https://vvtesh.sarahah.com/

# Information Retrieval

Venkatesh Vinayakarao

Term: Aug – Dec, 2018
Indian Institute of Information Technology, Sri City

**My brain is like the Bermuda Triangle.**

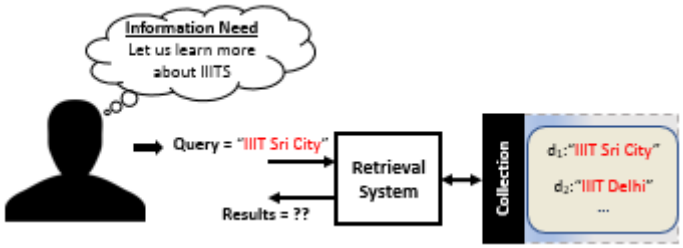
**Information goes in and is never found again.**

Source – Unknown.

# Agenda

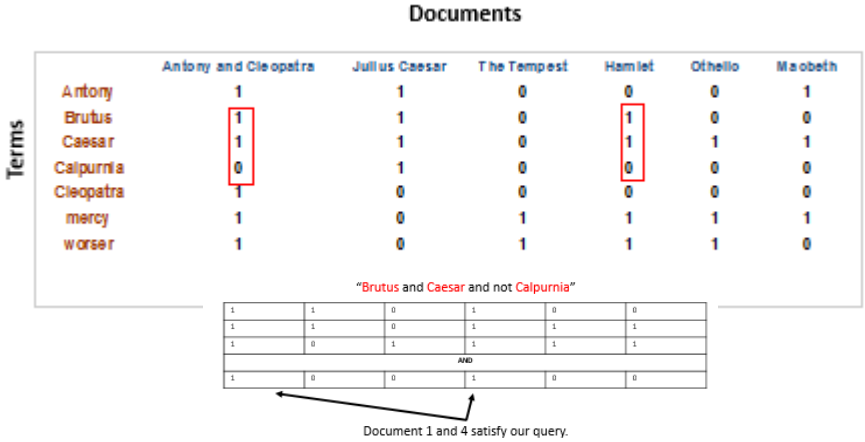
- Review
- Index Construction
  - Posting List and Inverted Index
  - Building the Index
- Introduction to Evaluation
  - Precision and Recall

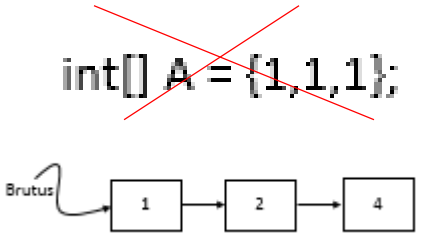
# Review




## One (bad) Approach

- First match the **term** IIIT.
  - Filter out documents that contain this term.
- Next match the **term** Sri.
  - Filter out documents that contain this term.
- Next match the **term** City.
  - Filter out documents that contain this term.



# Quiz

- Considering the following vectors:

| | IIIT | Sri | City | Delhi |
|---|---|---|---|---|
| q | 1 | 1 | 1 | 0 |
| $d_1$ | 1 | 1 | 1 | 0 |
| $d_2$ | 1 | 0 | 0 | 1 |

- What is the Natural Language (NL) equivalent of (0,1,1,0) ?
- What is the NL equivalent of (1,0,0,1) ?
- What is the vector for Delhi?
- If q represents query, d1 and d2 are documents, what is the NL query here?

# Tokenization

- Task
  - Chop documents into pieces.
  - Throw away characters such as punctuations.
  - Remaining terms are called tokens.
- Example
  - Document 1
    - I did enact Julius Caesar. I was killed i' the Capitol; Brutus killed me.
  - Document 2
    - So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

| | |
|---|---|
| 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

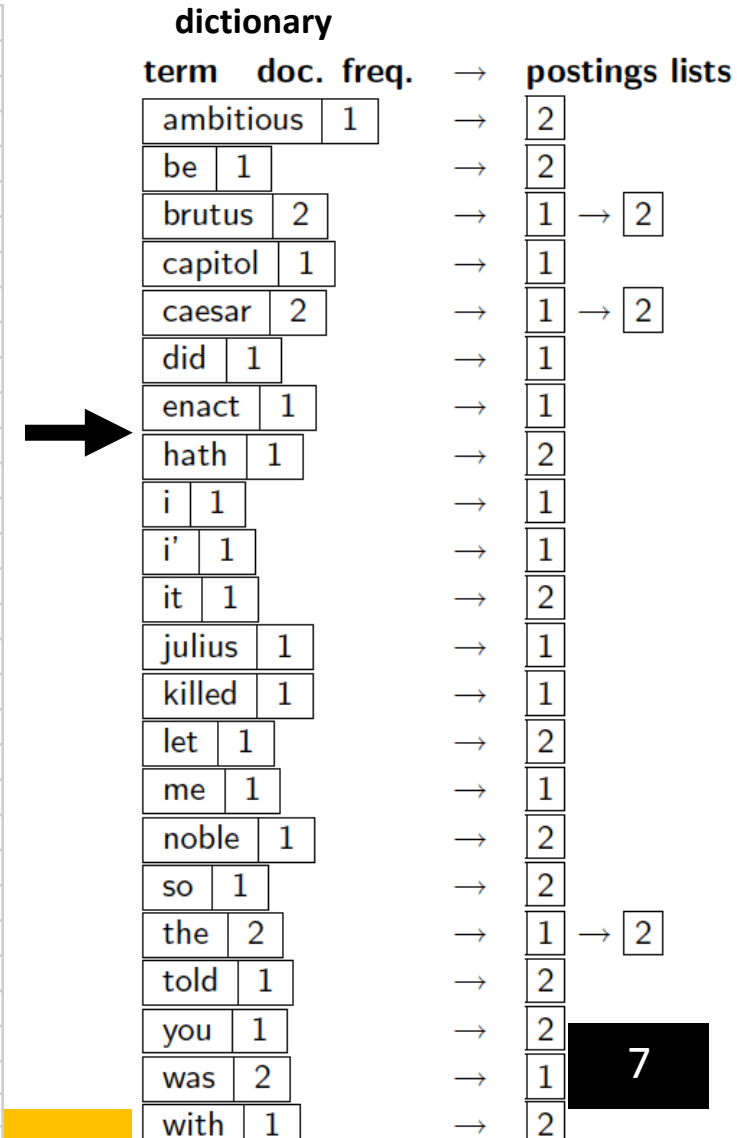
| Term | docID |
|---|---|
| I | 1 |
| did | 1 |
| enact | 1 |
| julius | 1 |
| caesar | 1 |
| I | 1 |
| was | 1 |
| killed | 1 |
| i' | 1 |
| the | 1 |
| capitol | 1 |
| brutus | 1 |
| killed | 1 |
| me | 1 |
| so | 2 |
| let | 2 |
| it | 2 |
| be | 2 |
| with | 2 |
| caesar | 2 |
| the | 2 |
| noble | 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 |

# Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings

| 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 |



# Query Processing with Inverted Index

# Boolean queries: Exact match

- The Boolean retrieval model is being 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: document matches condition or not.
  - Perhaps the simplest model to build an IR system on

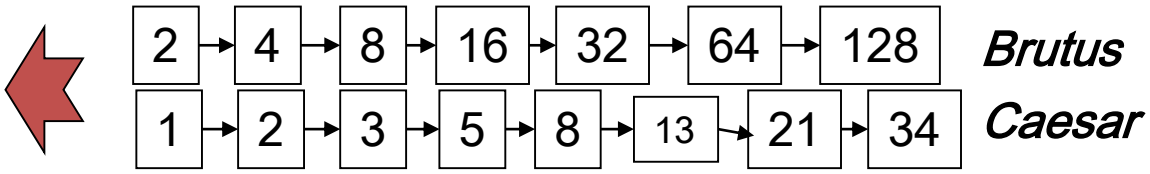
# Query processing: AND



- Consider processing the query:

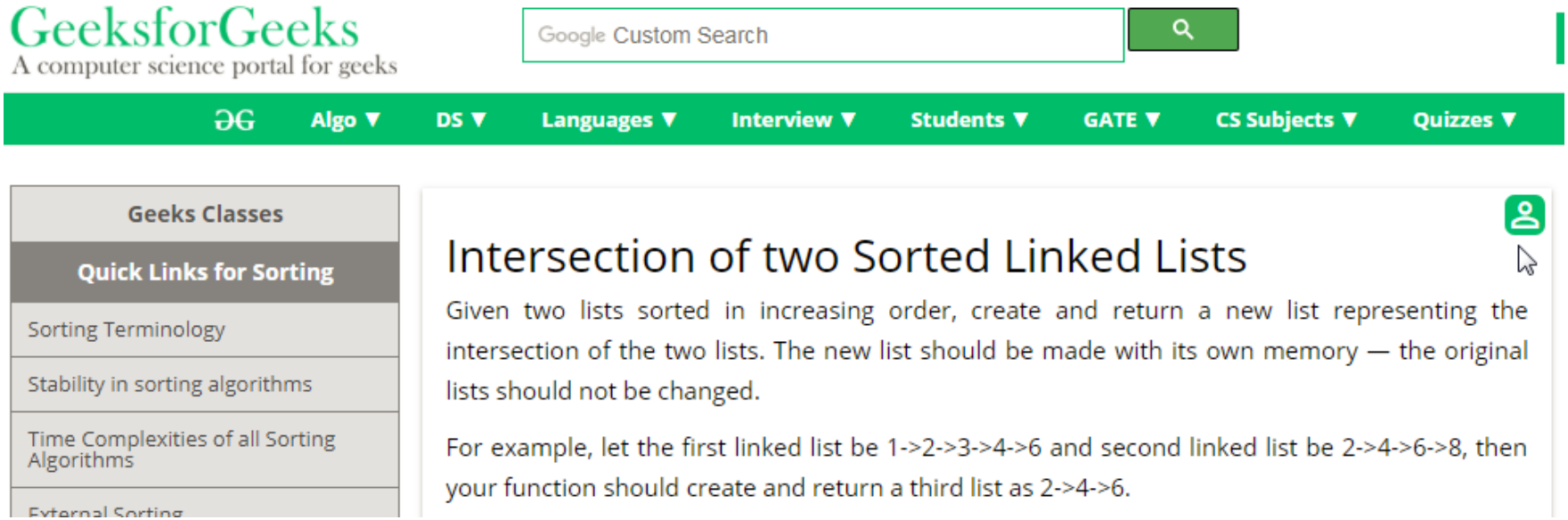
  ***Brutus*** *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings (intersect the document sets):



# Common Interview Question

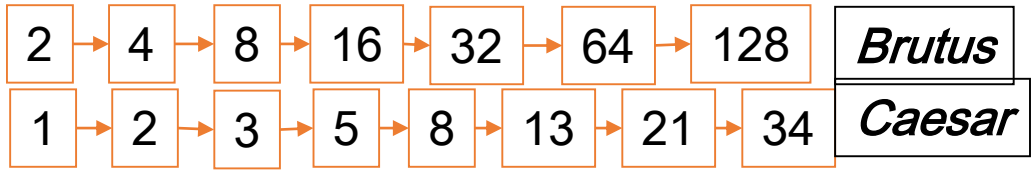
- https://www.geeksforgeeks.org/intersection-of-two-sorted-linked-lists/



# The Merge



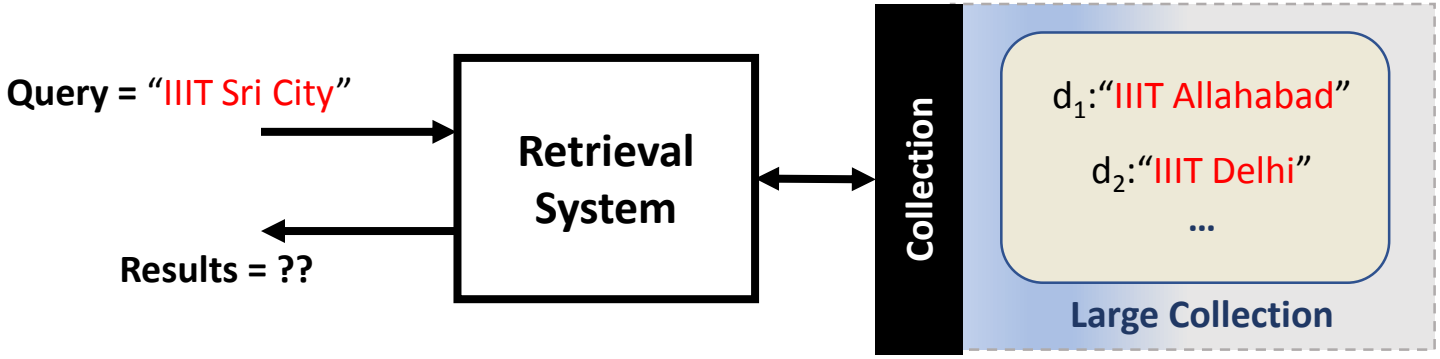
- Walk through the two postings simultaneously
  - Clue: Use two pointers




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

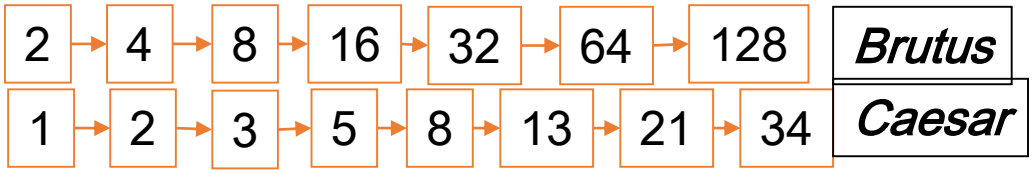
# The Big Picture

- Content Processing
  - Build Term Document Matrix or Build Inverted Index
- Query Handling
  - Boolean AND or Intersect the Posting Lists (called merging process)



# The Merge

- Walk through the two postings simultaneously
  - Clue: Use two pointers




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

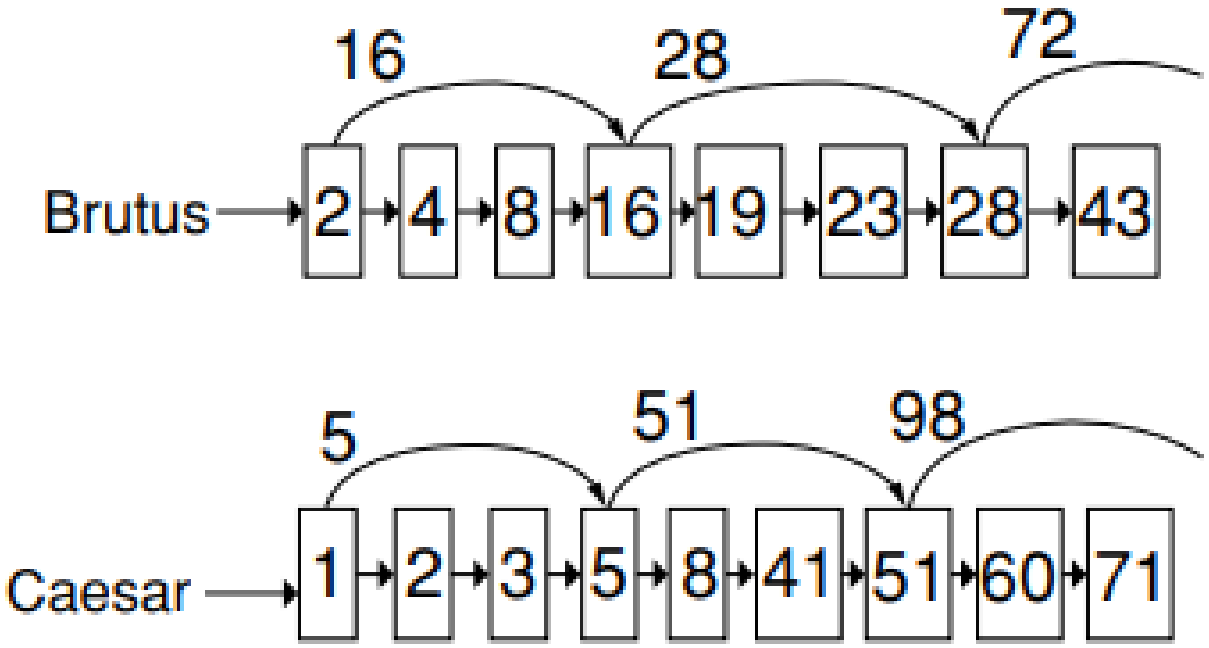
**Can we do better?**

Inspired from multiple index idea of DBMS

# A Better Approach

**Revisiting**
**Multiple Indices of DBMS**

# Skip Pointers



# Indexing

# How to Index?


Captain Haddock

# What is a Document?

- Some systems store a single email in multiple files. Is each file a document?
- Some files can contain multiple documents (as in XML, Zip).

# Tokens Vs. Terms

- Tokens
  - Input: Friends, Romans, Countrymen, lend me your ears.
  - Output:

| Friends | Romans | Countrymen | lend | me | your | ears |
|---|---|---|---|---|---|---|

  - Sequence of characters → Semantic Units
    - Throw away “less important” parts (like punctuation)
- Terms
  - Indexed by the IR system

# Quiz

- Tokenize O’Neil Can’t study.

| O’Neil | Can’t | study |
|---|---|---|

What if we tokenize based on ‘ ?

| O | Neil | Can | t | study |
|---|---|---|---|---|

| O | Neil | Can’t | study |
|---|---|---|---|

# How to Index?

Billions of blistering barnacles! **Decide what a document is**. **Know how to tokenize it.** Take any document, tokenize, sort, prepare posting lists. That is all!

# Which Tokens to Index?

- Which tokens are interesting?

*It is difficult to imagine living without search engines*

**Stop Word Removal**

*difficult imagine living search engines*

- it, is, to, without are “Stop Words” for us here.

# How to Index?


Billions of blue blistering barnacles! **Decide what a document is**. **Know how to tokenize it. Prepare a stop words list.** Take any document, tokenize, remove stop words, sort, prepare posting lists. That is all!

# Token Normalization

- Equivalence Classes
  - (case folding) window, windows, Windows, Window → window
  - anti-theft, antitheft, anti theft → antitheft
  - color, colour → color

Billions of bilious blue blistering barnacles! **Decide what a document is**. **Know how to tokenize it. Prepare a stop words list.** Take any document, tokenize, normalize, remove stop words, sort, prepare posting lists. That is all!

# Stemming and Lemmatization

- Stemming (chop the ends)
  - going → go
- Lemmatization
  - Return the dictionary form of the root word (lemma)
    - saw → see.
- More Examples
  - am, are, is → be
  - car, cars, car's, cars' → car
  - democrat, democratic, democracy, democratization → democrat

# How to Index?


Billions of bilious blue blistering barnacles! **Decide what a document is**. **Know how to tokenize it. Prepare a stop words list.** Take any document, tokenize, normalize, remove stop words, stem/lemmatize, sort, prepare posting lists. That is all!

# Quiz

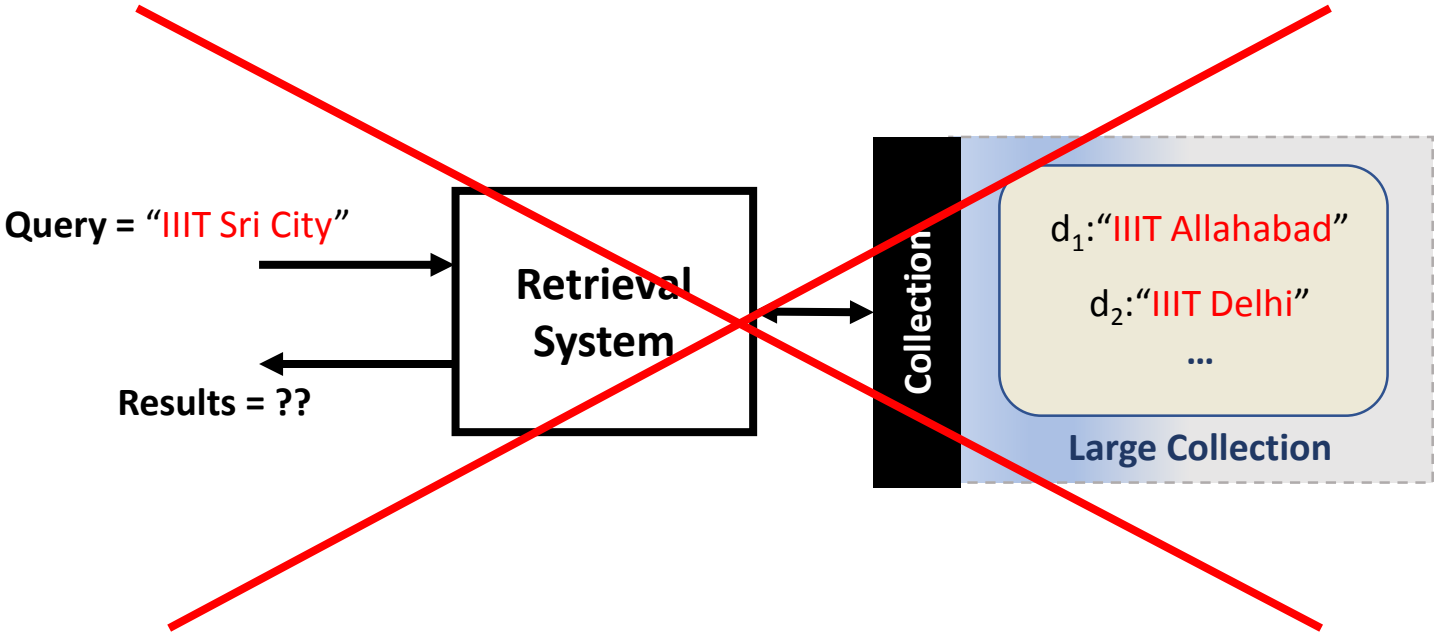
- Tokenize the following
  - **반갑습니다**
    - (Korean for "*Nice to meet you*")
  - **Bundesausbildungsförderungsgesetz**
    - A German compound word for "*Federal Education and Training Act*")
- Can you think of a case where splitting with white space is bad?
  - Los Angeles, Sri City

# Project Ideas 

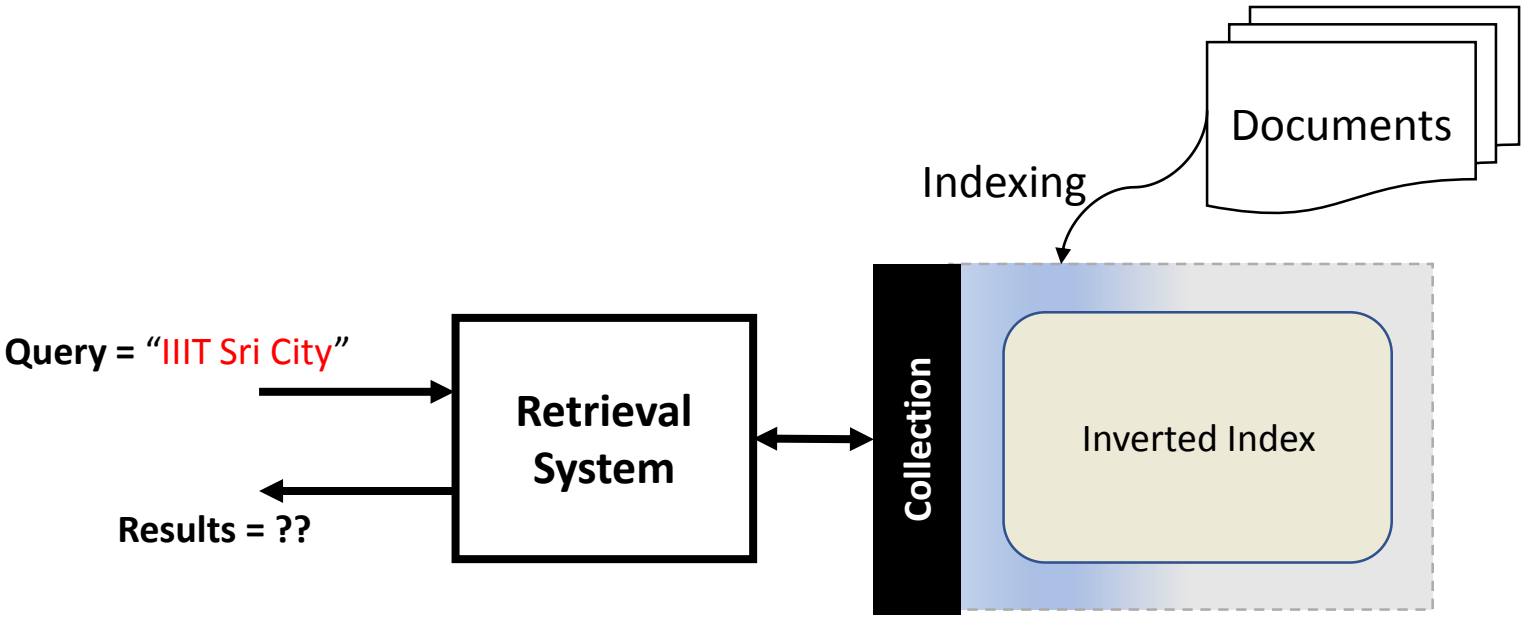
- Develop a **Compound Splitter/Stop Words/Stemmer** for variable names in source code.
  - Dataset: GitHub or StackOverflow
- Develop a **Search Engine** (using Apache Lucene) with and without your **Compound Splitter**. Show the difference.

Remember, all projects must implement a search engine.

# The Big Picture



# The Big Picture

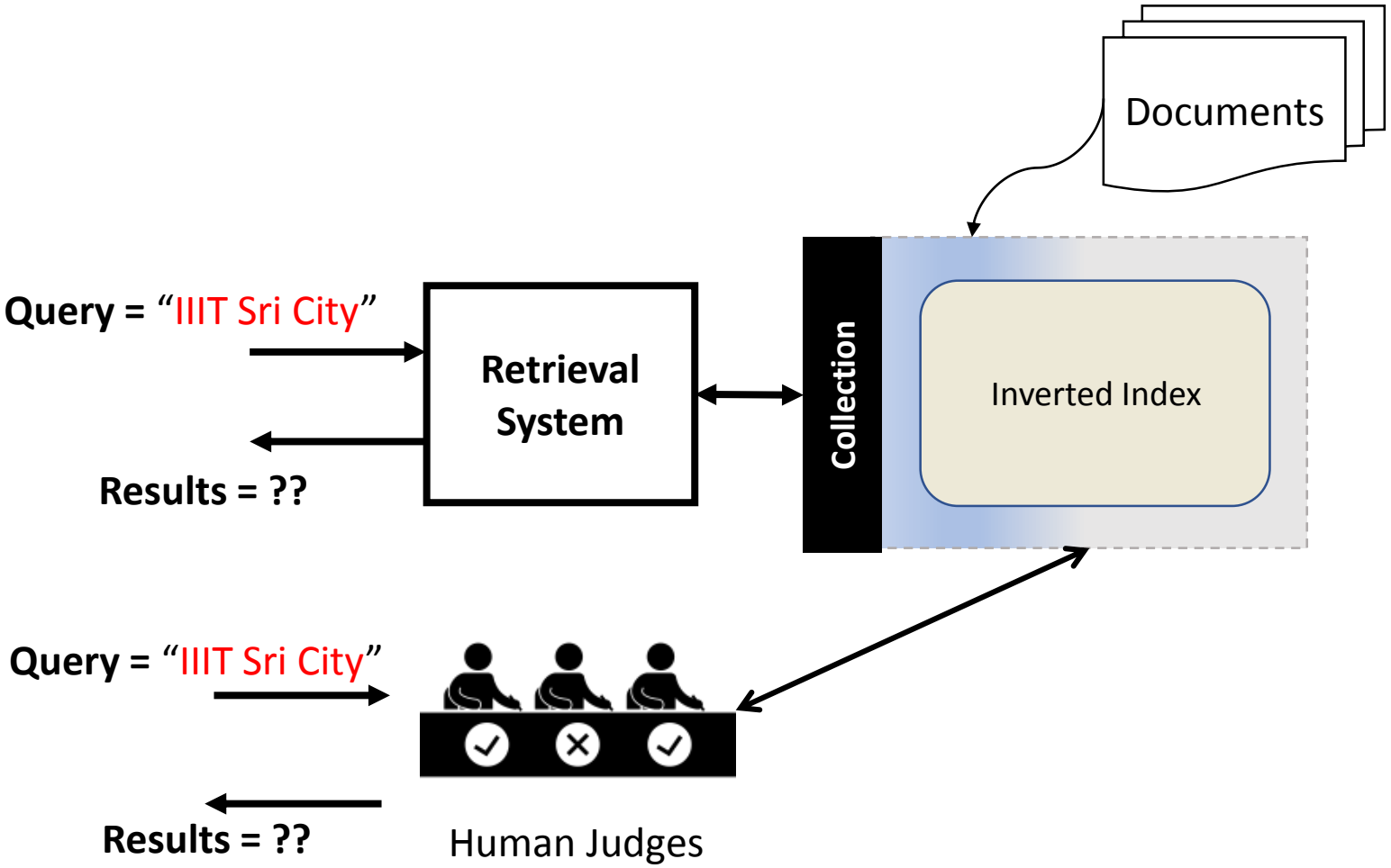


# Evaluation

# How Good is Our System?

- A **collection** having the following contents
  - d1: IIIT ALLAHABAD
  - d2: IIIT DELHI
  - d3: IIIT GUWAHATI
  - d4: ISI
  - d5: IIIT SRI CITY
  - d6: KREA SRI CITY
- **Query** is
  - SRI CITY
- **Result** is
  - IIIT SRI CITY
  - KREA SRI CITY

# Evaluation


Documents
Collection
Inverted Index
Query = "IIIT Sri City"
Retrieval System
Results = ??
Query = "IIIT Sri City"
Results = ??
Human Judges

# How Good is Our System?

- A **collection** having the following contents
  - d1: IIIT ALLAHABAD
  - d2: IIIT DELHI
  - d3: IIIT GUWAHATI
  - d4: ISI
  - d5: IIIT SRI CITY
  - d6: KREA SRI CITY
- **Query** is
  - IIIT
- **Result** is
  - IIIT SRI CITY
  - KREA SRI CITY

# Objective

We want all relevant documents and only relevant documents

# Relevance

- How many **relevant** documents?
  - Four (IIIT SRI CITY, IIIT ALLAHABAD, IIIT DELHI, IIIT GUWAHATI)
- How many **retrieved** documents?
  - Two (IIIT SRI CITY, KREA SRI CITY)

How to quantify the “goodness” of our system?

# Terminology

- Documents we see in results are "**positive**"
  - Positive
    - + IIIT SRI CITY,
    - + KREA SRI CITY
  - Negative
    - - IIIT ALLAHABAD
    - - IIIT DELHI
    - - IIIT GUWAHATI
    - - ISI

# Terminology

- Documents that we correctly classify are "**true**"
  - Positive
    - + IIIT SRI CITY (true)
    - + KREA SRI CITY
  - Negative
    - - IIIT ALLAHABAD
    - - IIIT DELHI
    - - IIIT GUWAHATI
    - - ISI (true)

**Here, query is "IIIT"**

# Quiz

- All retrieved results =
  1. tp + fp
  2. tp + fn
  3. tn + fp
  4. tn + fn

**Legend**
tp = true positive
tn = true negative
fp = false positive
fn = false negative

# Quiz

- All retrieved results =
  1. tp + fp
  2. tp + fn
  3. tn + fp
  4. tn + fn

**Legend**

tp = true positive

tn = true negative

fp = false positive

fn = false negative

# Quiz

- All relevant results =
  1. tp + fp
  2. tp + fn
  3. tn + fp
  4. tn + fn

**Legend**
tp = true positive
tn = true negative
fp = false positive
fn = false negative

# Quiz

- All relevant results =
  1. tp + fp
  2. tp + fn
  3. tn + fp
  4. tn + fn

**Legend**
tp = true positive
tn = true negative
fp = false positive
fn = false negative

# You have 100% Precision

- Everything you retrieved were relevant.
  - tp + fp = tp
  - fp = 0

# You have 100% Recall when

- You retrieved everything that were relevant. (Note: You could have retrieved more).
  - fn = 0
  - tp = all relevant documents

# Quiz

- The following list of Rs and Ns represents relevant (R) and nonrelevant (N) returned documents in a list of 20 documents retrieved in response to a query from a collection of 10,000 documents. This list shows 6 relevant documents. Assume that there are 8 relevant documents in total in the collection. Calculate Precision and Recall.

R R N NNNNN R N R N NN R N NNN R

# Precision and Recall

- Precision = 6/20
- Recall = 6/8

# Precision and Recall

**Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)

**Recall**: fraction of relevant docs that are retrieved

= P(retrieved|relevant)

| | Relevant | Nonrelevant |
|---|---|---|
| Retrieved | tp | fp |
| Not Retrieved | fn | tn |

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

# Exercise

**Suppose, a document is relevant only if both judges agree that it is relevant. Assume (0 = nonrelevant, 1 = relevant). What is the Precision and Recall? *Our system retrieves 5 documents in total.**

Query = “Taj”

| Document ID | Judge 1 | Judge 2 | Our System |
|---|---|---|---|
| d1 = Bru | 0 | 0 | 1 |
| d2 = 3Roses | 0 | 0 | 0 |
| d3 = Taj | 1 | 1 | 1 |
| d4 = Taj Tea | 1 | 1 | 0 |
| d5 = Taj Mahal | 1 | 0 | 0 |

# Answer

- Precision = 1/5
- Recall = 1/2