Boolean Queries

* Use of AND, OR and NOT.

AND

E.g. Brutus and Caesar

* Retrieve the linked list postings of Brutus. Then retrieve the linked list posting of Caesar. Find the postings that both Brutus and Caesar appear in. This is implemented by merging.
* Merging means going through the lists at the same time.
* If the lists are x and y: The merge takes O(x + y)
* Postings need to be sorted by the docID

OR

* Use merge to give the union of all posting.

Questions:

* What is the time behavior of Brutus AND NOT Caesar? Need to return all the postings containing Brutus. Then need to return all the postings containing Caesar. Then need to find the postings that both Brutus and Caesar do not appear in. So the time complexity here is also O(x + y)
* Brutus OR NOT Caesar? All the documents that contain Brutus plus all the documents that do not contain Caesar. This is O(N) where N is the total number of documents in the collection of results bounded by N not length of postings list. \*\*\*

Arbitrary Boolean formulae? \*\*\*

E.g. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

* Does not compose of only AND, NOT or OR
* We can merge in O(qN) where q is the number of query terms and N is the number of documents.
* Cannot do better than O(N) (tightest bound for results list is number of documents in the collection)

Query optimization

* Process in order of increasing frequency. This is why the document frequency is recorded for each term.

Query Processing \*\*\*

* Friends AND romans AND (NOT countrymen)
* If NOT countrymen appears a lot then use the length of the postings list not in descending order.
* If NOT countrymen appears infrequently then use length of the postings list for ordering, process latter group last.

Boolean Model: Pros

* Simple and efficient to implement
* Precise: document matches or does not
* Used a lot for commercial, legal retrieval and for **specialist searches**

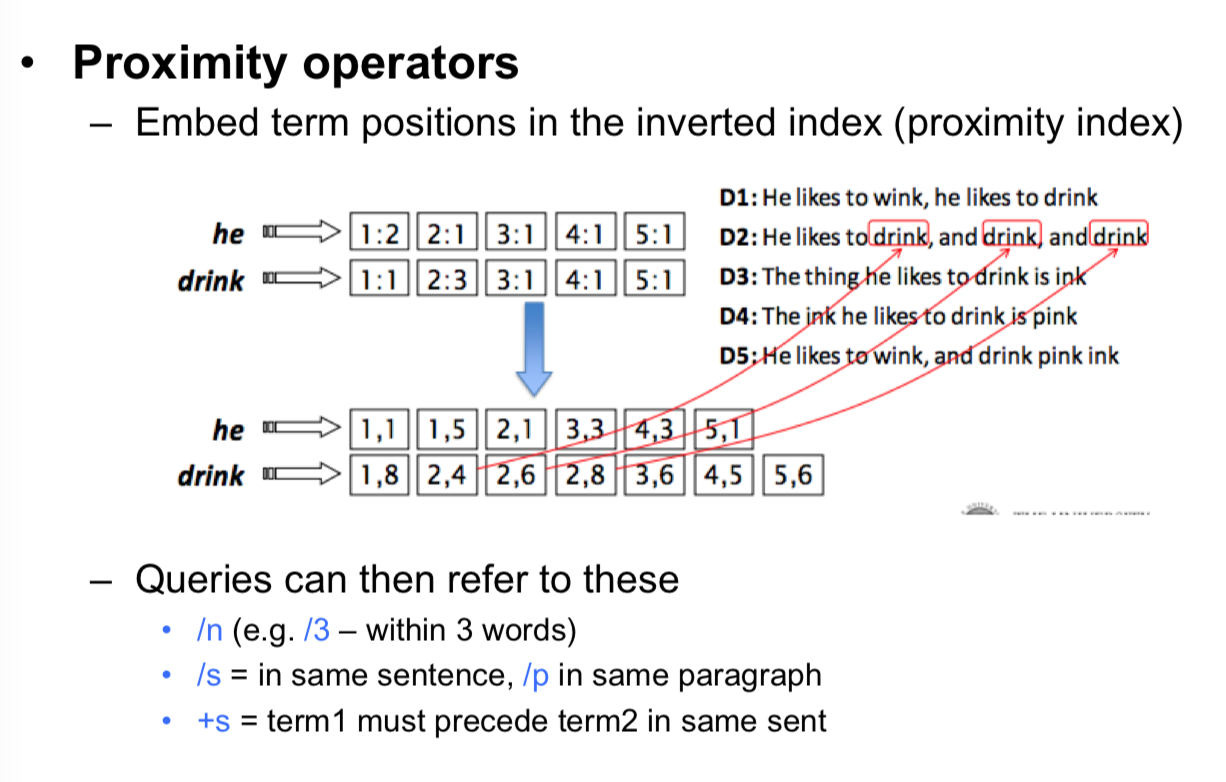
1. Long, precise queries
2. Works well when you know what you want
3. Users feel in control

* Many search systems hide the Boolean functionality (e.g. space is used as AND)

Boolean Model: Cons

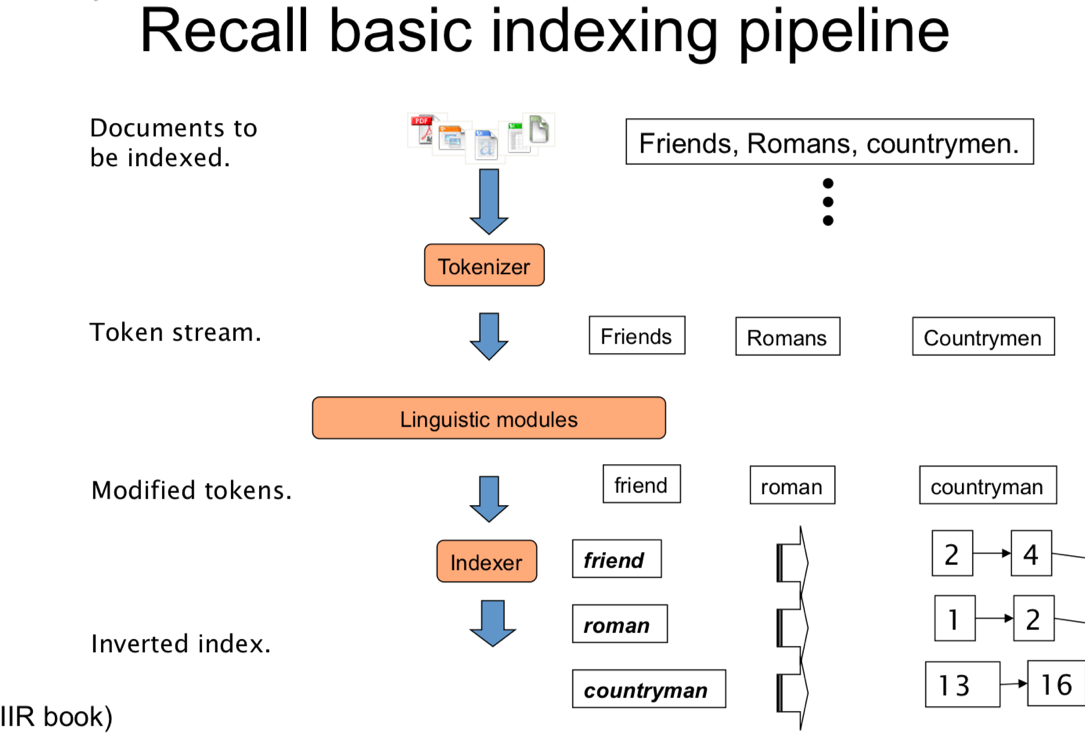
* Non expert users do not understand Boolean operators
* Confusion with natural language e.g. cats OR dogs. Which means cats and/or dogs in the Boolean model
* Documents that are close are not retrieved. For example: social AND worker AND union. If docs contain Social AND work but no union then not retrieved.
* Too ridged.

Extended Boolean Model



* Truncation: object!, this means will find the words object, objected, objection, objective etc.
* Similar to wildcards but does not have same flexibility e.g. reali\*se - > realise, realize

Workshop 3b: Preparing to index



**Document Conversion**

* Map representation such as .pdf, .doc, .xml to an internal representation.

**Language Identification**

* Need to identify the language of the document.
* Can be easily done
* For example google language API can easily do language identification.

Domain Identification

* Need to find out what the domain (Science, Math, Business etc. ) is and text type (Social media, News, Literature, Emails).
* Select domain resources (controlled vocabulary, lexicons, grammars and trained ML models)
* Punctuation varies with text type/domain
* Domain/type can be mixed (email with a report)

Tokenization

* Break input into tokens
* Not all tokens are words e.g. ‘-‘
* One definition for a token: “whitespace-delimited char sequence”
* How a word is sectioned depends on the language and alphabet.
* In Chinese and Japanese there is no whitespace between tokens. There are sentence boundaries, but no word boundaries.
* Arabic and Hebrew is written from right to left but not always. Words are separated but complex **ligatures** are used within words.
* English: whitespace plus punctuation marks, but there still is ambiguity in tokenization.
* Three classes of tokens are considered

1. Morphosyntactic: Has meaning and we know where it is derived from etc.
2. Punctuation mark or special symbol
3. A number

* But there are some tokens which cannot be classified in the above classes:
* Possessives: president’s speech, 1980’s, GP’s

Hyphenations : sister-in-law, e-mail, Manchester-based.

* Abbreviations: M/cr
* Numbers: 0.05, 123,456.78
* Telephone numbers: have many different formats
* Dates: 12 Oct 2018, 12/10/2018
* Multi-words: San Francisco, in spite of.
* Chemical names

Filtering: stop words

* Highly occurring words have low distinguishing power.
* Top 30 words account for ~30% of postings
* Closed class function words (the, and, of, it,…)
* ‘content free’ words (e.g information)
* Should filter out these words
* But it the trend it to not do that

Normalization

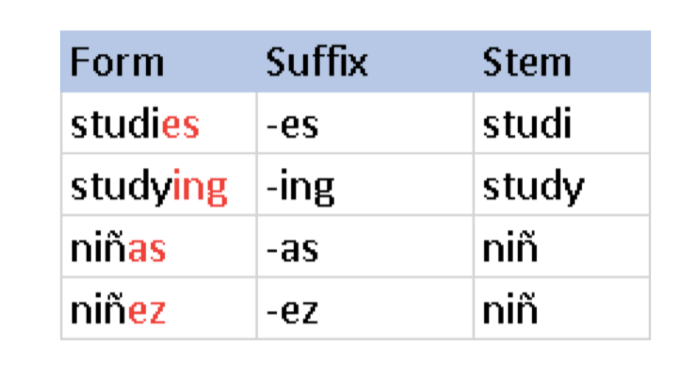
* Can map tokens to a normalized form.
* E.g B.B.C -> BBC
* {Multi-national, multinational} -> multinational
* Map all upper case to lower case.

**Lemmatization**

* Reduction to “dictionary headword” form.
* E.g {am, are, is} -> be
* The girls’ horses are different heights -? The girl horse be different height.
* Language dependent??
* But this many be slow

**Stemming**

* Remove the end of words before indexing.
* Collapse similar form to a canonical form.
* So would you reduce the query to a canonical form also?



* Under-stemming fails to conflate related terms

- divide -> divid

- division -> divis

* Over stemming conflates to unrelated forms

-neutron, neutral -> neutr

**Synonyms**

* {car, automobile}
* {favour, favour}

**Homonyms**

* Bank (river) vs. bank (finance)
* Bow (ship), bow (tie), bow (salute)
* Do research into stemming
* Why not remove stop words?
* Need to clarify query processing relating to negation
* Need to understand the complexity of arbitrary Boolean formulae

Try at home:

1. Explain why Boolean query processing is linear in number of documents and query terms
2. Explain the proximity operators in the extended Boolean model.

– Queries can then refer to these  
• /n(e.g./3–within3words)  
• /s=insamesentence,/pinsameparagraph • +s=term1 must precede term2 in same sent

3.  Have a look at example documents in Blackboard. Discuss how you would do tokenisation on these files. Apply a couple of existing tokenizers (see next pages) and discuss whether you are satisfied with the tokenisation results. To what extent your DOs and DON’Ts have been applied?

4.  Try different stemmers on the same texts. Do they match your expectations? What is more problematic for IR: under- or over- stemming?

I would say under-stemming because that would give less relevant results, whereas over stemming would at least give max relevant results and some irrelevant ones.

5.  How would lemmatisation help IR? What are the risks?

Lemmatisation is a form of indexing that changes words to their dictionary form. E.g. {am, are, is} -> be. This is good because it is language dependent and makes mapping a query to documents more easier and more accurate. Although the downsides are that to map the words you need to do a dictionary look up which may be slow.