

Information Retrieval and Organisation

Dell Zhang

Birkbeck, University of London

2015/16

Boolean Retrieval

Example IR Problem

- ▶ Let's look at a simple IR problem
 - ▶ Suppose you own a copy of Shakespeare's Collected Works
 - ▶ You are interested in finding out which plays contain the words Brutus AND Caesar AND NOT Calpurnia
- ▶ Possible solutions:
 - ▶ Start reading ...
 - ▶ Use string-matching algorithm (e.g. grep) scanning files
 - ▶ For simple queries on small to modest collections (Shakespeare's Collected Works contain not quite a million words) this is OK.

Limits of Scanning

- ▶ For many purposes, you need more:
 - ▶ Process large collections containing billions or trillions of words quickly
 - ▶ Allow for more flexible matching operations, e.g. Romans NEAR countrymen
 - ▶ Rank answers according to importance (when a large number of documents is returned)
- ▶ Let's look at the performance problem first:
 - ▶ Solution: do preprocessing

Term-Document Incidence Matrix

| | Anthony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth | ... |
|-----------|-----------------------------|------------------|----------------|--------|---------|---------|-----|
| Anthony | 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 | |
| ... | | | | | | | |

- ▶ Entry is 1 if term occurs.
 - ▶ Example: Calpurnia occurs in *Julius Caesar*.
- ▶ Entry is 0 if term doesn't occur.
 - ▶ Example: Calpurnia does not occur in *The Tempest*.

Incidence Vectors

- ▶ So we have a 0/1 vector for each term.
- ▶ To answer the query Brutus AND Caesar AND NOT Calpurnia:
 - ▶ Take the vectors for Brutus, Caesar, and Calpurnia
 - ▶ Complement the vector of Calpurnia
 - ▶ Do a (bitwise) AND on the three vectors
 - ▶ $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

Indexing Large Collections

- ▶ Consider $N = 10^6$ documents, each with about 1000 tokens
- ▶ On average 6 bytes per token, including spaces and punctuation \Rightarrow the size of document collection is about 6 GB
- ▶ Assume there are $M = 500,000$ distinct terms in the collection

Building Incidence Matrix


- ▶ $M = 500,000 \times 10^6 =$ half a trillion 0s and 1s.
 - ▶ We would use about 60GB to index 6GB of text, which is clearly very inefficient.
- ▶ But, wait a minute, the matrix has no more than one billion 1s.
 - ▶ The matrix is extremely sparse, i.e. 99.8% is filled with 0s.
- ▶ What is a better representations?
 - ▶ We only record the 1s.


Inverted Index

For each term t , we store a list of IDs of all documents that contain t .

documents that contain:

| | | | | | | | | | | |
|-----------|---|---|----|----|-----|----|----|-----|-----|-----|
| Brutus | → | 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174 | |
| Caesar | → | 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ... |
| Calpurnia | → | 2 | 31 | 54 | 101 | | | | | |
| ⋮ | | | | | | | | | | |


dictionary


postings

Index Construction

- ▶ Collect the documents to be indexed:

Friends, Romans, countrymen. So let it be with Caesar ...

- ▶ Tokenize the text, turning each document into a list of tokens:

Friends Romans countrymen So ...

- ▶ Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:

friend roman countryman so ...

- ▶ Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

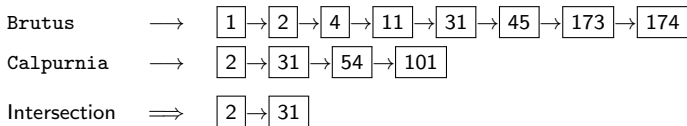
Index Construction

- ▶ Later on in this module, we'll talk about optimizing inverted indexes:
 - ▶ Index construction: how can we create inverted indexes for large collections?
 - ▶ How much space do we need for dictionary and index?
 - ▶ Index compression: how can we efficiently store and process indexes for large collections?
 - ▶ Ranked retrieval: what does the inverted index look like when we want the “best” answer?

Processing Boolean Queries

- ▶ Consider the conjunctive query:
 - ▶ Brutus AND Calpurnia
- ▶ To find all matching documents using inverted index:
 1. Locate Brutus in the dictionary
 2. Retrieve its postings list from the postings file
 3. Locate Calpurnia in the dictionary
 4. Retrieve its postings list from the postings file
 5. Intersect the two postings lists
 6. Return intersection to user

Intersecting Postings Lists



- Can be done in linear time if postings lists are sorted

Intersecting Postings Lists

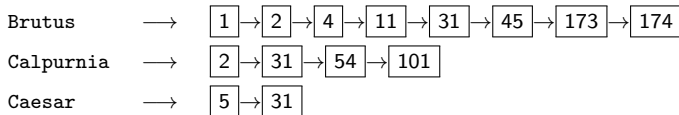
```
INTERSECT( $p_1, p_2$ )  
1   $answer \leftarrow \langle \rangle$   
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$   
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$   
4      then  $\text{ADD}(answer, \text{docID}(p_1))$   
5           $p_1 \leftarrow \text{next}(p_1)$   
6           $p_2 \leftarrow \text{next}(p_2)$   
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$   
8          then  $p_1 \leftarrow \text{next}(p_1)$   
9          else  $p_2 \leftarrow \text{next}(p_2)$   
10 return  $answer$ 
```

Mapping Operators to Lists

- ▶ The Boolean operators AND, OR, and NOT are evaluated as follows:
 - ▶ `term1 AND term2`: intersection of the lists for `term1` and `term2`
 - ▶ `term1 OR term2`: union of the lists for `term1` and `term2`
 - ▶ `NOT term1`: complement of the list for `term1`

Query Optimization

- ▶ What is the best order for query processing?
- ▶ Consider a query that is an AND of n terms, $n > 2$
- ▶ For each of the terms, get its postings list, then AND them together
- ▶ Example query:
 - ▶ Brutus AND Calpurnia AND Caesar



Query Optimization

- ▶ Simple and effective optimization:
 - ▶ Process in the 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

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

Commercial Boolean IR: Westlaw

- ▶ Largest commercial legal search service in terms of the number of paying subscribers (www.westlaw.com)
- ▶ Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- ▶ The service was started in 1975.
- ▶ In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users . . .
- ▶ . . . although ranked retrieval has been available since 1992.

Westlaw Example Queries

- ▶ *Information need:* Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company
 - ▶ “trade secret” /s disclos! /s prevent /s employe!
- ▶ *Information need:* Requirements for disabled people to be able to access a workplace
 - ▶ disab! /p access! /s work-site work-place (employment /3 place)
- ▶ *Information need:* Cases about a host's responsibility for drunk guests
 - ▶ host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Westlaw Example Queries

- ▶ /s = within same sentence
- ▶ /p = within same paragraph
- ▶ /n = within n words
- ▶ Space is disjunction, not conjunction (This was the default in search pre-Google.)
- ▶ & is AND
- ▶ ! is a trailing wildcard query

Summary

- ▶ The Boolean retrieval model can answer any query that is a Boolean expression.
 - ▶ Boolean queries are queries that use AND, OR and NOT to join query terms.
 - ▶ Views each document as a **set** of terms.
 - ▶ It is precise: document matches condition or not.
- ▶ Primary commercial retrieval tool for 3 decades
- ▶ Many professional searchers (e.g., lawyers) still like Boolean queries
 - ▶ You know exactly what you are getting.
- ▶ When are Boolean queries the best way of searching?
 - ▶ It depends on: information need, searcher, document collection, ...