CSCI 5417
Information Retrieval Systems
Jim Martin

Lecture 2
8/25/2011

Today 8/25

- Basic indexing, retrieval scenario
- Boolean query processing
- More on terms and tokens

Simple Unstructured Data Scenario

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- We could `grep` all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***. This is problematic:
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Lines vs. Plays

3

Grepping is Not an Option

- So if we can't search the documents in response to a query what can we do?
- Create a data structure up front that will facilitate the kind of searching we want to do.

4

Term-Document Matrix

	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

1 if play contains word, 0 otherwise

Brutus *AND* ***Caesar*** but *NOT* ***Calpurnia***

5

Incidence Vectors

- So we have a 0/1 vector for each term
  - Length of the term vector = number of plays
- To answer our query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) and then do a bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100
  - That is, plays 1 and 4
  - "Antony and Cleopatra" and "Hamlet"

6

Answers to Query

- Antony and Cleopatra, Act III, Scene ii
- *Agrippa* [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
- When Antony found Julius ***Caesar*** dead,
- He cried almost to roaring; and he wept
- When at Philippi he found ***Brutus*** slain.

- Hamlet, Act III, Scene ii
- *Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the
- Capitol; ***Brutus*** killed me.

7

Bigger Collections

- Consider *N* = 1M documents, each with about 1K terms.
- Avg 6 bytes/term including spaces and punctuation
  - 6GB of data just for the documents.
- Assume there are *m* = 500K *distinct* terms among these.
  - Types

8

The Matrix

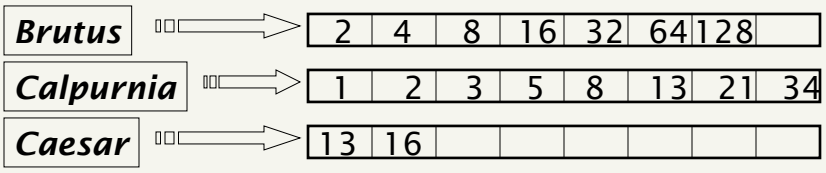
- 500K x 1M matrix has 1/2 trillion entries
- But it has no more than one billion 1's
  - Matrix is extremely sparse.
  - What's the minimum number of 1's in such an index?
- What's a better representation?
  - Forget the 0's. Only record the 1's.

Why?

9

Inverted index

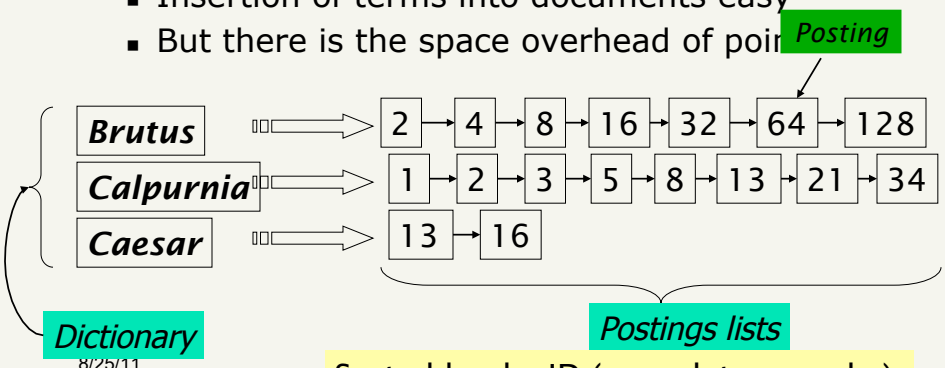
- For each term *T*, we must store a list of all documents that contain *T*.



What happens if the word ***Caesar*** is later added to document 14?

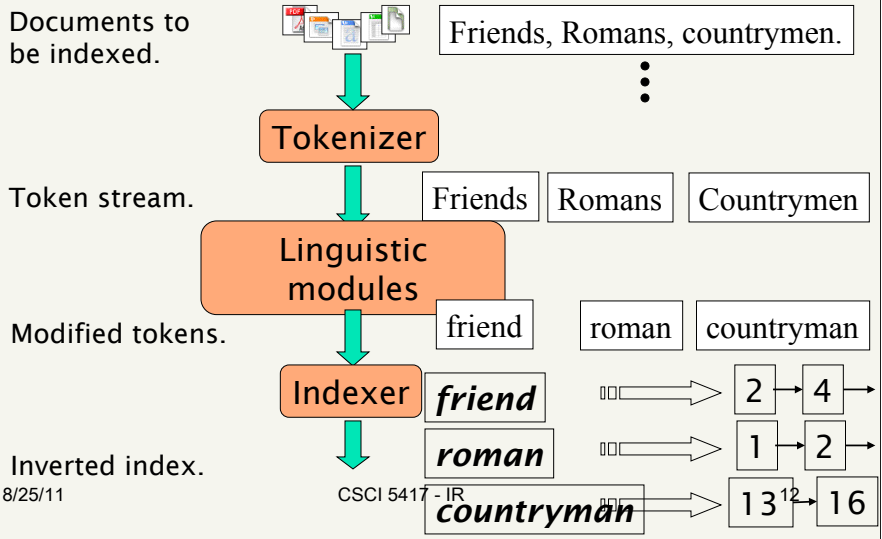
Inverted index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - But there is the space overhead of poi



Sorted by docID (more later on why).

Index Creation



Indexer steps

- From the documents generate a stream of (Modified token, Document ID) pairs.

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	Doc #
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

- Sort pairs by terms.

Core indexing step.

Term	Doc #
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	Doc #
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

- Multiple term entries in a single document are merged.
- Frequency information is added.

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

- The result is then split into a *Dictionary* file and a *Postings* file.

Where's the primary storage cost?

Why split into two files?

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

docs	Coll freq
1	1
1	1
2	2
1	1
2	3
1	1
1	1
1	2
1	1
1	1
1	2
1	1
1	1
1	2
1	1
1	1
2	2
1	1
2	3

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Indexing

Of course you wouldn't really do it that way for large collections. Why?

The indexer would be too slow

17

Given an Index

- So what is such an index good for?
  - Processing queries to get documents
  - What's a query?
    - An encoding of a user's information need
    - For now we'll keep it simple: boolean logic over terms.

Example: WestLaw http://www.westlaw.com/

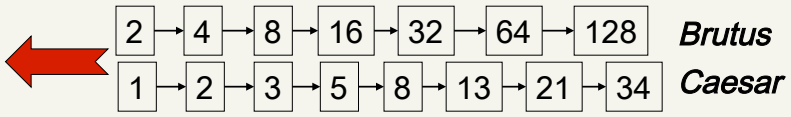
- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- Tens of terabytes of data; 700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- /3 = within 3 words, /S = in same sentence

Boolean queries: Exact match

- The Boolean retrieval model is able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: a document matches condition or not
  - Perhaps the simplest model of an IR system
- Primary commercial retrieval tool for 3 decades
- Many search systems you still use are Boolean:
  - Email, library catalog, Mac OS X Spotlight

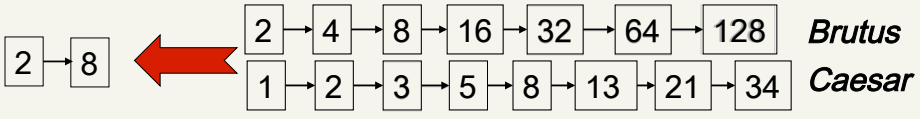
Query processing: AND

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



The Merge (Intersection)

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.
Crucial: postings sorted by docID.

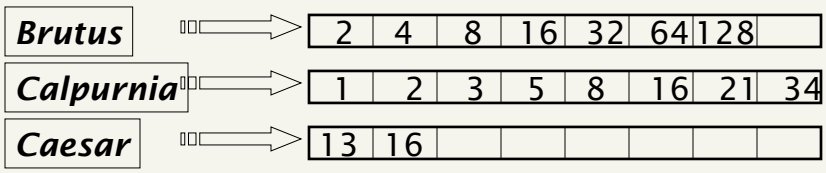
Intersecting two postings lists (a "merge" algorithm)

```
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
```

23

Query optimization

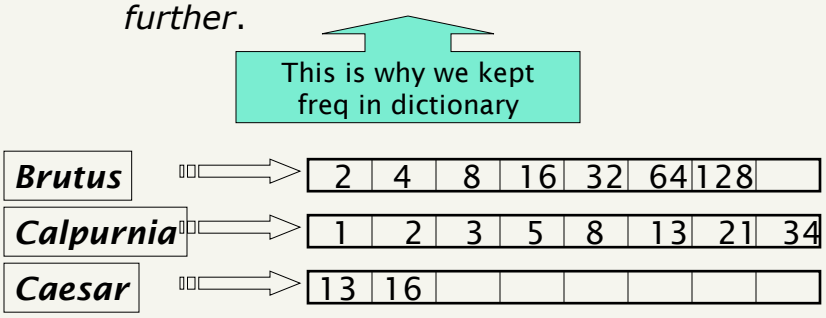
- What is the best order for query processing?
- Consider a query that is an *AND* of *t* terms.
- For each of the *t* terms, get its postings, then *AND* them together.



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

Query optimization example

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



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

More general optimization

- For example
  - *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Get frequencies for all terms
- Estimate the size of each *OR* by the sum of its frequencies (conservative)
- Process in increasing order of *OR* sizes.

Break

- Waitlist
  - Is everyone ok?
- Homework 1

Assignment 1: Due 9/1

Build an indexer that produces a postings file for a small document collection (MED)

```
.I 13
.W
analysis of mammalian lens proteins by electrophoresis .
  lens proteins of different mammalian species were analyzed
by two-dimensional starch gel electrophoresis . the number of
fractions detected by this means varied from 11-20 .
a-crystallin was resolved into two to three components,
b-crystallin into 5-11, and y-crystallin into three to five
components . this technique provides a sensitive method for
the fractionation of lens proteins and for analyzing species
differences .
```

Assignment 1: Due 9/1

- More specifically, a file with
  - One line for each term in the collection
  - Sorted alphabetically by terms
  - With a postings list for each term
    - Sort by document number
- Terms are...
  - Maximal sequences of alphanumerics and dashes
- Don't use Lucene; any programming language is ok.

Terms Revisited

- What's a term and how do we find them?
  - Tokenizing
  - Stop lists
  - Stemming
  - Multi-word units

Tokenization

- Input: "*Friends, Romans and Countrymen*"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***and***
  - ***Countrymen***
- Each such token is now a candidate for an index entry, after further processing
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - *Finland's capital* →
    Finland? Finlands? Finland's?
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
    - ***State-of-the-art***: break up hyphenated sequence
      - Sometimes
        - Lists, machine learning and voodoo
  - ***San Francisco***: one token or two?
    - How do you decide if it is one token?

Numbers

- *3/12/91* *Mar. 12, 1991*
- *55 B.C.*
- *303*
- *11222*
- *324a3df234cb23e*
- *100.2.86.144*
  - Often, indexed by semantic type (if known)

Tokenization: Language issues

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

- German noun compounds
  - Lebensversicherungsgesellschaftsangestellter
  - 'life insurance company employee'

Tokenization: Language issues

- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique segmentation
- Further complicated when alphabets can intermingle

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

- 'Algeria achieved its independence in 1962 after 132 years of French occupation.'

Normalization

- May want to "normalize" terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- Most commonly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but difficult to discover

Normalization: other languages

- Accents: ***résumé*** vs. ***resume***.
- Most important criterion:
  - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
- German: Tuebingen vs. Tübingen
  - Should be equivalent

Case folding

- Reduce all letters to lower case
  - exception: upper case (in mid-sentence?)
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***IRA*** vs. ***Ira***
  - May require named entity recognition
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

Stop words

- With a stop list, you exclude from dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - They take a lot of space: ~30% of postings for top 30
- But the trend is now away from doing this:
  - Good index compression techniques means the space for including stopwords in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: "King of Denmark"
    - Various song titles, etc.: "Let it be", "To be or not to be"
    - "Relational" queries: "flights to London" vs. "flights from London"

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is → be*
  - *car, cars, car's, cars' → car*
- *the boy's cars are different colors → the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form

Next time

- Read Chapters 1 through 3 of IIR for next Tuesday.