Indexing

COMP3009J: Information Retrieval

Dr. David Lillis (david.lillis@ucd.ie)

UCD School of Computer Science
Beijing Dublin International College

See Chapter 1 of Manning, Raghavan and Schütze book.
Some slides adapted from P. Nayak and P. Raghavan

Introduction: Indexing

- So that we can search a document collection, we must represent the documents in some type of appropriate data structure.
- We frequently refer to this data structure as an **index**, and the process of creating this data structure is called **indexing**.
- As we saw previously, an incidence matrix is one possibility, but does not scale well for larger document collections.

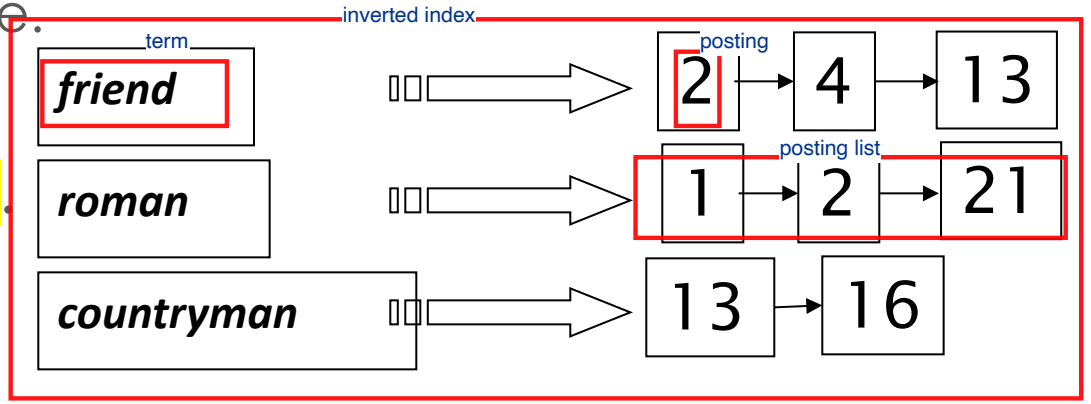
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Inverted Index

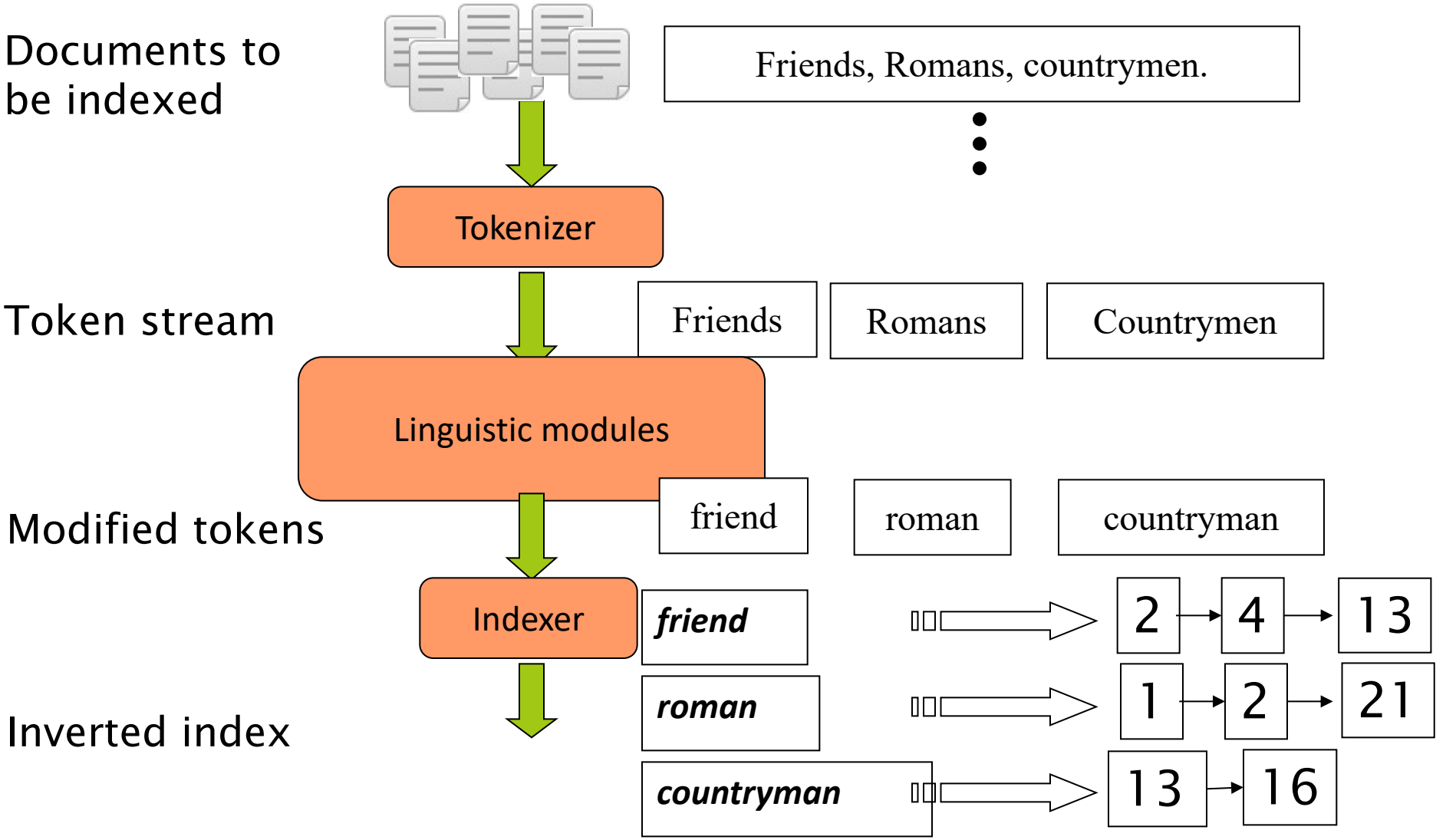
- An **inverted index** is a type of data structure where each term *t* is mapped to a list of documents that contain *t*.
  - We identify each document using a **docID**: a unique identifier for each document.
- We need variable-sized list for each term, called a **postings list**. A linked list would be suitable.
  - Each docID in the list is a **posting**.
- Each list should be **ordered** by the **docID**.
  - We will see the reason for this later.

What is a **term**?

For now, we can think of a term as being a word. Later we will see that they are slightly different.



Construction of an Inverted Index



1. Split the text into **tokens** (generally words)

2. Tokens must be processed further, according to the language the text is in (more on this later).

   Here, we notice that they are converted to lowercase. Why?

3. Then, we create the inverted index.

Indexer step 1: Token sequence

- Sequence (list) of pairs: (token, docID).
- As part of this process, punctuation is generally removed.
- In this example, we also convert to lowercase.

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	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 step 2: Sort

- Sort this sequence by the terms (alphabetically) and then by the docIDs.

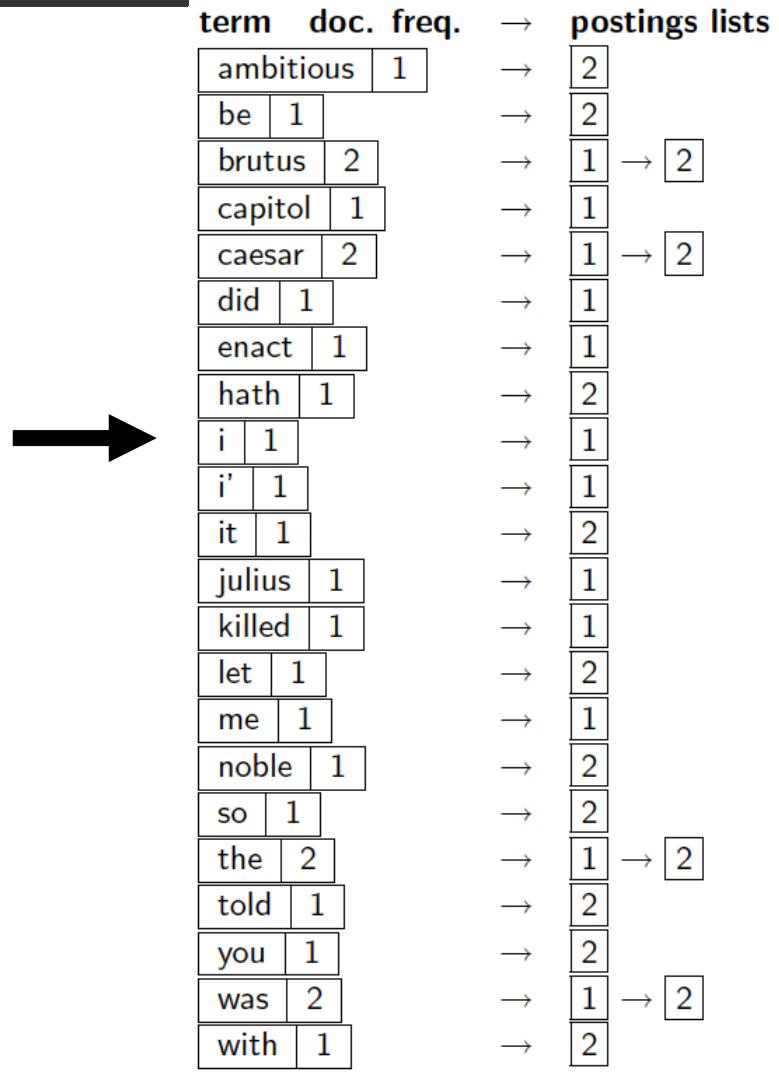
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

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 step 3: Dictionary & Postings

- If a term appears more than once in a document, these entries are merged (i.e. a term is recorded only once for each document).
- Split into a **dictionary** (alphabetical **order** of terms) and **postings** (**ordered** list of documents each term appears in).
  - Why is ordering important?
- We also record the **document frequency** of each term (the number of documents it appears in).
  - This information will be useful 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
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



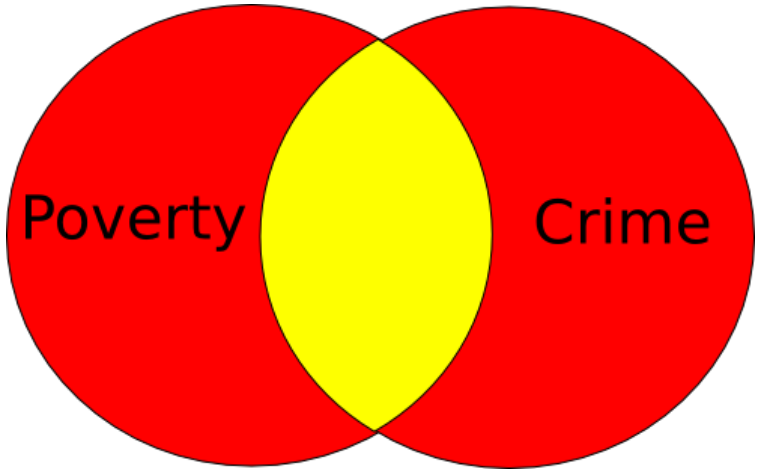
Using the Inverted Index: How do we process a query?

- Consider the following query:
  - ***Brutus*** *AND* ***Caesar***

1. Locate ***Brutus*** in the dictionary and retrieve its postings.
   - What is the time complexity for finding ***Brutus*** in our dictionary?
2. Locate ***Caesar*** in the dictionary and retrieve its postings.
3. Merge the two postings (find the **intersection**) of the document sets.
   - Each of our boolean operators (AND, OR, NOT) is equivalent to a set operation.

Set Theoretic Representation of AND

If *P* is the set of documents that contain the term "**poverty**" and *C* is the set of documents that contain the term "**crime**", the query "**poverty AND crime**" can be calculated by $P \cap C$ (i.e. the area in yellow).

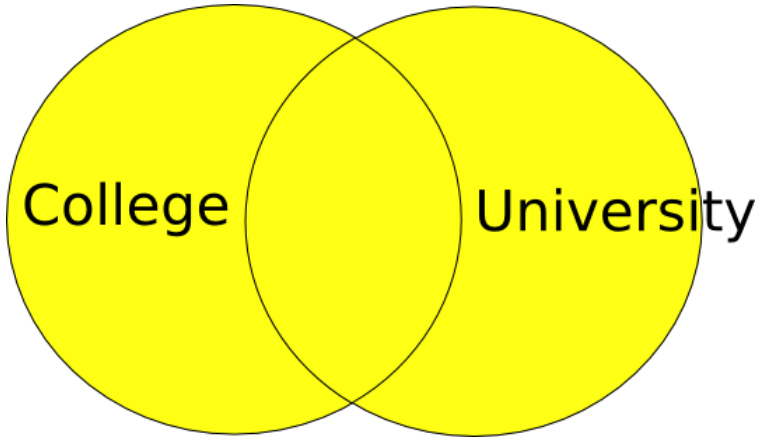


Boolean Operators: AND

- **AND** is used to **narrow** a search by ensuring that all the search terms should appear in the results.
- It is commonly used to search for relationships between two concepts or terms, for example:
  - *poverty*: 783,447 results
  - *crime*: 2,962,165 results
  - *poverty* **AND** *crime*: 1,677 results
- The more terms or concepts combined in a search with AND, the fewer records that will be retrieved.

Set Theoretic Representation of OR

If *C* is the set of documents that contain the term **"college"** and *U* is the set of documents that contain the term **"university"**, the query **"college OR university"** can be calculated by *C* ∪ *U*. (i.e. the area in yellow)

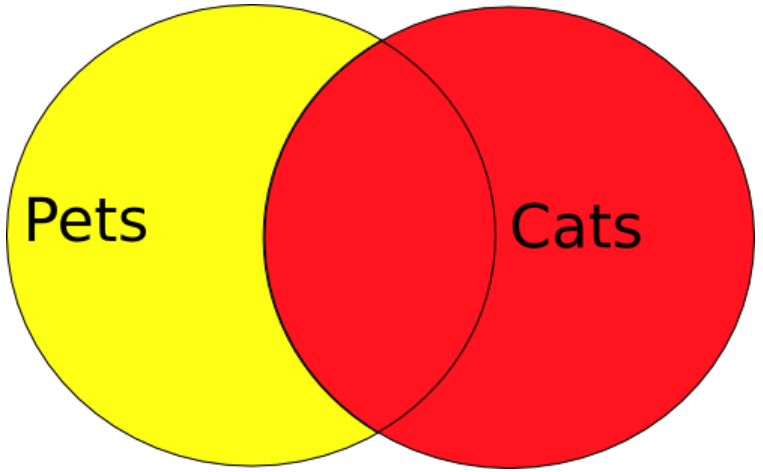


Boolean Operators: OR

- **OR** is used to **broaden** a search by retrieving any, some or all of the keywords used in the search statement.
- It is commonly used to search for synonymous terms or concepts, for example:
  - *college*: 17,320,770 results
  - *university*: 33,685,205 results
  - *college* **OR** *university*: 38,702,660 results
- Note that the final result is not the same as the sum of the individual results: documents containing both terms (e.g. "University College Dublin") will only be counted once.
- The effect of using OR is that documents containing **any number of the terms specified** will be returned.

Set Theoretic Representation of NOT

If *P* is the set of documents that contain the term "**pets**" and *C* is the set of documents that contain the term "**cats**", the query "**pets NOT cats**" can be calculated by P \ C (i.e. the area in yellow).

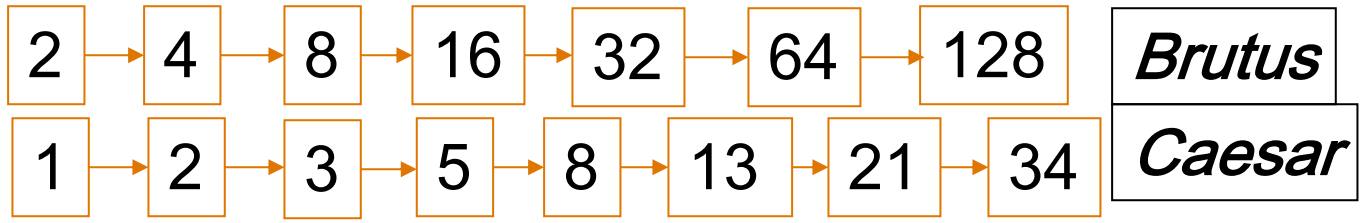


Boolean Model Operators: NOT

- **NOT** is used to **specifically exclude** a term from your search, for example:
  - *pets*: 4,556,515 results
  - *cats*: 3,651,252 results
  - *pets* **NOT** *cats*: 1,481,497 results
- One difficulty with using NOT is that a document that is highly relevant to what you're searching for may also contain the term you had attempted to avoid.
- Again, the more terms or concepts combined in a search with NOT, the fewer records that will be retrieved.

Merging postings lists

- How would you merge two postings lists into one?
  - **Hint:** Why is it important that the lists are sorted?



Merging two postings lists: Intersection (AND)

```
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 lengths are x and *y*, algorithm is O(x+y). **Why?**
- **Answer:**
  - Each iteration of the loop advances to the next document in at least one of the lists.
  - Worst case (two distinct lists with nothing in common):
    - x iterations to reach end of first list.
    - *y* iterations to reach end of second list.
    - *O(x+y)*
  - This **only** works because the lists are sorted by docID.

Exercise

- Can we adapt the merge for the following query types? How?
  - ***Brutus*** *OR* ***Caesar***
    - Documents that contain either the term ***Brutus*** or the term ***Caesar****, or both.*
  - ***Brutus*** *AND NOT* ***Caesar***
    - Documents that contain the term ***Brutus*** but do not contain the term ***Caesar***.
  - ***Brutus*** *OR NOT* ***Caesar***
    - Documents that contain ***Brutus*** or do not contain ***Caesar***.
- If not, what else do we need?

Questions?