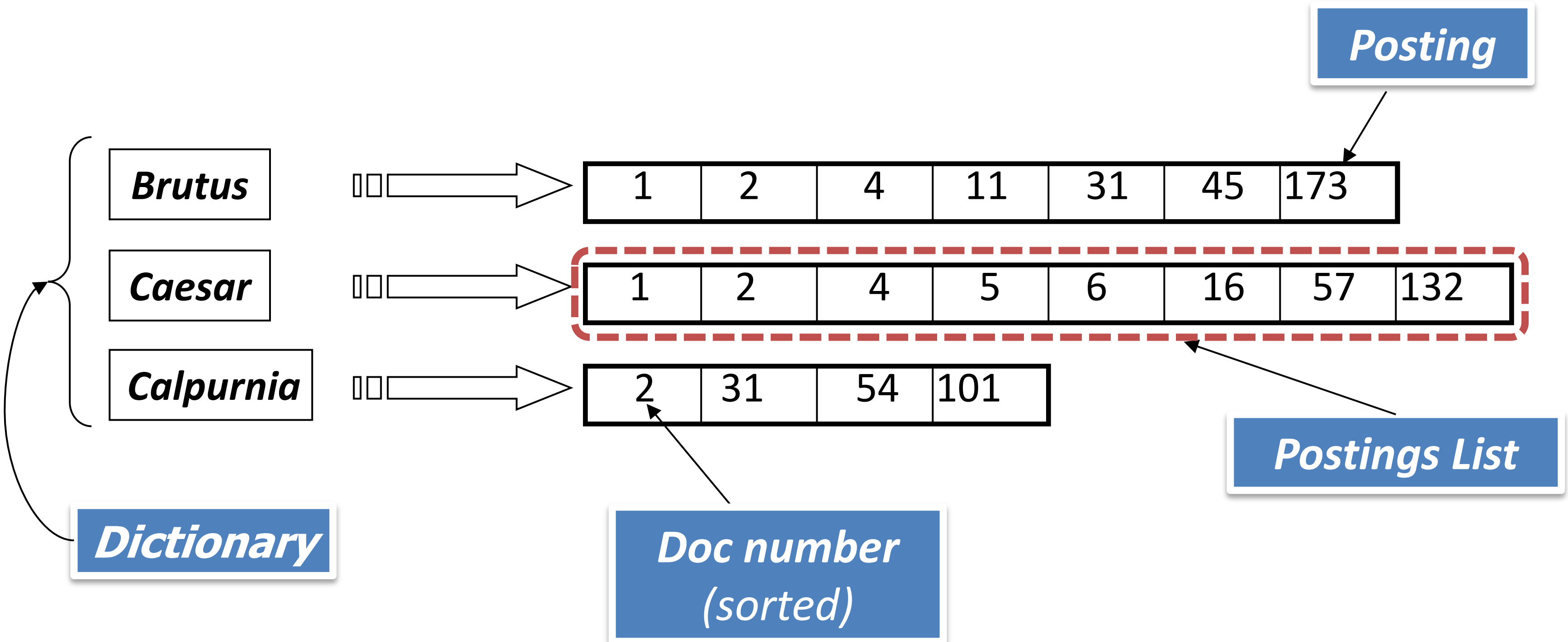
- Consider *N* = 1 million documents, each with about 1000 words.
- n = 1M x 1K = 1B words
  ➔ Heap's law → $v \approx$ 500K
- Matrix size = 500K unique terms x 1M documents = 0.5 trillion 0's and 1's entries!
- If all words appear in many documents
  → max{count(1's)} = N * doc. length = 1B
- Actually, from Zip's law → 250k terms appears once!
- Collection matrix is extremely **sparse**. (*mostly 0's*)

$$\begin{array}{c} \text{term}_x \\ \begin{pmatrix} 1\\0\\0\\0\\1\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\ \vdots \end{pmatrix} \end{array}$$

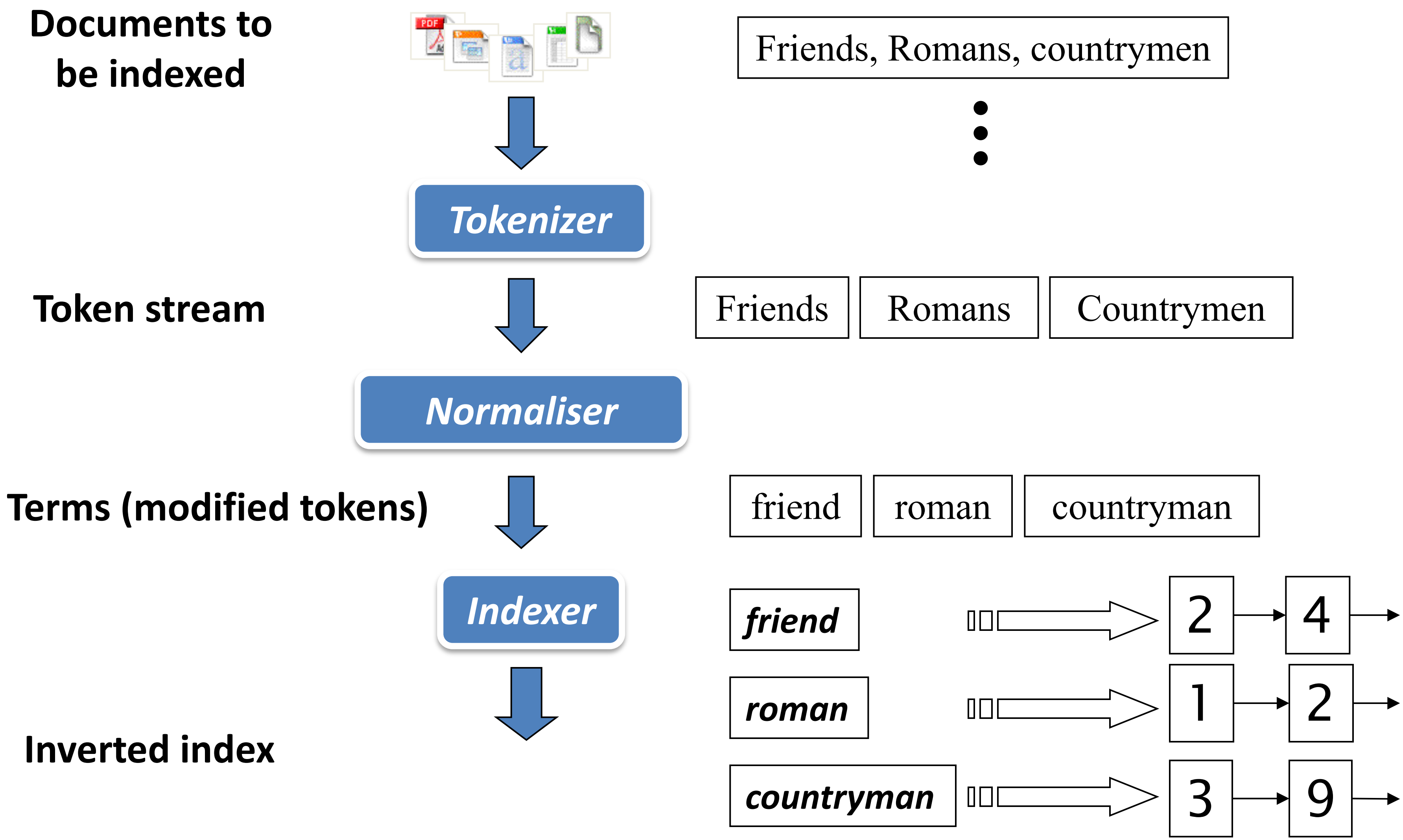
1M

Inverted Index: Sparse representation

- For each term *t*, we must store a list of all documents that contain *t*.
    - Identify each by a **docID**, a document serial number



Inverted Index Construction



Step 1: Term Sequence

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

Sequence of (term, Doc ID) pairs

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

Step 2: Sorting

- **Sort by:**
  - **1) Term**

  then

  - **2) Doc ID**

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

Sorting →

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

Step 3: Posting

1. Multiple term entries in a single document are merged

2. Split into Dictionary and Postings

3. Doc. Frequency (*df*) information is added

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

Inverted Index: matrix → postings

he	drink	ink	likes	pink	thing	wink	
2	1	0	2	0	0	1	← **D1:** He likes to wink, he likes to drink
1	3	0	1	0	0	0	← **D2:** He likes to drink, and drink, and drink
1	1	1	1	0	1	0	← **D3:** The thing he likes to drink is ink
1	1	1	1	1	0	0	← **D4:** The ink he likes to drink is pink
1	1	1	1	1	0	1	← **D5:** He likes to wink, and drink pink ink

he	1	2	3	4	5
drink	1	2	3	4	5
ink	3	4	5		
pink	4	5			
thing	3				
wink	1	5			

Inverted Index: with frequency

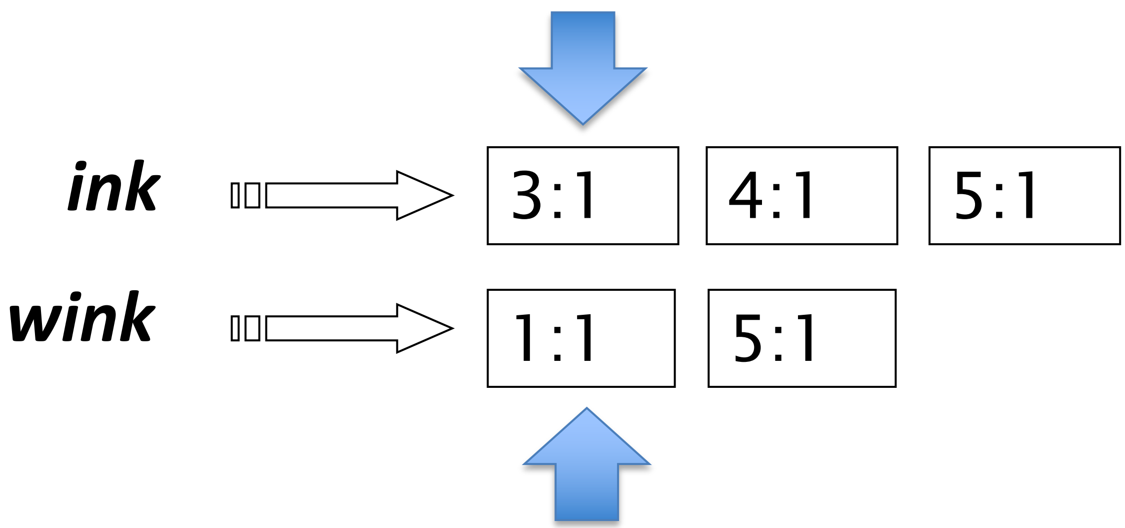
- Boolean: term → DocIDs list
- Frequency: term → touples (DocID,count(term)) lists

he	1:2	2:1	3:1	4:1	5:1
drink	1:1	2:3	3:1	4:1	5:1
ink	3:1	4:1	5:1		
pink	4:1	5:1			
thing	3:1				
wink	1:1	5:1			

appeared in
D2 3 times

Query Processing

- Find documents matching query {ink **AND** wink}
  1. Load inverted lists for each query word
  2. Merge two postings lists → **Linear merge**
- Linear merge → O(*n*)
  n: total number of posts for all query words



Matches

1: f(0,1)

3: f(1,0)

4: f(1,0)

5: f(1,1)

Phrase Search

- Find documents matching query "pink ink"
  1. Find document containing both words
  2. Both words has to be a phrase

- Bi-gram Index:

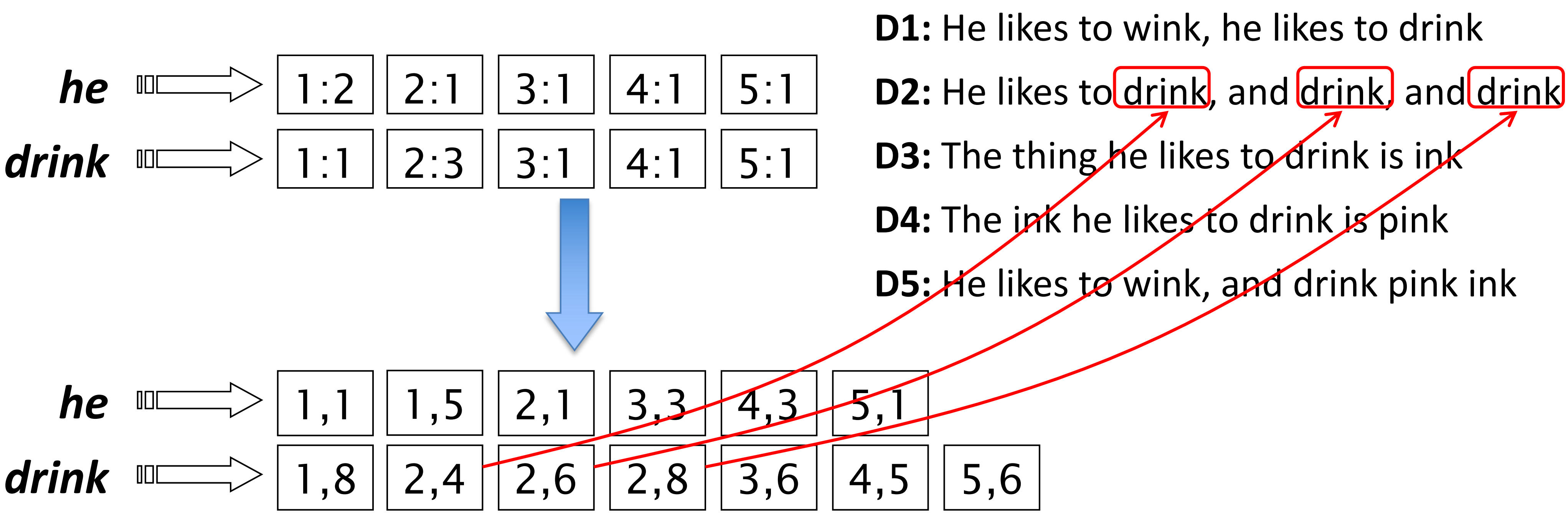
He likes to wink, and drink pink ink $\xrightarrow{\text{Convert to bigrams}}$

He_likes likes_to to_wink wink_and and_drink drink_pink pink_ink

- Bi-gram Index, issues:
  - Fast, but index size will explode!
  - What about trigram phrases?
  - What about proximity? "ink is pink"

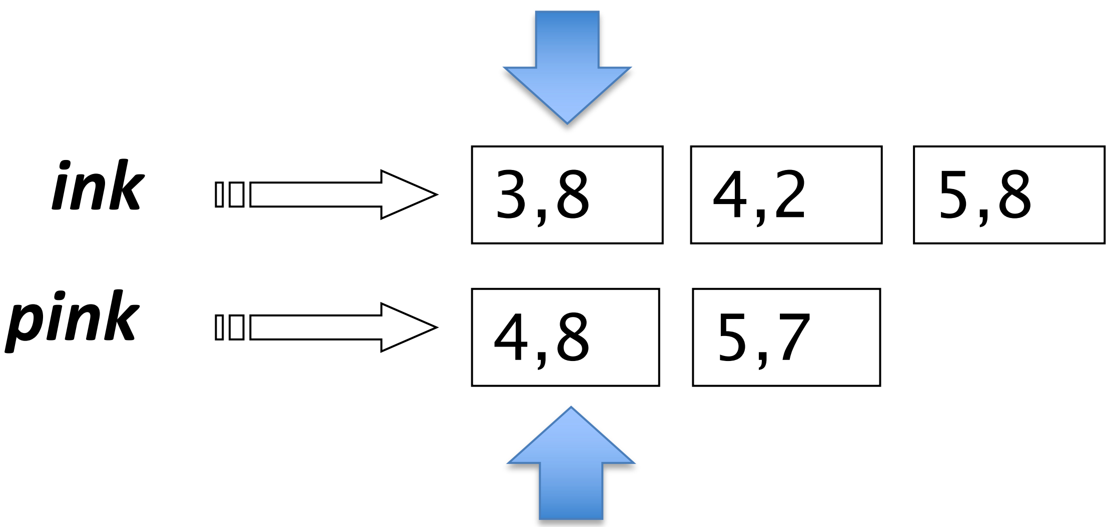
Proximity Index

- Terms positions is embedded to the inv. Index
  - Called proximity/positional index
  - Enables phrase and proximity search
  - Toubles (DocID, term position)



Query Processing: Proximity

- Find documents matching query “pink ink”
  1. Use **Linear merge**
  2. Additional step: check terms positions
- **Proximity search**:
  pos(term1) – pos(term2) < |*w*| → #5(pink,ink)



Matches

3: f(1,0) = **0**

4: f(1,1) = ? =
pos(ink) – pos(pink) == 1?

5: f(1,1) = ? =
pos(ink) – pos(pink) == 1?

Proximity search: data structure

- Possible data structure:

  <term: df;

  DocNo: pos1, pos2, pos3

  DocNo: pos1, pos2, pos3

  ……. >

- Example:

  <***be***: 993427;

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

  2: 3, 149;

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

  5: 363, 367, …>

Practical

Summary

- Document Vector
- Term Vector
- Inverted Index
- Collection Matrix
- Posting
- Proximity Index
- Query Processing → Linear merge

Resources

- Textbook 1: Intro to IR, Chapter 1 & 2.4
- Textbook 2: IR in Practice, Chapter 5
- Lab 2