# Information Retrieval & Social Web

CS 525/DS 595
Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

## Grader

Meng Wang

## Unofficial TAs

Thanh Tran

Nguyen "Ben" Vo

## What is Information Retrieval?

- …
- …
- …

## IR is Everywhere …

Web search engines

Recommenders

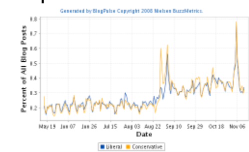
Domain-specific search

Info filtering / classification

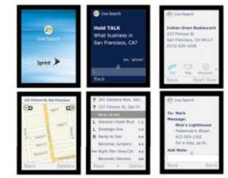
## IR is Everywhere …

Social search/Web

Topic detection and tracking



Mobile and location-based



## IR is Everywhere …

- Domain specific applications of information retrieval
  - Expert search finding
  - Genomic information retrieval
  - Geographic information retrieval
  - Information retrieval for chemical structures
  - Information retrieval in software engineering
  - Legal information retrieval
  - Vertical search (domain/topic specific search)

## IR is Everywhere …

- General applications of information retrieval
  - Digital Libraries
  - Information Filtering
    - Recommender Systems
  - Media Search
    - Blog, image, music, news, speech, video
  - Search engines
    - Desktop, enterprise, federated, mobile, social, Web search

## IR is Everywhere …

- Other retrieval methods
  - Adversarial information retrieval
  - Automatic document summarization
  - Cross-lingual retrieval
  - Document classification
    - Spam filtering
  - Question answering
  - Structured document retrieval
  - Topic detection and tracking

## This course

- What makes a system like Google, Yahoo, or Bing?
  - How does it gather information?
  - What tricks does it use?
- How can those approaches be made better?
- What can we do to make things work more quickly?
- How do we decide whether it works well?
- ...

## So ... What is Information Retrieval?

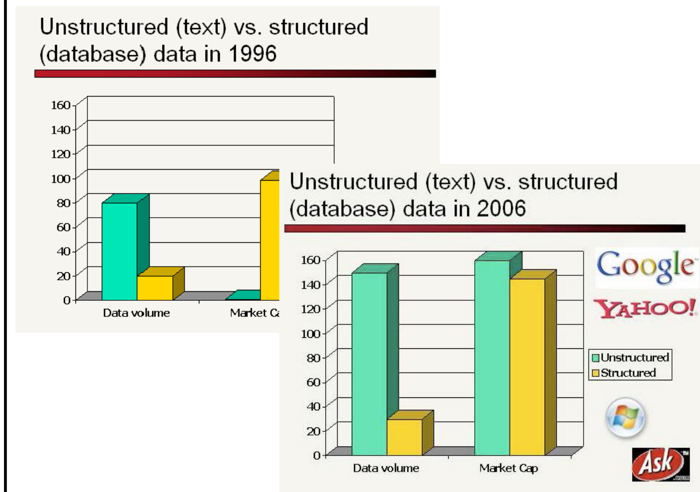
- IR is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

This smells a bit like Databases ...

What's the difference?

## Information Retrieval versus Databases

| | Databases | IR |
|---|---|---|
| Data | **Structured** | **Unstructured** |
| Fields | **Clear semantics** (SSN, age) | **No fields** (other than text) |
| Queries | **Defined** (relational algebra, SQL) | **Free text** (natural language, Boolean) |
| Recoverability | **Critical** (concurrency control, recovery, atomic operations) | **Downplayed** (though still an issue) |
| Matching | **Exact** (results are always "correct") | **Imprecise** (need to measure effectiveness) |




## Why take this course?

- IR is at the core of CS
- IR is incredibly important to society (and you?)
- The topic is intellectually rich
- It's not that much work
- Looks good on your resume

## Why take this course?

**IR is at the core of CS**

- Shift from computation to information
  - True in corporate computing for years
  - Web, P2P made this clear for personal computing
  - Increasingly true of scientific computing
- Need for IR technology has exploded in the last few years
  - Web: Search engines, e-commerce, blogs, wikis, other "web services"
  - Corporate: enterprise knowledge management, search, etc.
  - Scientific: Digital libraries, genomics, satellite imagery, physical sensors, simulation data
  - Personal: Music, photo, & video libraries. Email archives. File contents ("desktop search")

## Why take this course?

**IR is incredibly important to society**

- “Knowledge is Power” -- Sir Francis Bacon
- “With great power comes great responsibility” -- Uncle Ben

Policy-makers should understand technological possibilities.
Informed Technologists needed in public discourse on usage.

## Why take this course?

**The topic is intellectually rich**

- How do we model and represent data, information, and knowledge?
- Foundations in mathematics, computer science, natural language processing, statistics, psychology, ...
- New problems every day ... information overload and how to deal with it

## ~~Why take this course?~~

~~**It’s not that much work**~~

## Why take this course?

**Looks good on your resume**

- Yes, but why?
  - It is a course for well-educated computer scientists and data scientists
  - Information retrieval system concepts and techniques increasingly used “outside the box”
  - Ask your friends at Microsoft, Google, Apple, etc.
  - Actually, they may or may not realize it!
  - A rich understanding of these issues is a basic and (un?) fortunately unusual skill.

# Course Objectives

- Introduce
  - theory, design, and implementation of text-based and Web-based information retrieval systems.
- Study
  - crawling, Indexing, vector space model, web search, link-based algorithms, and etc.

## Goal of the Class

- Understand the key concepts and models relevant to information storage and retrieval, including efficient text indexing, vector space model, Web search.
- Design, implement, and evaluate the core algorithms underlying a fully functional IR system, including the indexing, retrieval, and ranking components.
- Identify the salient features and apply recent research results in information storage and retrieval, including topics such as adversarial information retrieval, question answering, and social information management.

# Course Structure and Administrivia

## Course Information

- Instructor
  - Kyumin Lee
  - kmlee@wpi.edu
  - Office: Fuller Labs 130
  - Office hours: T: 9:30-10:30am, W: 4:00-5:00pm or by appointment
- Grader
  - Meng Wang
  - mwang2@wpi.edu
- Unofficial TAs
  - Thanh Tran, thanhtd.ithut@gmail.com
  - Nguyen "Ben" Vo, vknguyen09@gmail.com
  - Office:
  - Office hours:
- Class hours:
  - 6:00 ~ 8:50 pm W
  - Fuller Labs 311

## Course Information

- Course web page
  - http://web.cs.wpi.edu/~kmlee/cs525
  - Check frequently
- Canvas
  - https://canvas.wpi.edu/
- Sign up our Google Groups
  - https://groups.google.com/d/forum/cs525-spring2018
- Our group mailing list
  - cs525-spring2018@googlegroups.com

## Course Materials

- Primary Textbook:
  - Introduction to Information Retrieval (2008)
- References
  - Mining of Massive Datasets (2014).
  - Data-Intensive Text Processing with MapReduce (2010).

## Course Communication

- The website (especially, schedule page) will be updated often
  - Check it regularly
- I will email important announcements and post them to the website
- You may email me anytime ... but I only guarantee a response within three days
- The best way to discuss general questions or share something cool stuff is to email it to our google group.

## Class Structure

- Lectures
  - By instructor -- I'll teach information retrieval techniques
  - By us - Discussion and interaction in the class
- Your part
  - Homework
    - 4 assignments
  - Exams
  - Project
    - Proposal, execution, workshop presentation
- Participation
  - Ask good questions

## Grading

- 5% Attendance and In-class discussion
- 24% (four) Assignments
- 20% Midterm
- 20% Final
- 31% Project

## Assignments

- 4 assignments
  - Be familiar with Python
- Submit your solution to Canvas
  - You only use Canvas for submitting your assignments
- Late day policy: look at the syllabus

# Midterm and Final

## Exams

- The exams are closed book.
- You may bring one standard 8.5" by 11" piece of paper with any notes you think appropriate or significant (front and back).
- No electronic devices allowed.

# Project

## The Project

- 3 or 4-person team
- Project idea:
  - Propose anything you wish (related to IR and/or social systems)
  - You are encouraged to talk to me
- **31% of your final grade!!**

## Project Grading Criteria

- [7%] Project Proposal: March 18 by 11:59pm
- [8%] Project website: April 24 by 11:59pm
- [16%] Project Workshop: April 25 in-class

## So far…

- Sign up our Google Groups
  - https://groups.google.com/d/forum/cs525-spring2018
- Be familiar with Python for Assignments
- Form a team and notify the names of your team by Feb 1.

## So far…

- Information Retrieval (IR) is finding material (usually documents) of an **unstructured** nature (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).

## Next

Boolean Retrieval

## Unstructured data in 1680

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- One could `grep` all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
- Why is that not the answer?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return)
    - The **key feature** of modern search engines

## Term-document incidence

| | 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 |

***Brutus** AND **Caesar** BUT NOT **Calpurnia***

1 if play contains word, 0 otherwise

## Term-document incidence

| | 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 |

***Brutus** AND **Caesar** BUT NOT **Calpurnia***

1 if play contains word, 0 otherwise

## Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) → bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100.

## 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.

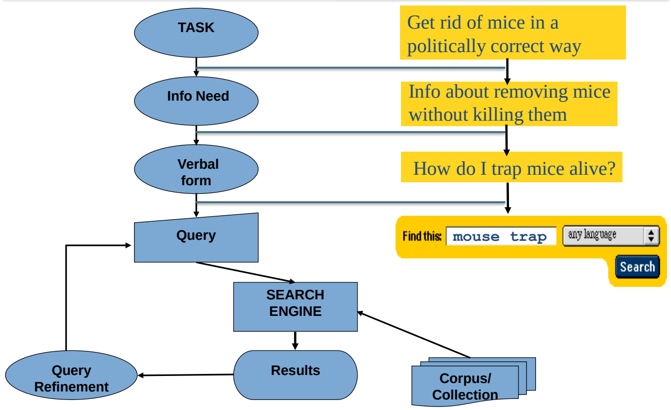
- Hamlet, Act III, Scene ii

*Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the Capitol; ***Brutus*** killed me.

## Basic assumptions of Information Retrieval

- Collection: Fixed set of documents
- Goal: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task

## The classic search model




## How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved
- More precise definitions and measurements to follow in later lectures

## Bigger collections

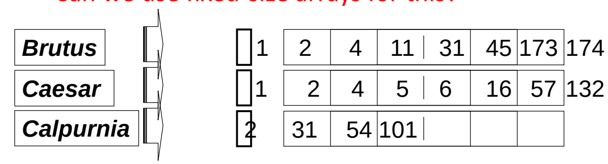
- Consider $N$ = 1 million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
  - 6GB of data in the documents.
- Say there are $M$ = 500K *distinct* terms among these.

## Can't build the matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's. Why?
  - matrix is extremely sparse.
- What's a better representation?
  - We only record the 1 positions.

## Inverted index

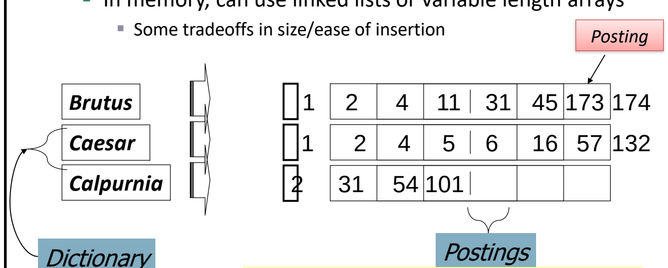
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number
- Can we use fixed-size arrays for this?




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

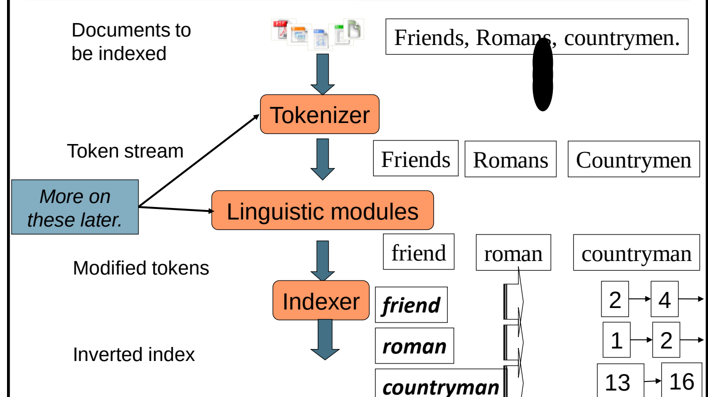
## Inverted index

- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion




Sorted by docID (more later on why).

## Inverted index construction



## Indexer steps: Token sequence

- Sequence 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 | 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 steps: Sort

- Sort by terms
  - And then docID

Core indexing step

| 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 steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency? Will discuss 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 |

| 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 |

## Where do we pay in storage?

Terms and counts


| 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 |


Lists of docIDs

Pointers

IR system implementation
- How do we index efficiently?
- How much storage do we need?

## The index we just built

- How do we process a query?
  - Later - what kinds of queries can we process?

Today's focus

## 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:

2 → 4 → 8 → 16 → 32 → 64 → 128 ***Brutus***

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 ***Caesar***

## The merge

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

2 → 8

2 → 4 → 8 → 16 → 32 → 64 → 128 ***Brutus***

1 → 2 → 3 → 5 → 8 → 13 → 21 → 34 ***Caesar***

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

## Intersecting two postings lists (a "merge" algorithm)

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
return answer
```

## Boolean queries: Exact match

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

## 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: More general merges

- Exercise: Adapt the merge for the queries:

  ***Brutus** AND NOT **Caesar***

  ***Brutus** OR NOT **Caesar***

Can we still run through the merge in time O($x$+$y$)?

What can we achieve?

## Exercise Solution

- Brutus AND NOT Caesar
  - Time is O(x+y). Instead of collecting documents that occur in both postings lists, collect those that occur in the first one and not in the second
- Brutus OR NOT Caesar
  - Time is O(N) (where N is the total number of documents in the collection) assuming we need to return a complete list of all documents satisfying the query. This is because the length of the result list is only bounded by N, not by the length of the postings lists.

## Merging

What about an arbitrary Boolean formula?

***(Brutus** OR **Caesar)** AND NOT*

***(Antony** OR **Cleopatra)***

- Can we always merge in “linear” time?
  - Linear in what?
- Can we do better?

## Solution

- We can always intersect in O(qN) where q is the number of query terms and N the number of documents, so the intersection time is linear in the number of documents and query terms. Since the tightest bound for the size of the result list is N, the number of documents, one cannot do better than O(N).

- But… still we can reduce computation time even though time complexity is still O(N). How?