Information Retrieval

M. Narasimha Murty
Professor, Dept. of CSA
Indian Institute of Science
Bangalore-560 012
mnm@csa.iisc.ernet.in

September 5 ,
2015

Information Retrieval

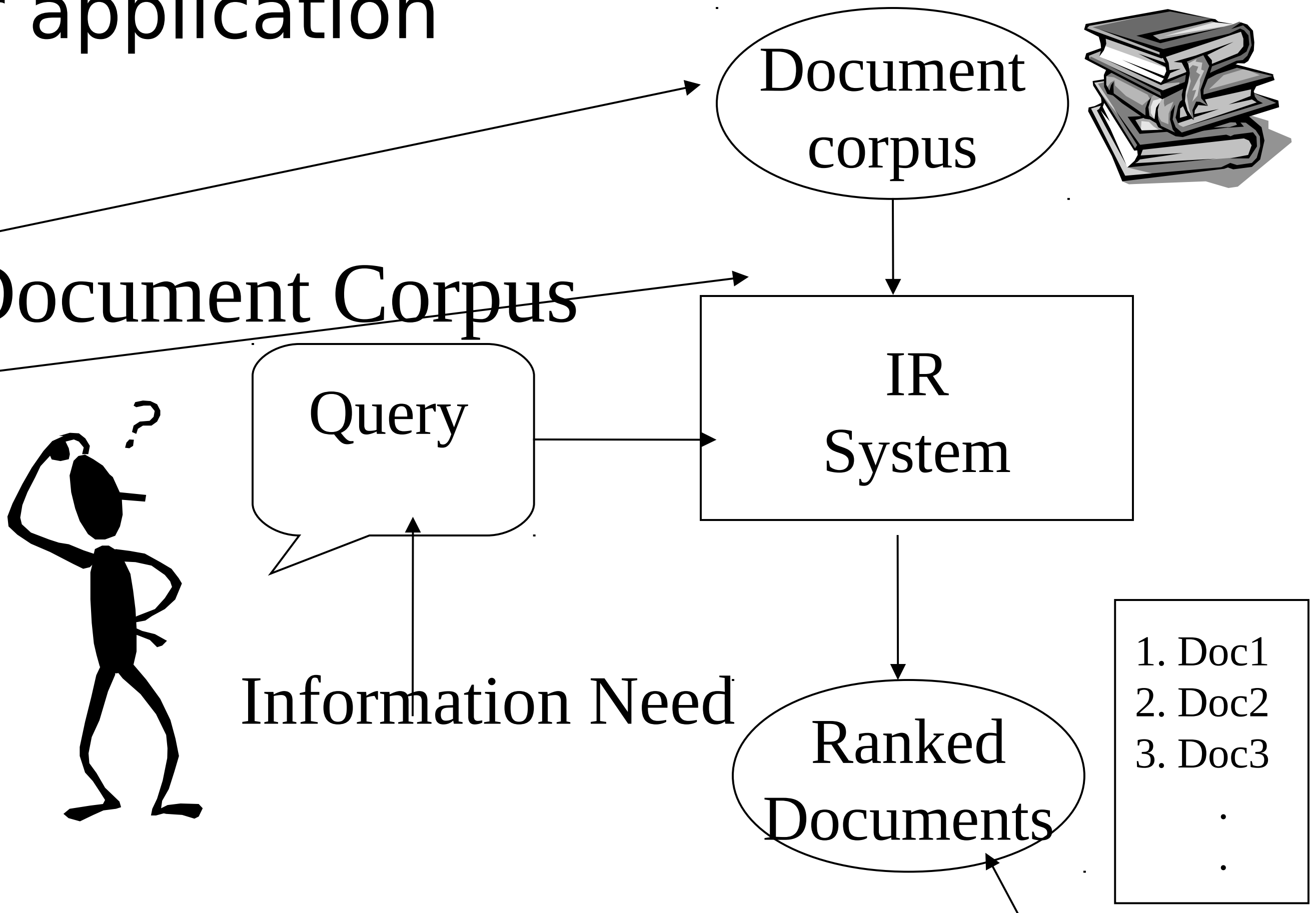
"Information Retrieval (IR) is
finding material (usually *documents*)
of *semi-structured* nature (usually *text*)
that satisfies an *information need*
from within *large collections* (usually *stored on computers*)."

Reference: Manning, Raghavan, Schutze, *Information Retrieval*

IR System

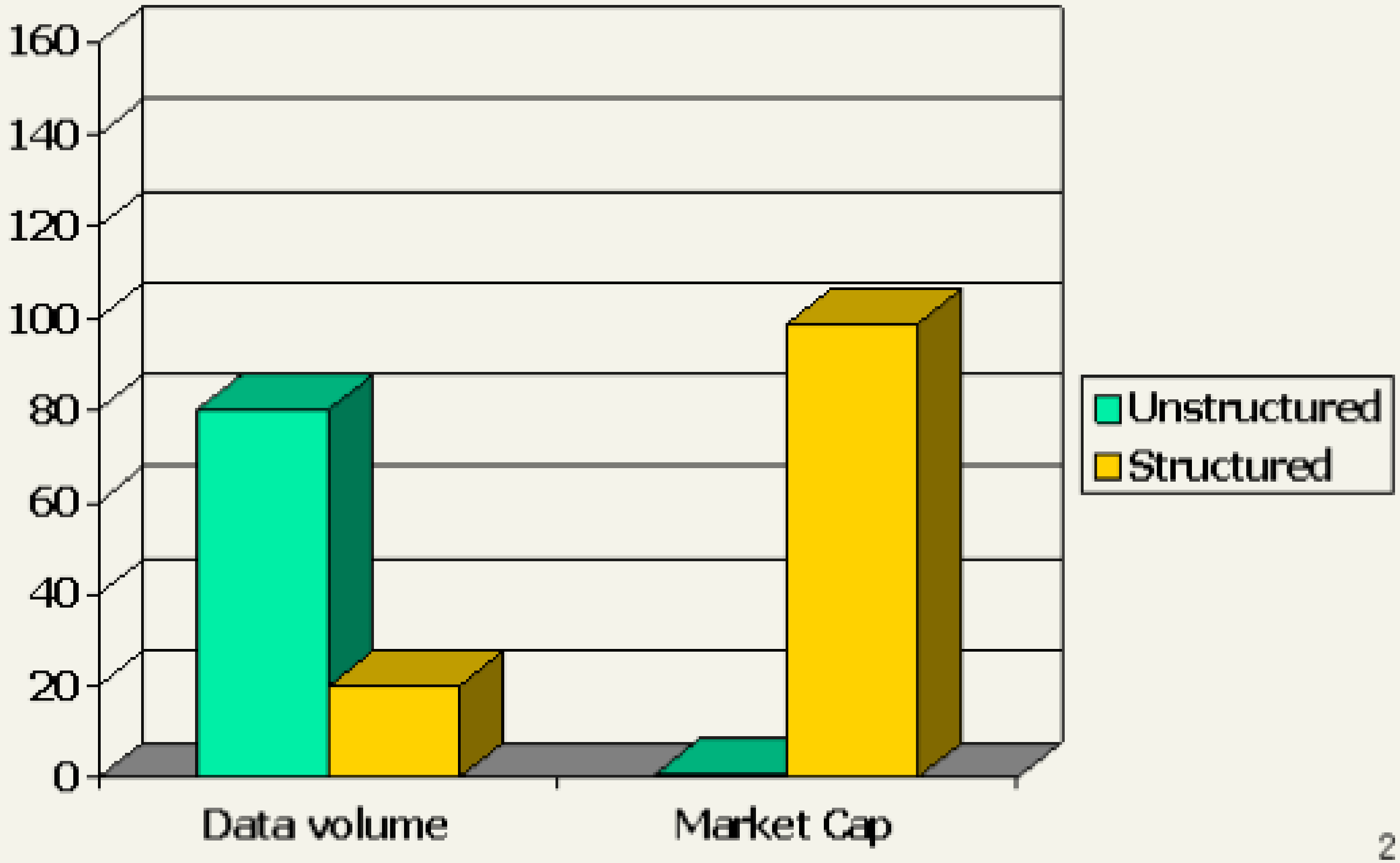
Searching the World Wide Web
is a popular application

from *large* Document Corpus
efficiently

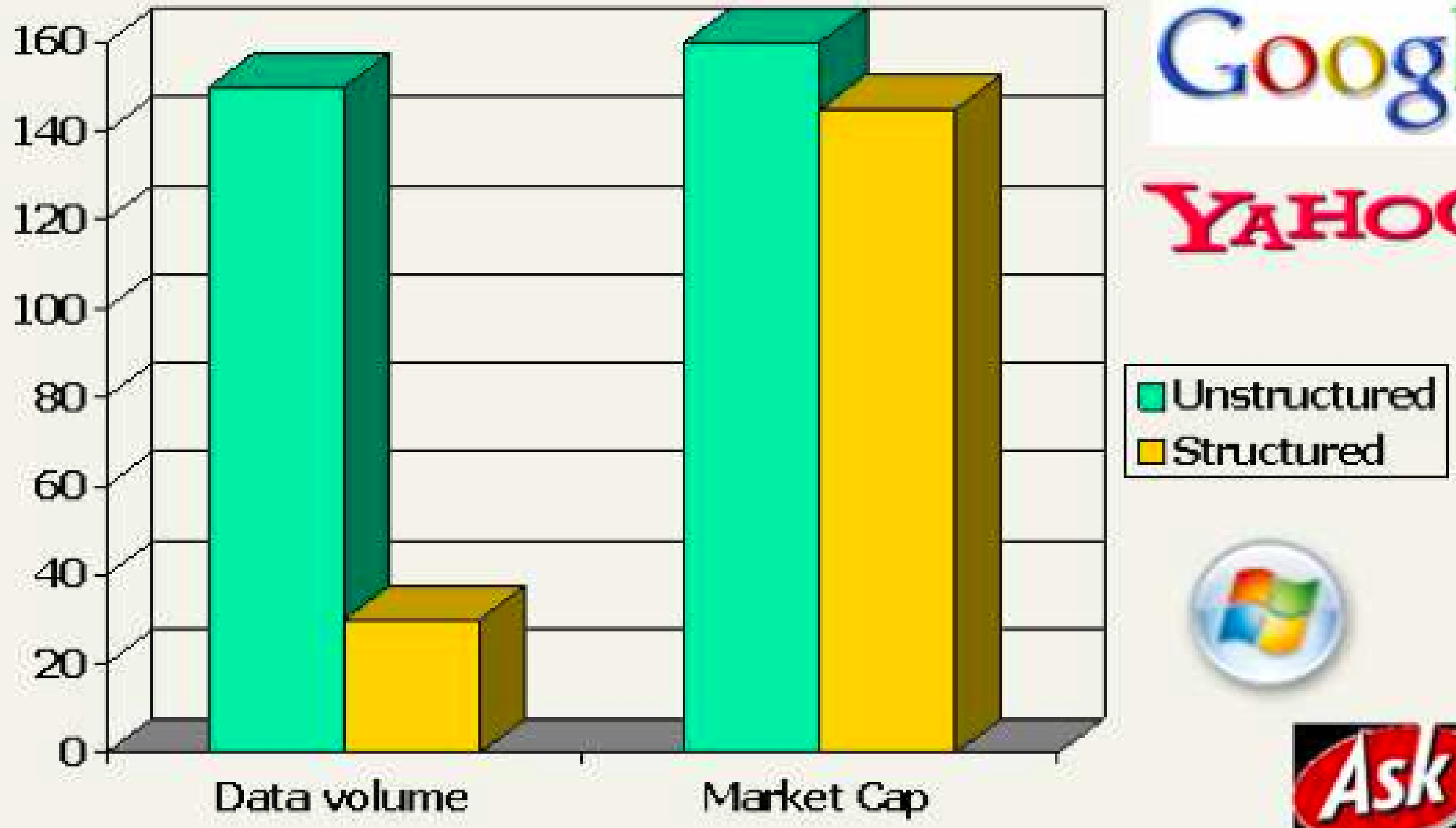


retrieving *relevant* documents to a query; ranking

Unstructured (text) vs. structured (database) data in 1996



Unstructured (text) vs. structured (database) data in 2006



Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The seach engine returns all documents that satisfy the
  Boolean expression

Does Google use the Boolean

Unstructured data in 1650: Shakespeare

Unstructured data in 1650

- 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 grep not the solution?
- Slow (for large collections)
- grep is line-oriented, IR is document-oriented
- "NOT CALPURNIA" is non-trivial
- Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible

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
doesn’t occur in *The tempest*.

1
0

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 NOT CALPURNIA
- Complement the vector of CALPURNIA
- Do a (bitwise) and on the three vectors
- 110100 AND 110111 AND 101111 = 100100

0/1 vector for BRUTUS

	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
. . .						
result:	1	0	0	1	0	0

Answers to query

Anthony 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.

Typical IR Task

- Given:
  - A corpus of textual natural-language documents
  - A user query in the form of a textual string
- Find:
  - A ranked set of documents that are relevant to the query
- Large Collection: (For ex., over 10 years)
  - Papers of a researcher – 100 MB
  - E-mail archive – 1 GB
  - Texts of all books in a small library – 100 GB
  - Complete text of web – hundreds of

What is a Document?

- Web page
- News paper article
- Acad. publication
- Company report
- Research grant application
- Manual page
- Encyclopedia
- Images (video)
- Speech records
- Bank transaction slip
- Multimedia record
- Historical record
- Electronic mail
- Court transcript
- Health record
- Legal record
- Fingerprint
- Software
  - Code
  - Bug reports

Collection Size

- Consider *N* = 1 million documents, each with about 1000 words.
- Avg 8 bytes/word including spaces/punctuation
  - 8GB of data in the documents
  - 1 M x 1000 x 8 = 8000000000 B = 8GB
- Say there are *M* = 500K *distinct* terms among these.

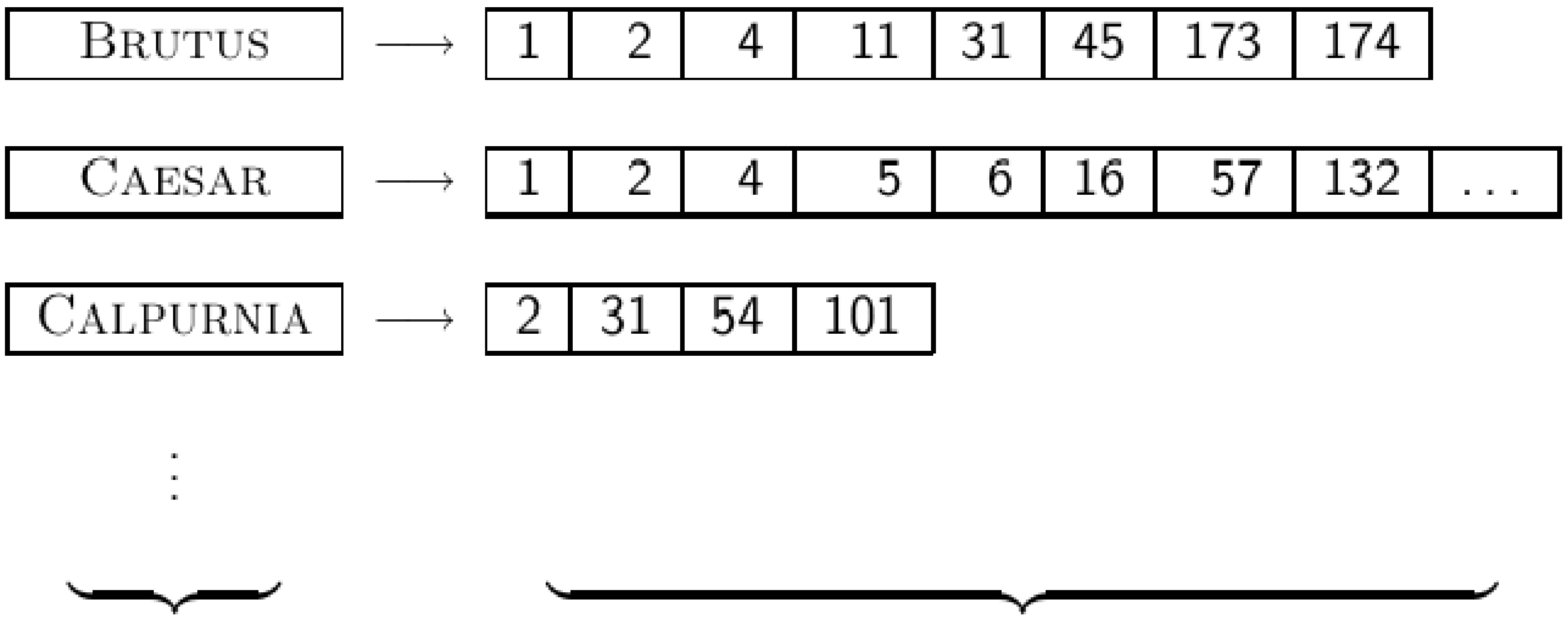
Term-Document Matrix Size

- 500K x 1M matrix has half-a-trillion 0’s and 1’s (0.5 x 10^12)
- But it has no more than one billion 1’s.
    - matrix is extremely sparse
    - number of 1’s = 1M x 1K = 10^9
    - number of 0’s = 499 x 10^9
- What is a better representation?
    - We only record the 1 positions

1
7

Inverted Index

For each term *t*, we store a list of all documents that cc

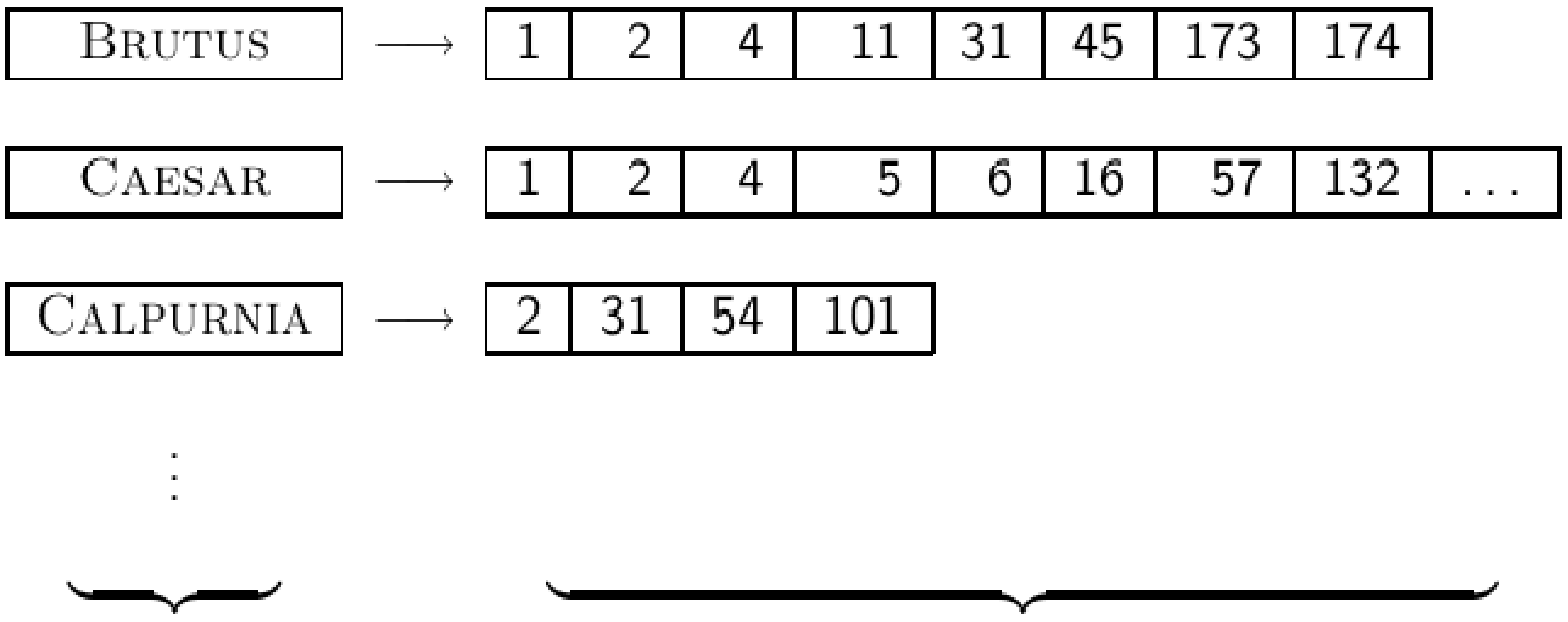


dictionary postings

1
3

Inverted Index

For each term *t*, we store a list of all documents that
cc

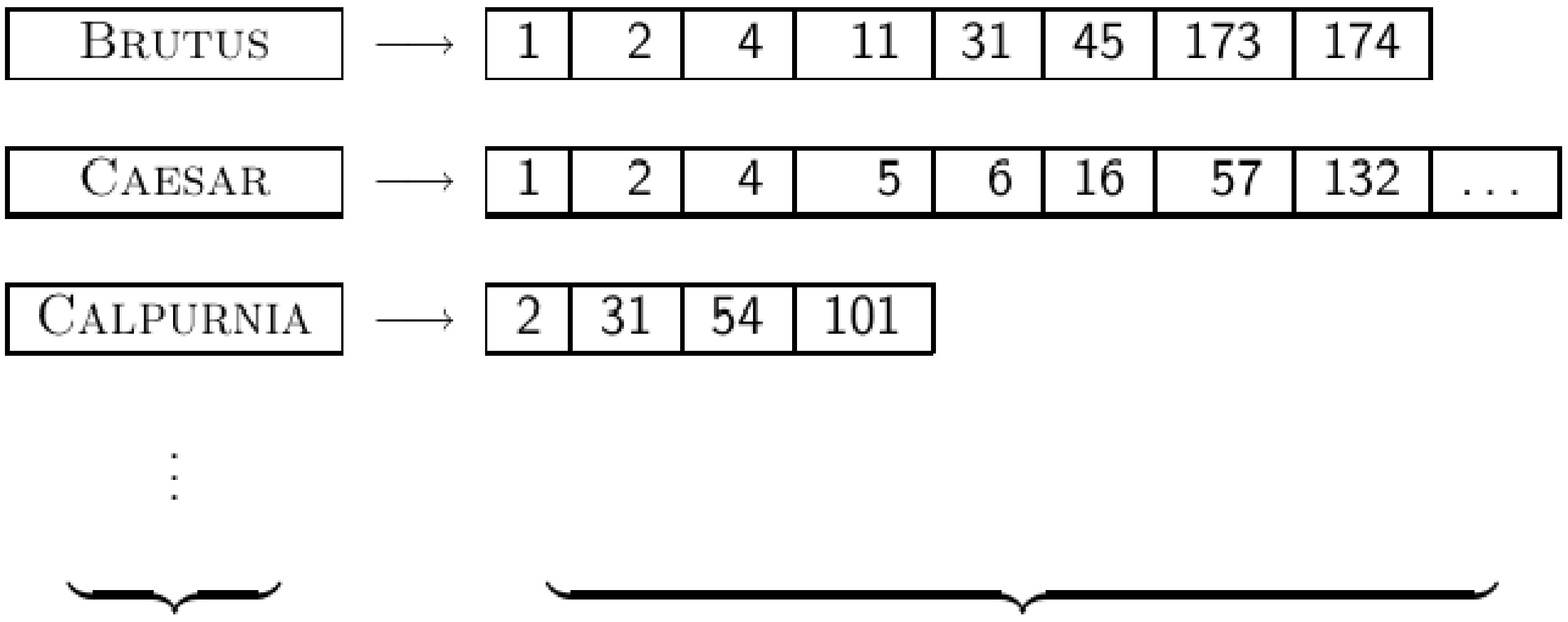


dictionary postings

1
9

Inverted Index

For each term *t*, we store a list of all documents that cc



dictionary

postings

Inverted index construction

❶Collect the documents to be indexed:

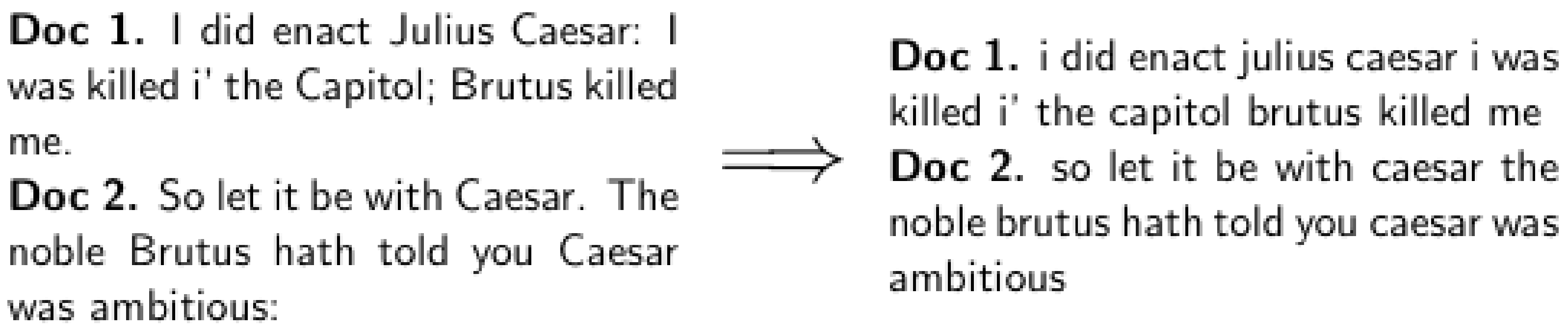
$\boxed{\text{Friends, Romans, countrymen.}}$ $\boxed{\text{So let it be with Caesar}}$ . . .

❷Tokenize the text, turning each document into a list of $\boxed{\text{Friends}}$ $\boxed{\text{Romans}}$ $\boxed{\text{countrymen}}$ $\boxed{\text{So}}$ . . .

❸ Do linguistic preprocessing, producing a list of nor[illegible] which are th $\boxed{\text{friend}}$ $\boxed{\text{roman}}$ erms: $\boxed{\text{countryman}}$ $\boxed{\text{so}}$ . . .

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

Tokenizing and preprocessing

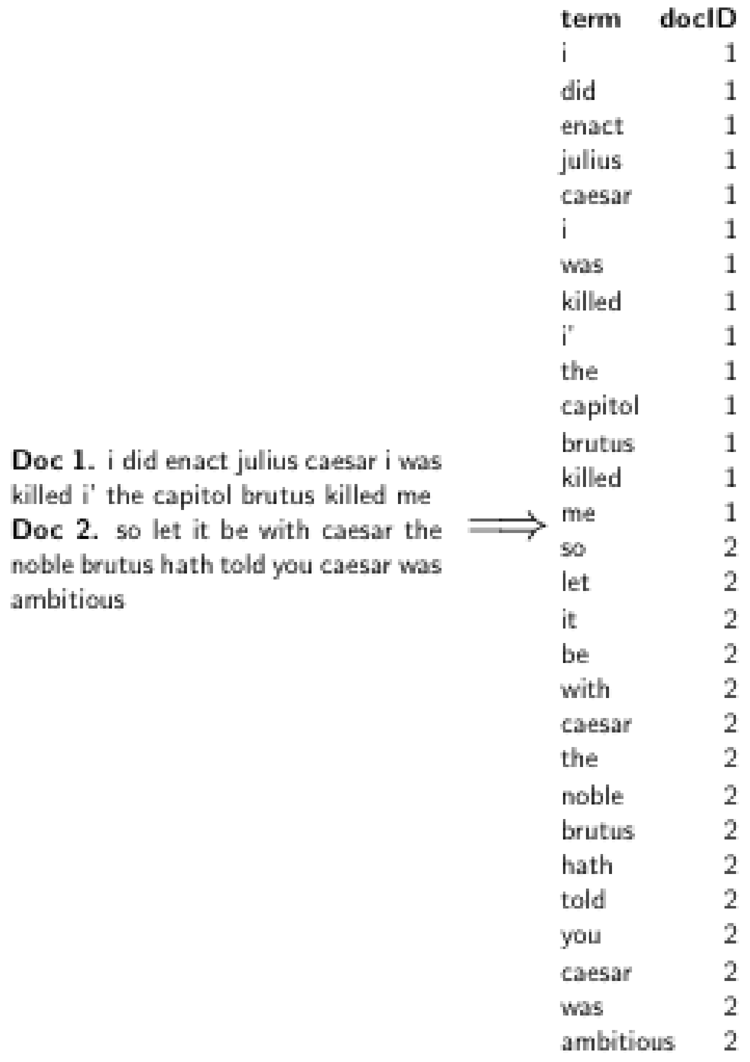


Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

$\Longrightarrow$

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

Generate posting

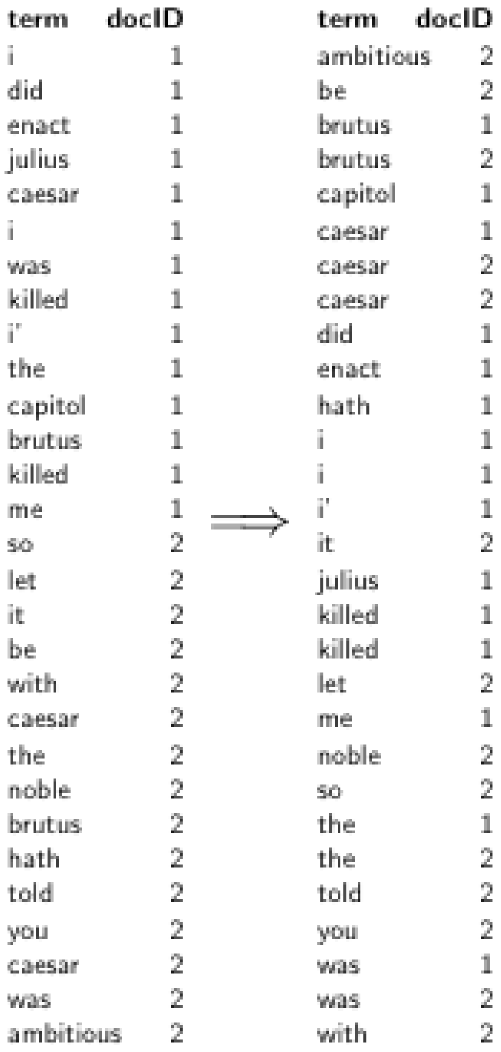


Doc 1. i did enact julius caesar i was killed i' the capitol brutus killed me
Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious

$\Longrightarrow$

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

Sort postings

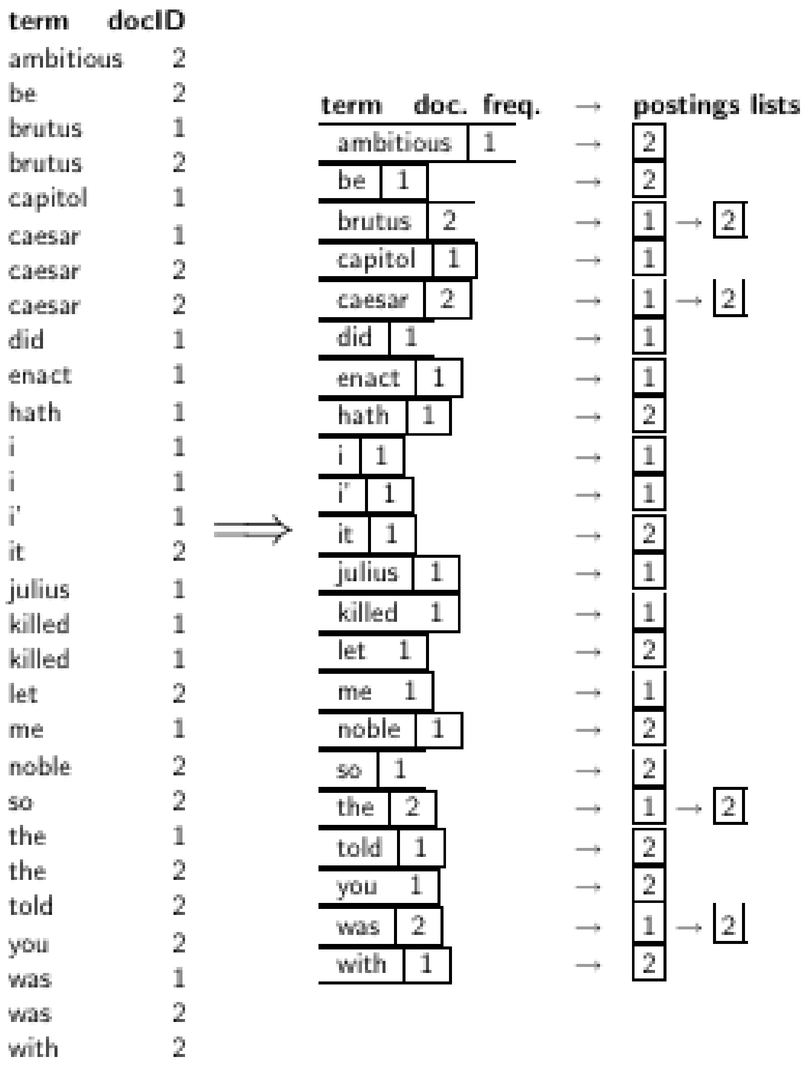


term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

$\Longrightarrow$

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

Create postings lists, determine document frequency

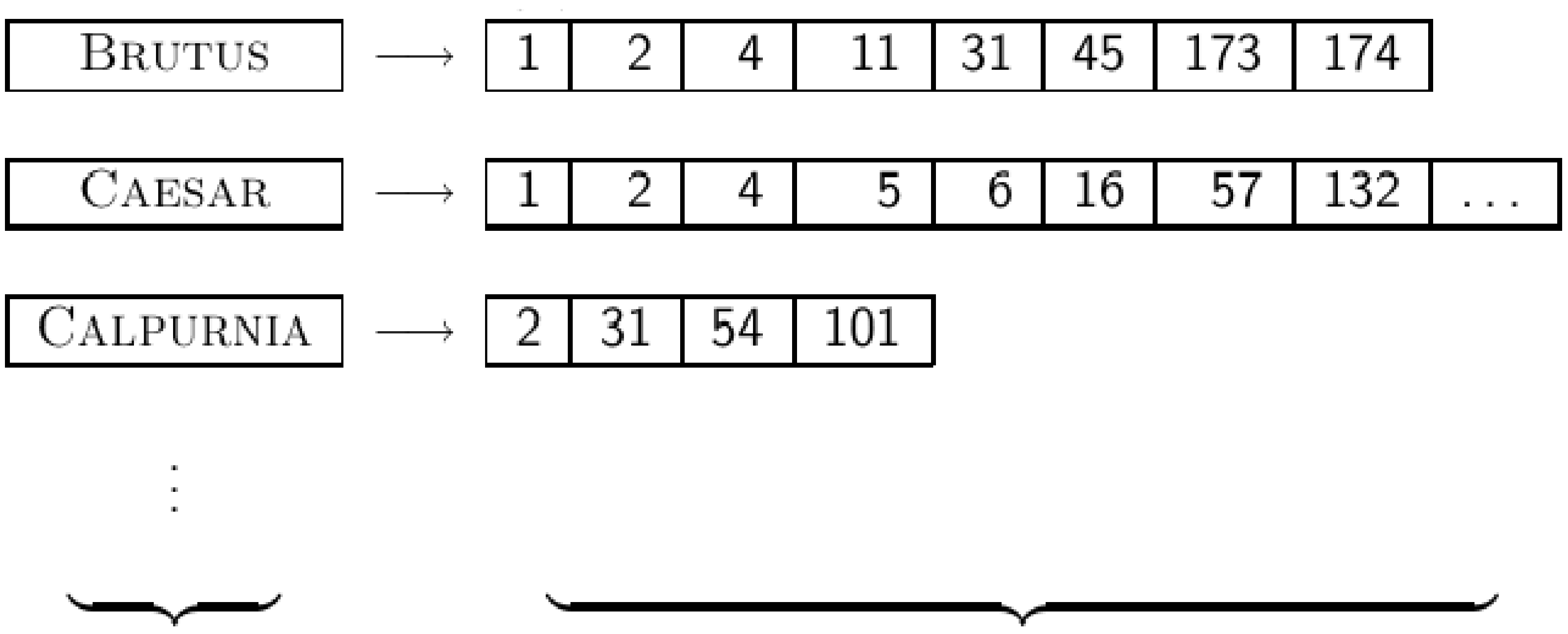


term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

$\Longrightarrow$

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

Split the result into dictionary and postings file



dictionary postings

Retrieval of Books

- Which books contain the words ***cluster*** *AND* ***class*** but *NOT* ***grammar***?
- One could grep all the books for ***cluster*** and ***class,*** then remove books containing ***grammar!***
- The problems are:
  - It is slow (for large document and term sets)
  - *NOT* ***grammar*** is non-trivial
  - Other operations (e.g., find the word ***pattern*** near ***recognition***) not feasible

Term-document Matrix

	Pattern Recognition& MachineLearning C. Bishop	Pattern Classification Duda, Hart, Stork	Fundamental Algorithms DE Knuth	Mining of Massive Datasets A. Rajaraman, JD ullman	Elements of Statistical Learning Hastie, Tibshirani, Friedman
Class	1	1	0	1	1
Cluster	1	1	0	1	1
Grammar	0	1	0	0	0
Graph	1	1	1	1	0
Learn	1	1	0	1	1

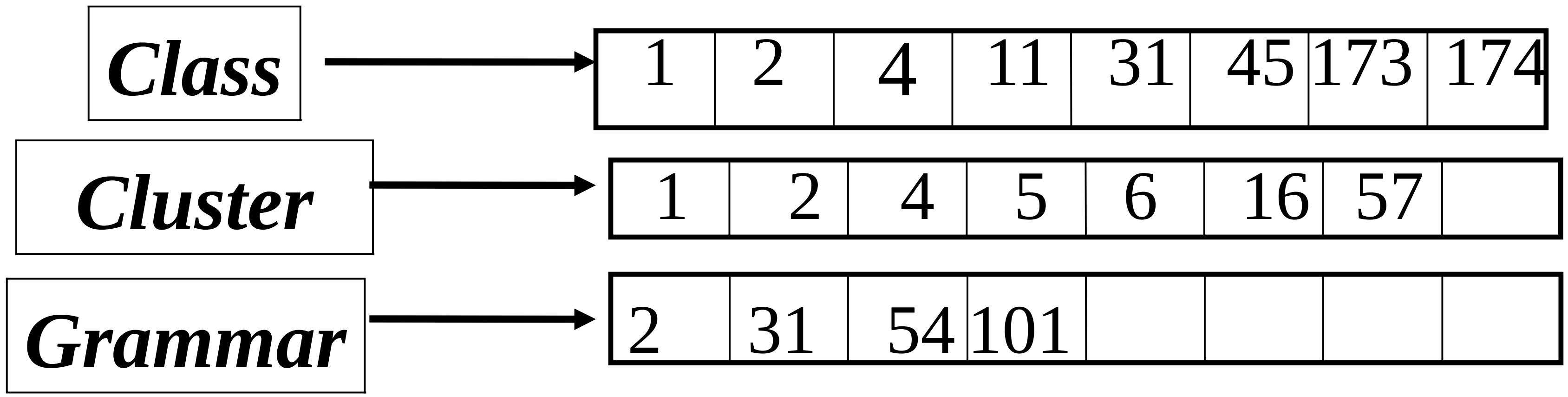
Class *AND* ***Cluster*** *AND* ***Grammar:***

Pattern Classification: DHS

1 if book contains word, 0 otherwise

Inverted index

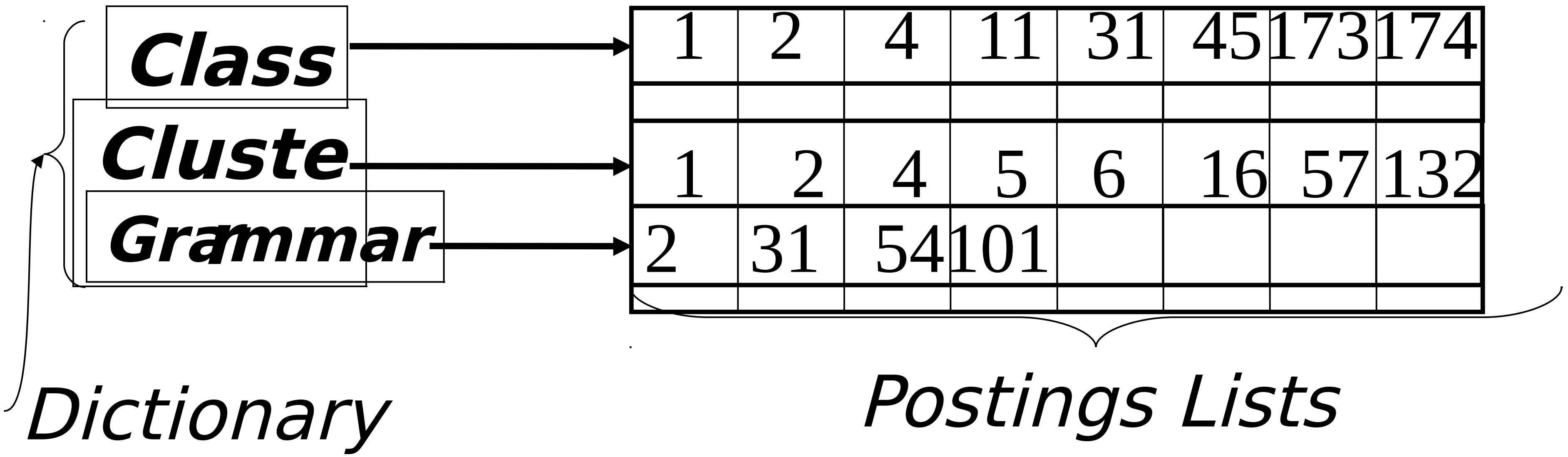
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number
- Fixed-size array may not be right!



What happens if the word ***Cluster*** is added to document 14 or document 31 is deleted? It happens with web pages; they change periodically!

Inverted index

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

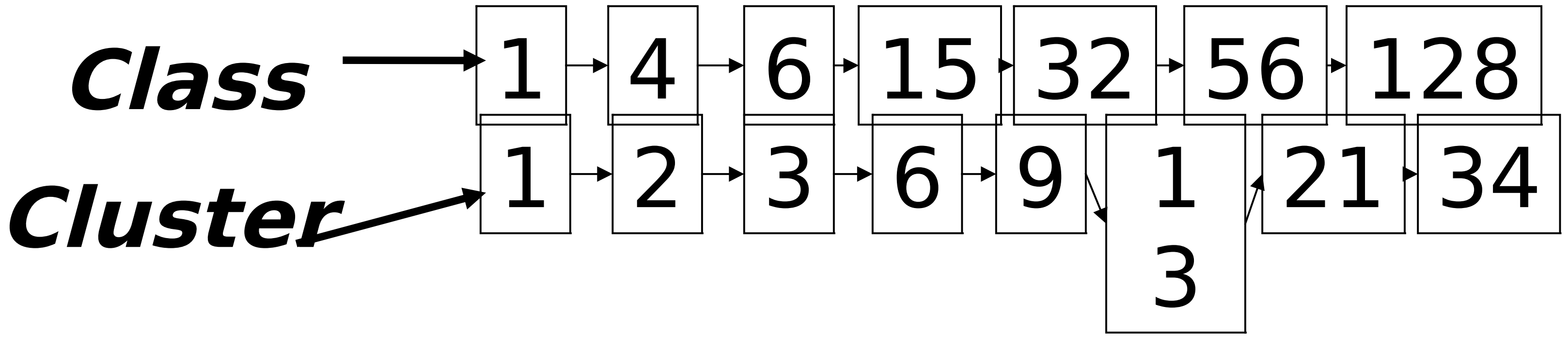


Query processing: Merging

- Let the query be:

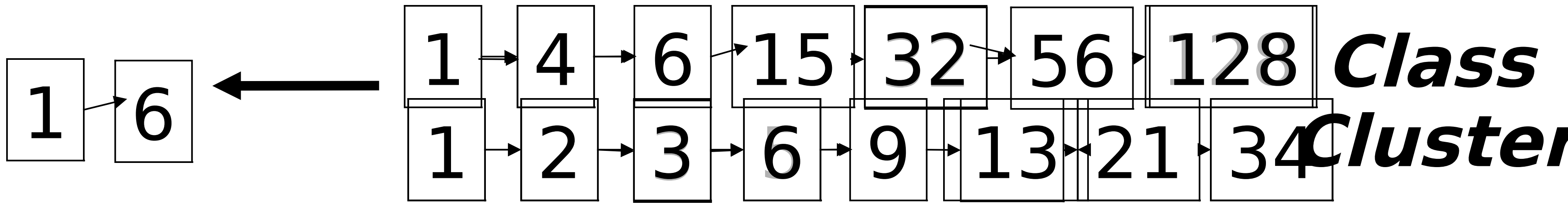
  Class *AND* ***Cluster***

  - Locate ***Class*** in the Dictionary;
    - Retrieve its postings.
  - Locate ***Cluster*** in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



Merging Sorted Posting Lists

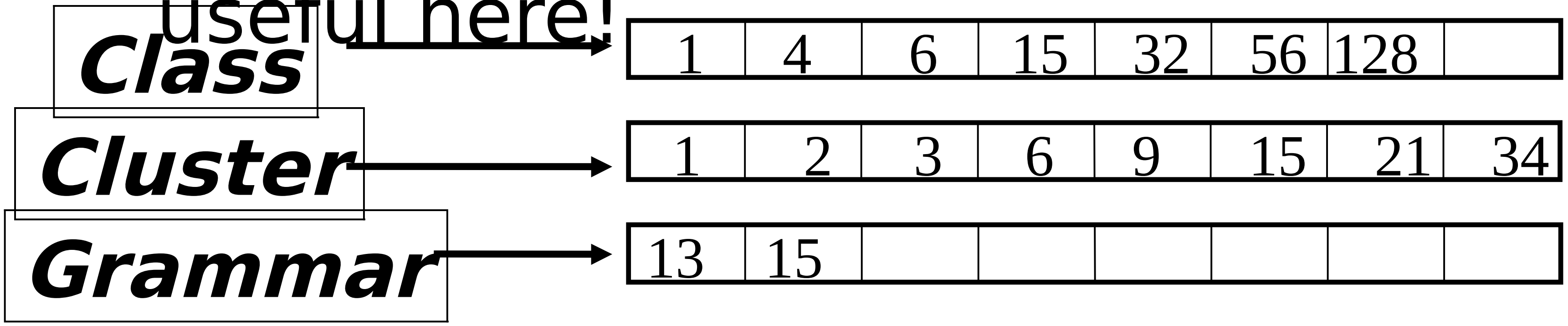
- Go through the two sorted postings lists; so, merging can be efficiently performed.



If ***Class*** has m elements and ***Cluster*** has n elements in their postings, then merge requires O(m+n) comparisons. Because lists are sorted by document numbers. This type of merging is part of merge sort.

Example: Efficient Query Processing

- Process in the order of increasing freq:
  - *start with smallest set, then keep reducing further*. Grammar: 2; Class: 7; Cluster: 8
  - Document frequency (number of documents in which a term occurs) is useful here!



Execute the query: (***Grammar*** *AND* ***Class)*** *AND* ***Cluster***.

More general optimization

- Example query: (MADDING OR CROWD) and (IGNOBLE OR STRIFE)
- Get frequencies for all terms
- Estimate the size of each <u>or</u> by the sum of its frequencies (conservative)
- Process in increasing order of or sizes

Text Processing

- Definition:
  - given a text string T and a pattern string P, find the pattern inside the text
    - T: “the rain in spain stays mainly on the plain”
    - P: “n th”
- Applications:
  - text editors, Web search engines

Pattern Matching - Brute Force Algorithm

- Check each position in the text T to see if the pattern P starts in that position

T: a n d r e w

P: r e w

T: a n d r e w

P: r e w

P moves 1 char at a time through T

Knuth Morris Pratt, Boyer-Moore Algorithms

The Boyer-Moore Algorithm

- Start from the right most position in “P” and corresponding location in “T”.
- Construct a shift table from P and use it.

T: c o n u n d r u m

P: d r u m   u and m mismatch; shift by 1

d r u m   n and m mismatch, n is not in P

d r u m   Success

Knowledge in Information Retrieval

Knowledge is used in

- Collecting the documents
- Representing the query and documents
- Indexing documents
- Refining the query
- Finding similarity and in ranking
- Grouping and classification of documents

Keyword Search

- Simplest notion of relevance is that the query string appears as it is in the document
- Slightly less strict notion is that the words in the query appear frequently in the document, in any order (Bag of words!)
- Example:
  1. The good old teacher teaches several courses
  2. In the big old college in the big old town
  3. The college in the town likes the good old teacher
  4. Where the old teacher never did fail
  5. The good teacher teaches in the evenings
  6. And the students like his lecture notes

Keyword Search (continued)

- With casefolding, the vocabulary is:

  and (And) big college courses did evenings fail good his in (In) lecture like likes never notes old several students teacher teaches the (The) town where (Where)

- With stemming: (lossy, yet useful compression)

  and big college course did evening fail good his in lecture like (likes) never note (notes) old several student (students) teach (teacher, teaches) the town where

- Stopping then reduces the vocabulary:

  big college course evening fail good lecture like note old several student teach town

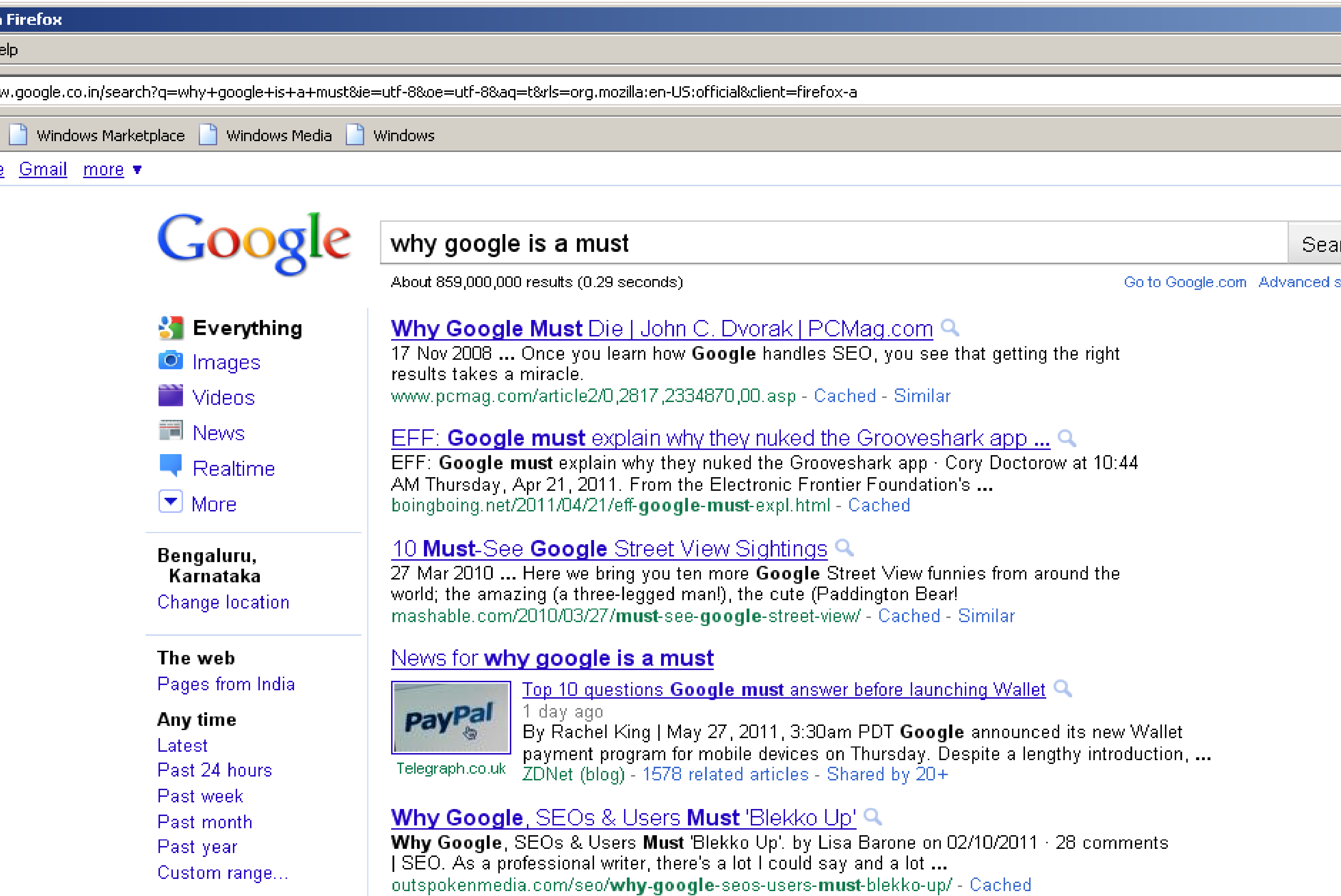
Inverted Index

TERM	INVERTED LIST
big	<2,2>
college	<2,1> <3,1>
course	<1,1>
evening	<5,1>
fail	<4,1>
good	<1,1> <3,1> <5,1>
lecture	<6,1>
like	<3,1> <6,1>
note	<6,1>
old	<1,1> <2,2> <3,1> <4,1> <5,1>
several	<1,1>
student	<6,1>
teach	<1,2> <3,1> <4,1><5,2>
town	<2,1> <3,1>

Boolean Model

- A document is represented as a set of keywords.
- Queries are Boolean expressions of keywords, connected by AND, OR, and NOT.
- Output: Document is relevant or not. No partial matches or ranking.
- For example, ***old college*** – docs 2 and 3
  2. In the big **old college** in the big old town
  3. The **college** in the town likes the good **old** teacher
- *Bag of Words* Paradigm!
- Order is not important, frequency is!

Simple Model



About 859,000,000 results (0.29 seconds)

Bengaluru, Karnataka

The web

Any time

Why Google Must Die | John C. Dvorak | PCMag.com
17 Nov 2008 ... Once you learn how **Google** handles SEO, you see that getting the right results takes a miracle.
www.pcmag.com/article2/0,2817,2334870,00.asp - Cached - Similar

EFF: **Google must** explain why they nuked the Grooveshark app ...
EFF: **Google must** explain why they nuked the Grooveshark app · Cory Doctorow at 10:44 AM Thursday, Apr 21, 2011. From the Electronic Frontier Foundation's ...
boingboing.net/2011/04/21/eff-**google**-**must**-expl.html - Cached

10 **Must**-See **Google** Street View Sightings
27 Mar 2010 ... Here we bring you ten more **Google** Street View funnies from around the world; the amazing (a three-legged man!), the cute (Paddington Bear!
mashable.com/2010/03/27/**must**-see-**google**-street-view/ - Cached - Similar

News for **why google is a must**

Telegraph.co.uk

Top 10 questions **Google must** answer before launching Wallet
1 day ago
By Rachel King | May 27, 2011, 3:30am PDT **Google** announced its new Wallet payment program for mobile devices on Thursday. Despite a lengthy introduction, ...
ZDNet (blog) - 1578 related articles - Shared by 20+

Why Google, SEOs & Users **Must** 'Blekko Up'
Why Google, SEOs & Users **Must** 'Blekko Up'. by Lisa Barone on 02/10/2011 · 28 comments | SEO. As a professional writer, there's a lot I could say and a lot ...
outspokenmedia.com/seo/**why-google**-seos-users-**must**-blekko-up/ - Cached

Type/Term distinction

- Token – an instance of a word or term occurring in a document
- Term – an equivalence class of tokens
- *In June, the dog likes to chase the cat in the barn.*
- 12 word tokens, 9 word types

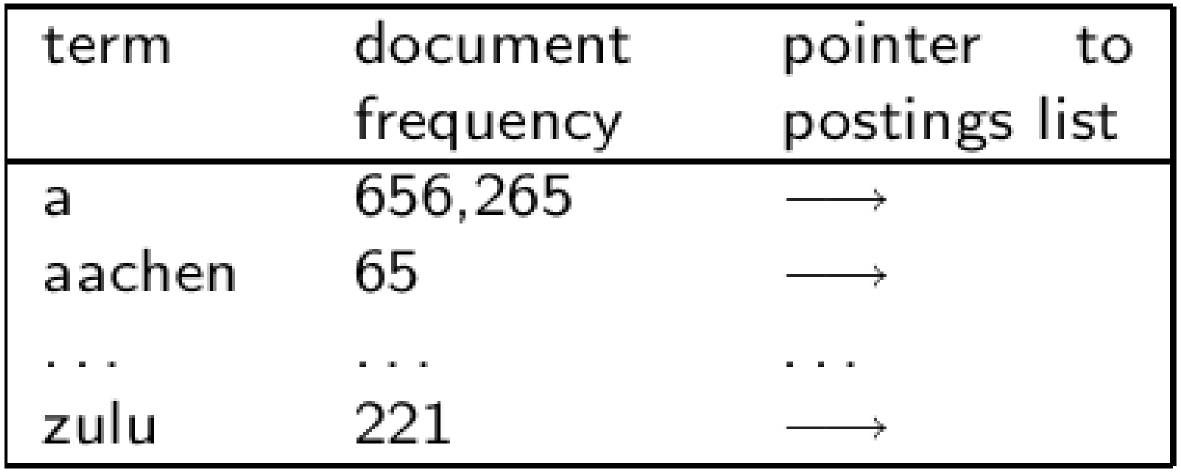
Problems in tokenization

- What are the delimiters? Space? Apostrophe? Hyphen?
- For each of these: sometimes they delimit, sometimes they don’t.
- No whitespace in many languages! (e.g., Chinese)
- No whitespace in Dutch, German, Swedish compounds (*Lebensversicherungsgesellschaftsangestellter*)

Problems with equivalence classing

- A term is an equivalence class of tokens.
- How do we define equivalence classes?
- Numbers (3/20/91 vs. 20/3/91)
- Case folding
- Stemming, Porter stemmer
- Equivalence classing problems in other languages
- More complex morphology than in English
- Finnish: a single verb may have 12,000 different forms
- Accents, umlauts
- Classifiers in the case of multi-lingual documents

Dictionary as array of fixed-width entries



term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
...	...	...
zulu	221	⟶

space needed: 20 bytes 4 bytes 4 bytes

How do we look up a query term *qi* in this array at query time? That is: which data structure do we use to locate the entry (row) in the array where *qi* is stored?

Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
- Is there a fixed number of terms or will it keep growing?
- What are the relative frequencies with which various keys will be accessed?
- How many terms are we likely to have?

Hashes

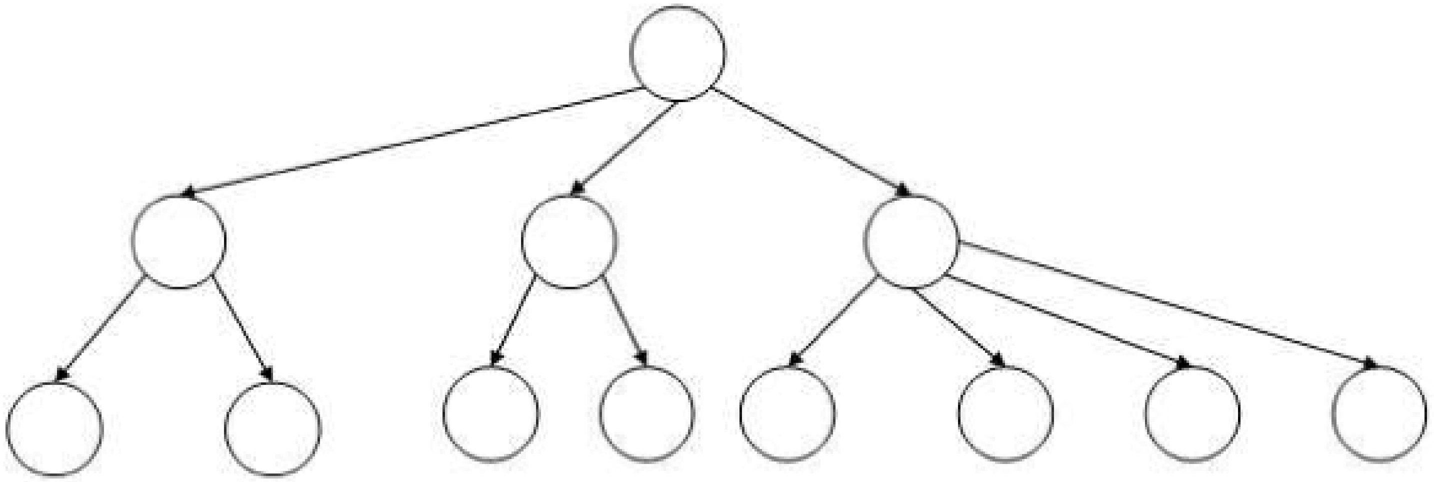
- Each vocabulary term is hashed into an integer
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree
    - Lookup time is constant
- Cons
    - no way to find minor variants (*resume* vs. *résumé*)
    - no prefix search (all terms starting with

Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where M is the size of the vocabulary.
- $O(\log M)$ only holds for balanced trees.
- Rebalancing binary trees is expensive.
- B-trees mitigate the rebalancing problem.
- B-tree definition: every internal node has a number of children in the interval $[a, b]$ where a, b are appropriate positive integers, e.g., [2, 4].

5
0

B-tree



Phrase queries

- Want to answer queries such as "**Bangalore *university*"** – as a phrase
- Thus the sentence *"I went to the university at Bangalore"* is not a match.
  - The concept of phrase queries has proven to be easily understood by users; around 10% of web queries are phrase queries
- It is no longer sufficient to store only

  <*term* : *docs*> entries in the inverted files!

Simple Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example the text "**Central University, Hyderabad**" would generate the biwords
  - ***Central University***
  - ***University Hyderabad***
- Each of these biwords is now a dictionary term.
- Two-word phrase query-processing is now immediate

Longer Phrase Queries

- ***Indian Institute of Science, Bangalore*** can be broken into the Boolean query on biwords:
- ***Indian Institute*** *AND* ***Institute of*** *AND* ***of Science*** *AND* ***Science Bangalore***
- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.
- So, *false positives* could be generated!
- For example, **Indian Institute of** *Technology conducts entrance examination at the Jain College* **of Science, Bangalore.**

Positional indexes

- Store, for each ***term***, entries of the form:

  <number of docs containing ***term***;

  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  etc.>

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to****:*
    - *2*:1,17,74,222,551; *4*:8,16,190,*429,433*; *7*:13,23,191; ...
  - ***be****:*
    - *1*:17,19; *4*:17,191,291,*430,434*; *5*:14,19,101; ...
- Same general method for proximity searches

Approximate Size

- A positional index is 2–4 times as large as a non-positional index
- Positional index size 35–50% of volume of original text
- This holds for “English-like” languages

Positional index size

- Compression of position values/offsets
- Problem is: positional index expands postings storage *substantially*
- It is now popularly used because of the power and usefulness of phrase and proximity queries.

Queries With Wild-card: *

- ***mon*:*** find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in “mon”: harder
  - Maintain an additional B-tree for terms *backwards.*
  - Can retrieve all words in range: ***nom ≤***

A * in the middle

- How can we handle *’s in the middle of query term?
- The solution: transform every wild-card query so that the *’s occur at the end
- This gives rise to the Permuterm Index.

Permuterm index

- Index term ***school*** under:
  - ***school$, chool$s, hool$sc, ool$sch, ol$scho, l$schoo,***
  - **where $ is a special symbol.**
- Queries:
  - **X** lookup on **X$** **X*** lookup on **X*$**
  - ***X** lookup on **X$*** **X*Y** lookup on **Y$X***
  -

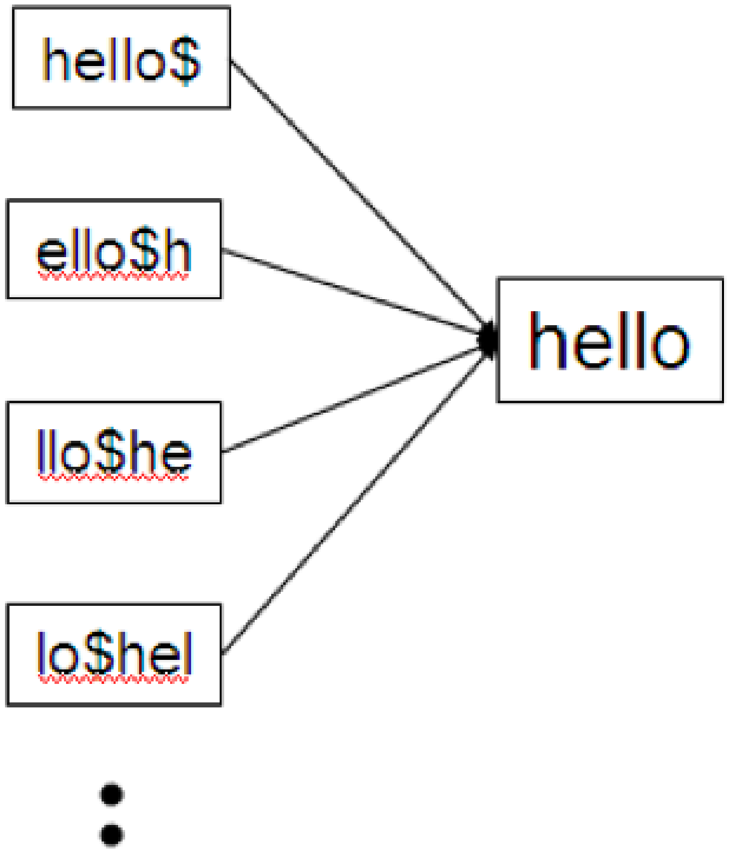
➢Query: ***scho*l***

➢**X=scho, Y=*l***

➢Lookup ***l*$scho***

➢Increases the lexicon size by a factor of 4

Permuterm → term mapping



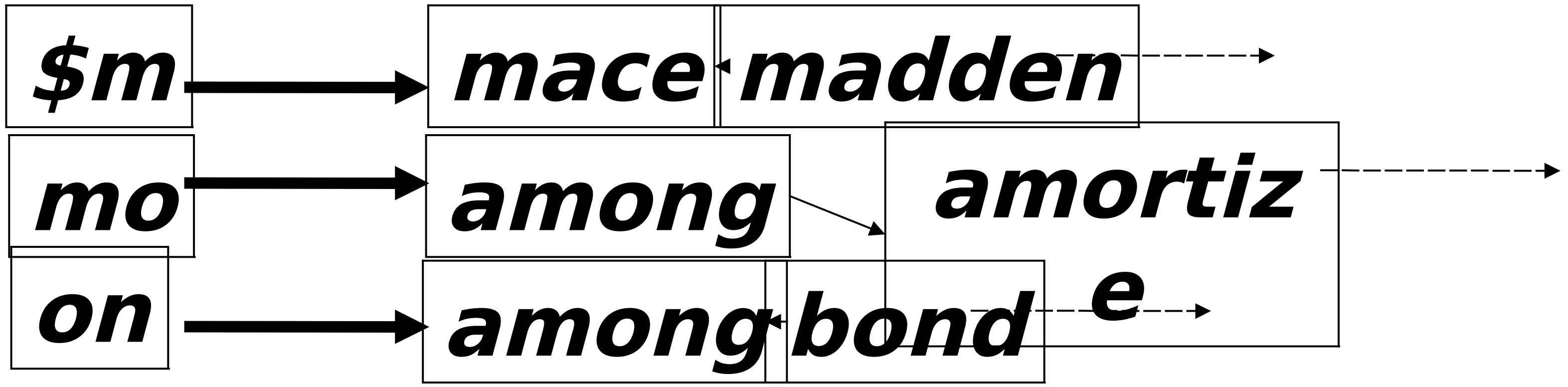
Bigram indexes

- Enumerate all *k*-grams (sequence of *k* chars) occurring in any term
- *e.g.,* from text "***April is the hottest month***" we get the 2-grams (*bigrams*)

a,ap,pr,ri,il,l,i,is,s,t,th,he,e,h,ho,ot, te,es,st,t, m,mo,on,nt,h

  - $ is a special word boundary symbol
- Maintain an "inverted" index from bigrams to *dictionary terms* that match each bigram.

Bigram index example



Processing *n*-gram wild-cards

- Query ***mon**** can now be run as
    - ***$m*** *AND* ***mo*** *AND* ***on***
- Fast and space efficient.
- Gets terms that match AND version of our wildcard query.
- But we enumerate ***moon*** also.
- Post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.

Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Retrieve matching documents when query contains a spelling error
- Two main flavors:
  - Isolated word
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words e.g., ***from*** and ***form***
  - Context-sensitive
    - Look at surrounding words, e.g., ***I flew <u>form</u> Chennai to Bangalore.***

Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come
- Two basic choices for this
  - A standard lexicon such as
    - Webster’s English Dictionary
    - An “industry-specific” lexicon – hand-maintained
  - The lexicon of the indexed corpus
    - E.g., all words on the web
    - All names, acronyms etc.
    - (Including the mis-spellings)

Isolated word correction

- Given a lexicon and a character sequence Q, return the words in the lexicon matching Q
- What is “matching”?
- There are two alternatives
  - Edit distance
  - *n*-gram overlap

Edit distance

- Given two strings S_1 and S_2, the minimum number of basic operations to convert one to the other
- Basic operations are typically character-level
  - Insert
  - Delete
  - Replace
- edit distance: ***cat*** to ***pot*** is 2; ***cat*** to ***rat*** is 1; ***cat*** to ***dog*** is 3.
- Weighted distance
  - Meant to capture keyboard errors, e.g. ***m*** more likely to be mis-typed as ***n*** than as ***q***
  - Therefore, replacing ***m*** by ***n*** is a smaller distance than by ***q***

Edit distance to all dictionary terms?

- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
  - Expensive and slow
  - Alternative?
- One possibility is to use *n*-gram overlap for this
- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams

  Variants – weight by keyboard layout, etc.

Example with trigrams

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*
- The query is ***december***
  - Trigrams are *dec, ece, cem, emb, mbe, ber*
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

- A commonly-used measure of overlap
- Let *X* and *Y* be two sets; then the J.C. is

$$\left|X \cap Y\right| / \left|X \cup Y\right|$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don’t have to be of the same size
- Always assigns a number between 0 and 1
  - Now threshold to decide if you have a match
  - E.g., if J.C. > 0.8, declare a match

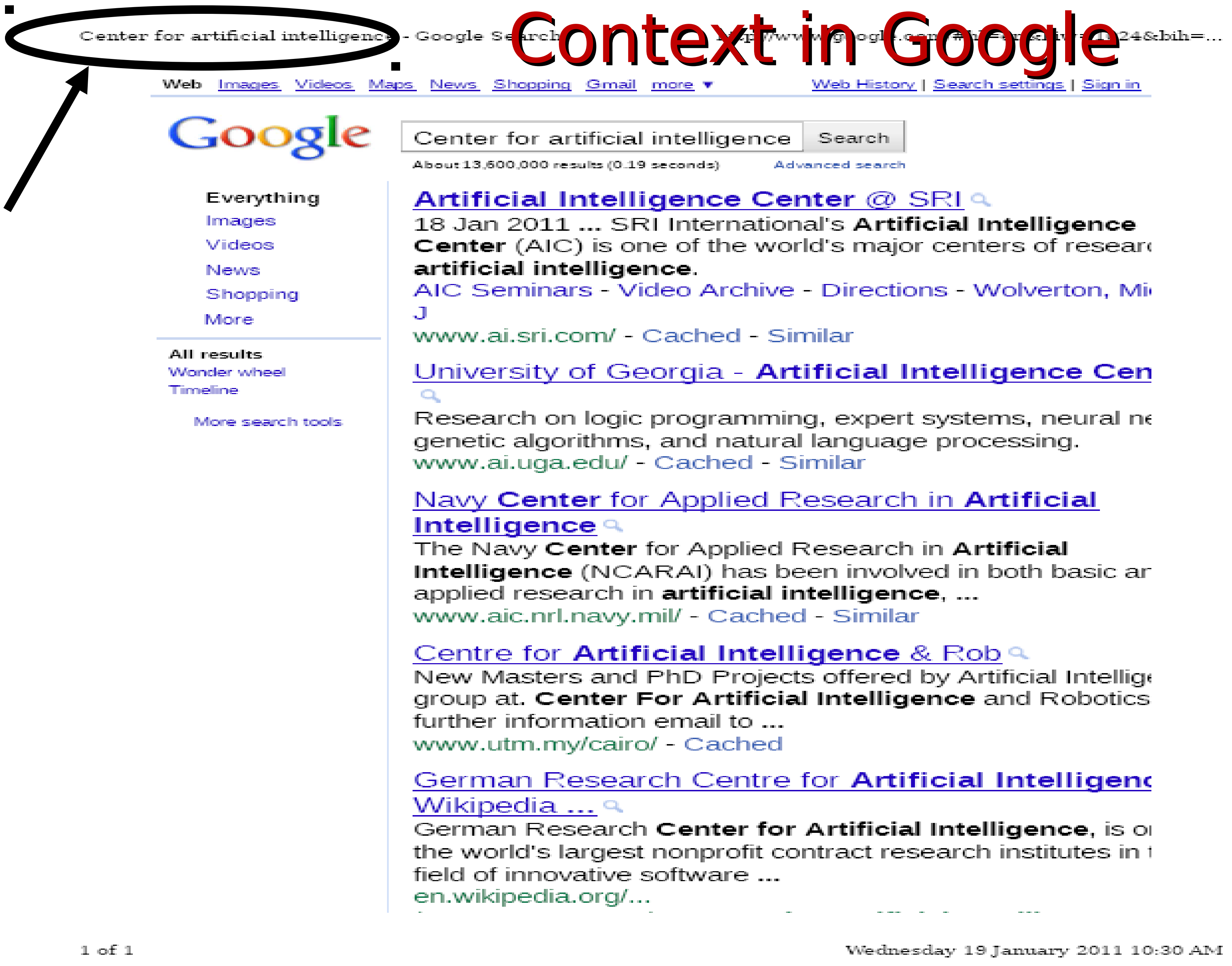
Context-sensitive spell correction

- Text: ***I flew <u>from</u> Bangalore to Chennai.***
- Consider the phrase query ***"flew <u>form</u> Bangalore"***
- We would like to respond
- Did you mean "***flew from Bangalore***"?
- because no documents matched the query phrase.

Context-sensitive correction

- Need surrounding context to catch this.
- First idea: retrieve dictionary terms close (in weighted edit distance) to each query term
- Now try all possible resulting phrases with one word “fixed” at a time
    - ***flew from bangalore***
    - ***fled form bangalore***
    - ***flea form bangalore***
    - ***etc.***
- Suggest the alternative that has lots of hits?

Context in Google



Center for artificial intelligence - Google Search http://www.google.com/#hl=en&biw=1024&bih=...

Web Images Videos Maps News Shopping Gmail more ▾ Web History | Search settings | Sign in

Center for artificial intelligence Search

About 13,600,000 results (0.19 seconds) Advanced search

Everything
Images
Videos
News
Shopping
More

All results
Wonder wheel
Timeline

More search tools

Artificial Intelligence Center @ SRI
18 Jan 2011 ... SRI International's **Artificial Intelligence Center** (AIC) is one of the world's major centers of researc
artificial intelligence.
AIC Seminars - Video Archive - Directions - Wolverton, Mi
J
www.ai.sri.com/ - Cached - Similar

University of Georgia - **Artificial Intelligence Cen**
Research on logic programming, expert systems, neural ne
genetic algorithms, and natural language processing.
www.ai.uga.edu/ - Cached - Similar

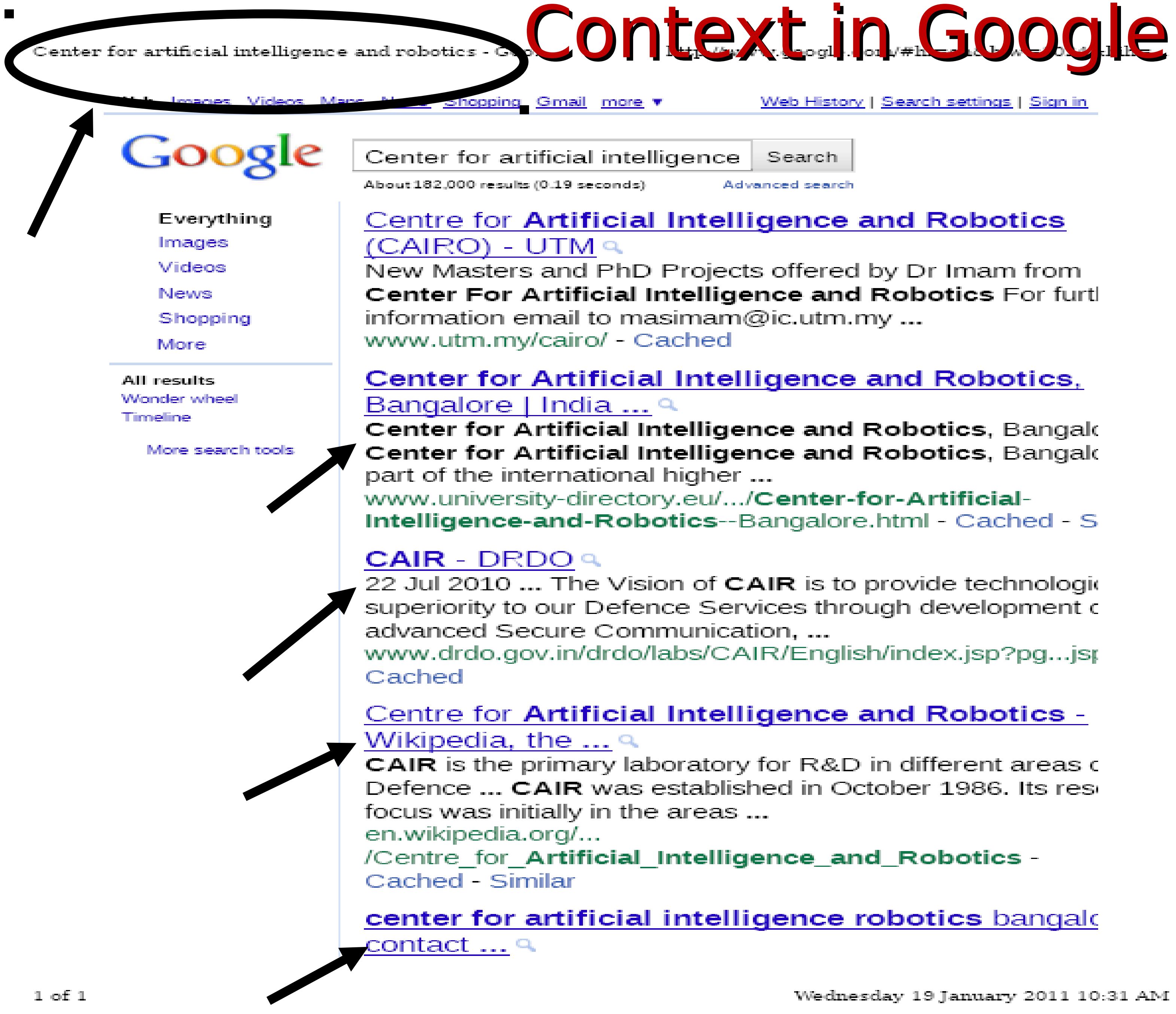
Navy **Center** for Applied Research in **Artificial Intelligence**
The Navy **Center** for Applied Research in **Artificial Intelligence** (NCARAI) has been involved in both basic ar
applied research in **artificial intelligence**, ...
www.aic.nrl.navy.mil/ - Cached - Similar

Centre for **Artificial Intelligence** & Rob
New Masters and PhD Projects offered by Artificial Intellige
group at. **Center For Artificial Intelligence** and Robotics
further information email to ...
www.utm.my/cairo/ - Cached

German Research Centre for **Artificial Intelligenc**
Wikipedia ...
German Research **Center for Artificial Intelligence**, is o
the world's largest nonprofit contract research institutes in t
field of innovative software ...
en.wikipedia.org/...

1 of 1 Wednesday 19 January 2011 10:30 AM

Context in Google



Center for artificial intelligence and robotics - Go... http://www.google.com/#hl=...

Web Images Videos Maps News Shopping Gmail more ▾ | Web History | Search settings | Sign in

Center for artificial intelligence | Search

About 182,000 results (0.19 seconds) | Advanced search

Everything
Images
Videos
News
Shopping
More

All results
Wonder wheel
Timeline

More search tools

Centre for **Artificial Intelligence and Robotics** (CAIRO) - UTM
New Masters and PhD Projects offered by Dr Imam from **Center For Artificial Intelligence and Robotics** For furtl
information email to masimam@ic.utm.my ...
www.utm.my/cairo/ - Cached

Center for Artificial Intelligence and Robotics, Bangalore | India ...
Center for Artificial Intelligence and Robotics, Bangalc
Center for Artificial Intelligence and Robotics, Bangalc
part of the international higher ...
www.university-directory.eu/.../**Center-for-Artificial-Intelligence-and-Robotics**--Bangalore.html - Cached - S

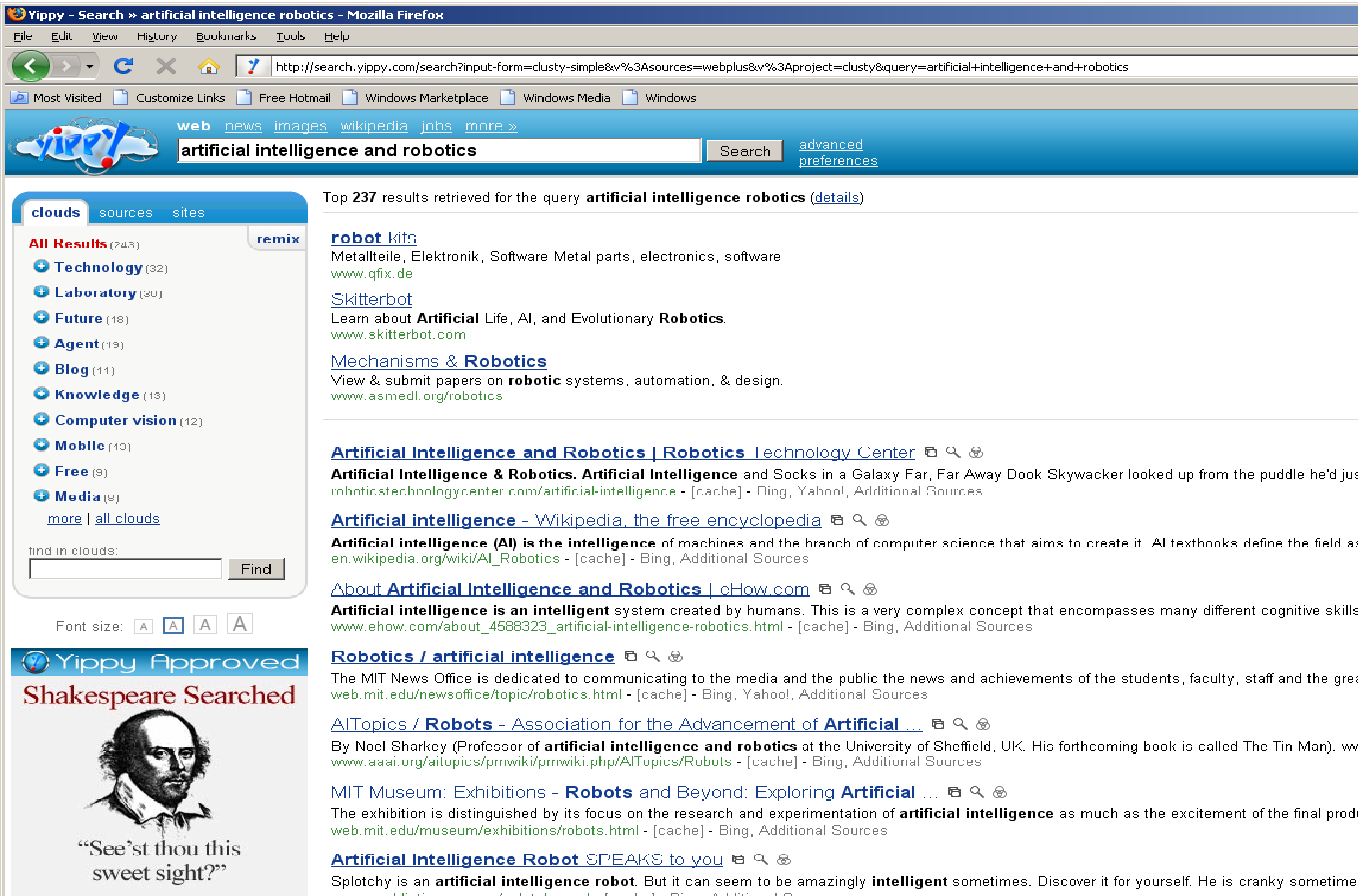
CAIR - DRDO
22 Jul 2010 ... The Vision of **CAIR** is to provide technologic
superiority to our Defence Services through development c
advanced Secure Communication, ...
www.drdo.gov.in/drdo/labs/CAIR/English/index.jsp?pg...jsp
Cached

Centre for **Artificial Intelligence and Robotics** - Wikipedia, the ...
CAIR is the primary laboratory for R&D in different areas c
Defence ... **CAIR** was established in October 1986. Its res
focus was initially in the areas ...
en.wikipedia.org/.../Centre_for_**Artificial_Intelligence_and_Robotics** -
Cached - Similar

center for artificial intelligence robotics bangalc
contact ...

1 of 1 | Wednesday 19 January 2011 10:31 AM

Context in Yippy


Yippy - Search » artificial intelligence robotics - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://search.yippy.com/search?input-form=clusty-simple&v%3Asources=webplus&v%3Aproject=clusty&query=artificial+intelligence+and+robotics
Most Visited Customize Links Free Hotmail Windows Marketplace Windows Media Windows
web news images wikipedia jobs more »
artificial intelligence and robotics
Search
advanced
preferences
clouds sources sites
remix
find in clouds:
Find
Font size: A A A A
Yippy Approved
Shakespeare Searched
"See'st thou this sweet sight?"

All Results (243)
- Technology (32)
- Laboratory (30)
- Future (18)
- Agent (19)
- Blog (11)
- Knowledge (13)
- Computer vision (12)
- Mobile (13)
- Free (9)
- Media (8)

more | all clouds

Top **237** results retrieved for the query **artificial intelligence robotics** (details)

robot kits
Metallteile, Elektronik, Software Metal parts, electronics, software
www.qfix.de

Skitterbot
Learn about **Artificial** Life, AI, and Evolutionary **Robotics**.
www.skitterbot.com

Mechanisms & **Robotics**
View & submit papers on **robotic** systems, automation, & design.
www.asmedl.org/robotics

Artificial Intelligence and Robotics | Robotics Technology Center
Artificial Intelligence & Robotics. Artificial Intelligence and Socks in a Galaxy Far, Far Away Dook Skywacker looked up from the puddle he'd jus
roboticstechnologycenter.com/artificial-intelligence - [cache] - Bing, Yahoo!, Additional Sources

Artificial intelligence - Wikipedia, the free encyclopedia
Artificial intelligence (AI) is the intelligence of machines and the branch of computer science that aims to create it. AI textbooks define the field as
en.wikipedia.org/wiki/AI_Robotics - [cache] - Bing, Additional Sources

About **Artificial Intelligence and Robotics** | eHow.com
Artificial intelligence is an intelligent system created by humans. This is a very complex concept that encompasses many different cognitive skills
www.ehow.com/about_4588323_artificial-intelligence-robotics.html - [cache] - Bing, Additional Sources

Robotics / artificial intelligence
The MIT News Office is dedicated to communicating to the media and the public the news and achievements of the students, faculty, staff and the grea
web.mit.edu/newsoffice/topic/robotics.html - [cache] - Bing, Yahoo!, Additional Sources

AITopics / **Robots** - Association for the Advancement of **Artificial** ...
By Noel Sharkey (Professor of **artificial intelligence and robotics** at the University of Sheffield, UK. His forthcoming book is called The Tin Man). wh
www.aaai.org/aitopics/pmwiki/pmwiki.php/AITopics/Robots - [cache] - Bing, Additional Sources

MIT Museum: Exhibitions - **Robots** and Beyond: Exploring **Artificial** ...
The exhibition is distinguished by its focus on the research and experimentation of **artificial intelligence** as much as the excitement of the final prod
web.mit.edu/museum/exhibitions/robots.html - [cache] - Bing, Additional Sources

Artificial Intelligence Robot SPEAKS to you
Splotchy is an **artificial intelligence robot**. But it can seem to be amazingly **intelligent** sometimes. Discover it for yourself. He is cranky sometime

Query expansion

- Usually expand query rather than index
  - No index blow up
  - Query processing slowed down
    - Documents frequently contain equivalences
  - May retrieve different results
    - ***puma*** and ***jaguar*** retrieves documents on cars instead of on jerkins.

Caching Results

- ıf many users are searching for
  criket and dhoni
- then you probably *do* need spelling correction, but you *don't* need to keep on intersecting those two postings lists
- Web query distribution is extremely skewed, and you can usefully cache results for common queries.

The Vector-Space Model

- Assume l distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These “orthogonal” terms form a vector space.

$$\text{Dimension} = l = |\text{vocabulary}|$$

- Each term, i, in a document or query, j, is given a real-valued weight, $w_{ij.}$
- Both documents and queries are expressed as l-dimensional vectors:

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{lj})$$

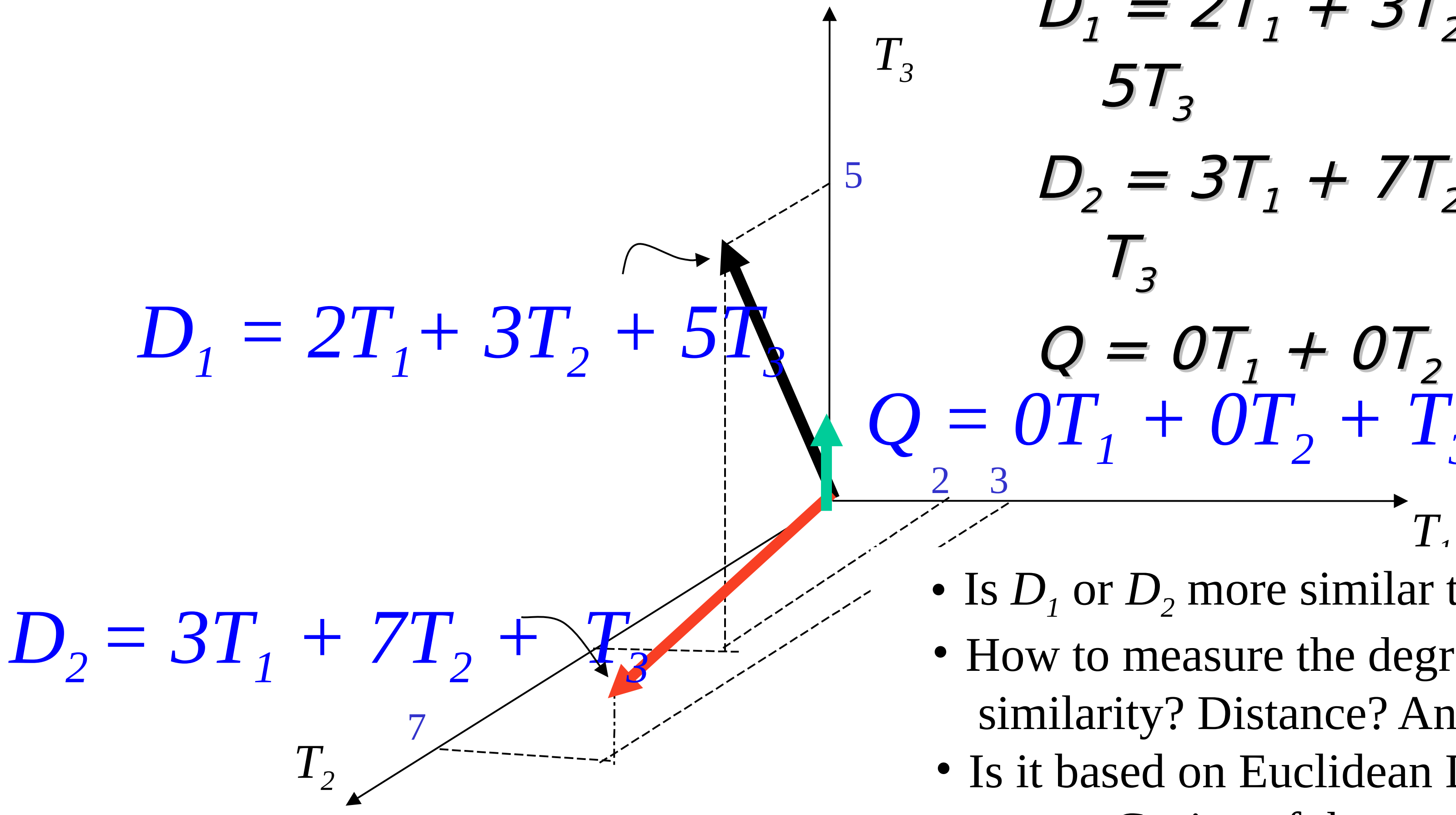
Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle?
- Is it based on Euclidean Dist? or Cosine of the angle between the vectors?

Similarity Measure - Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product:

$$\text{sim}(\boldsymbol{d}_j,\boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where w_{ij} is the weight of term *i* in document *j* and w_{iq} is the weight of term *i* in the query

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

How good is the retrieval?

- *Precision* : Fraction of retrieved docs that are relevant to user’s information need
- *Recall* : Fraction of relevant docs in collection that are retrieved

TF-IDF Weights

- *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N / df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Computing TF-IDF -- An Example

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and

document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

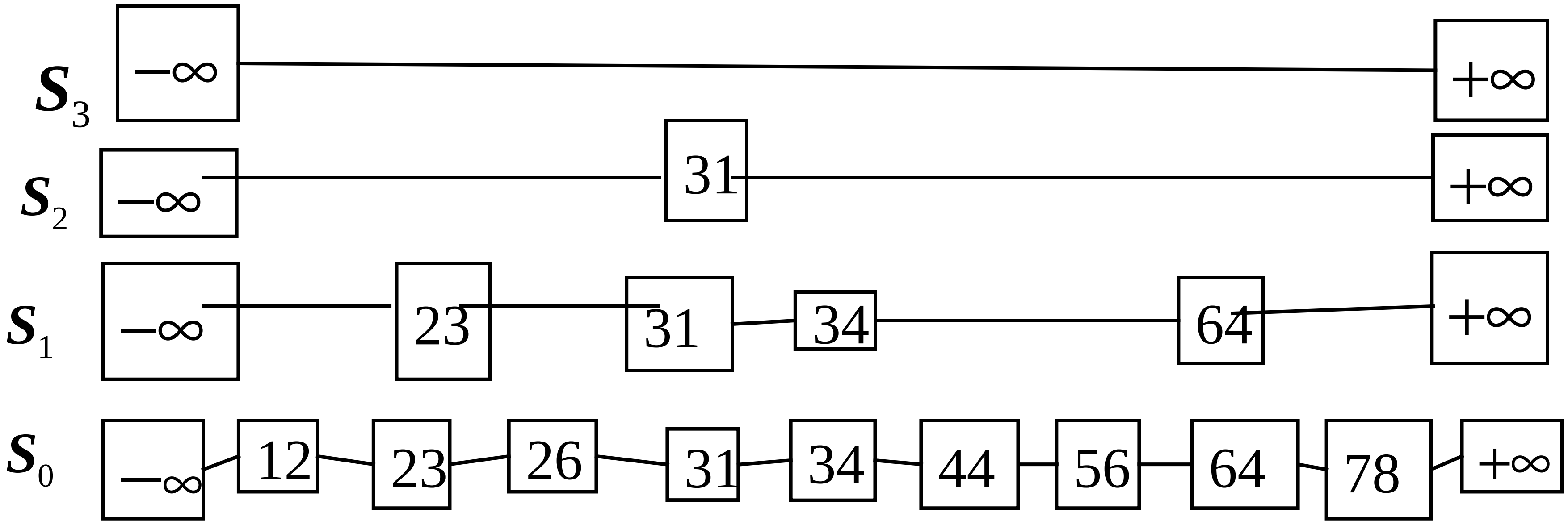
A: tf = 3/6; idf = log(10000/50) = 5.3; tf-idf = 2.6

B: tf = 2/6; idf = log(10000/1300) = 2.0; tf-idf = 0.66

What is a Skip List

A skip list for a set ***S*** of distinct (key, element) items is a series of lists $\boldsymbol{S}_0$, $\boldsymbol{S}_1$, … , $\boldsymbol{S}_h$ such that

- Each list $\boldsymbol{S}_i$ contains the special keys $+\infty$ and $-\infty$
- List $\boldsymbol{S}_0$ contains the keys of ***S*** in nondecreasing order
- Each list is a subsequence of the previous one, i.e., $\boldsymbol{S}_0 \supseteq \boldsymbol{S}_1 \supseteq \ldots \supseteq \boldsymbol{S}_h$

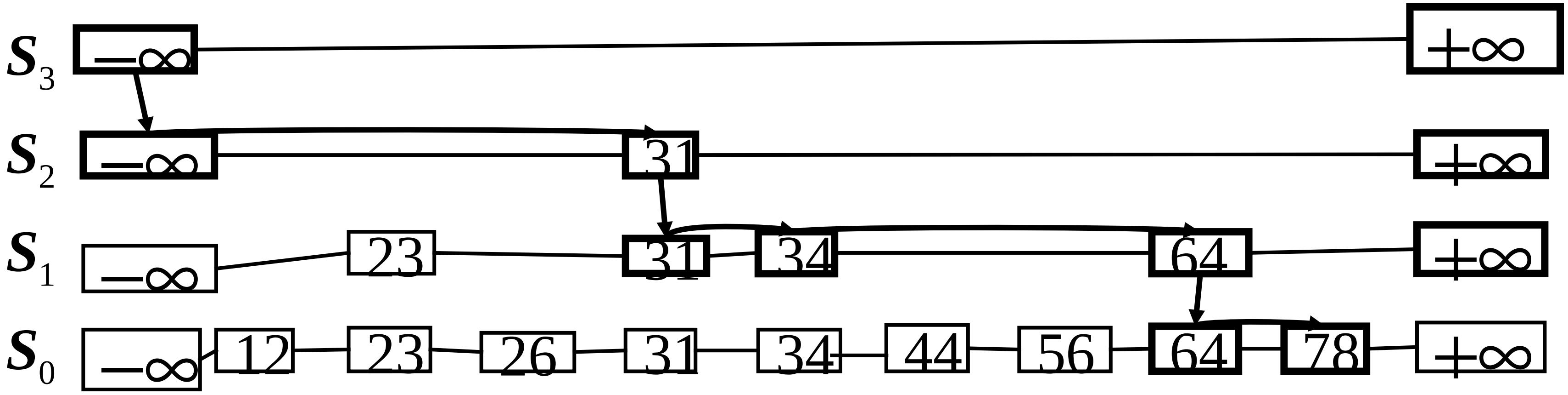


Search

We search for a key ***x*** in a a skip list as follows:

- We start at the first position of the top list
- At the current position ***p***, we compare ***x*** with ***y*** $\leftarrow$ ***key***(***after***(***p***))
- ***x*** = ***y***: we return ***element***(***after***(***p***))
- ***x*** > ***y***: we "scan forward"
- ***x*** < ***y***: we "drop down"
- If we try to drop down past the bottom list, we return ***NO_SUCH_KEY***

Example: search for 78

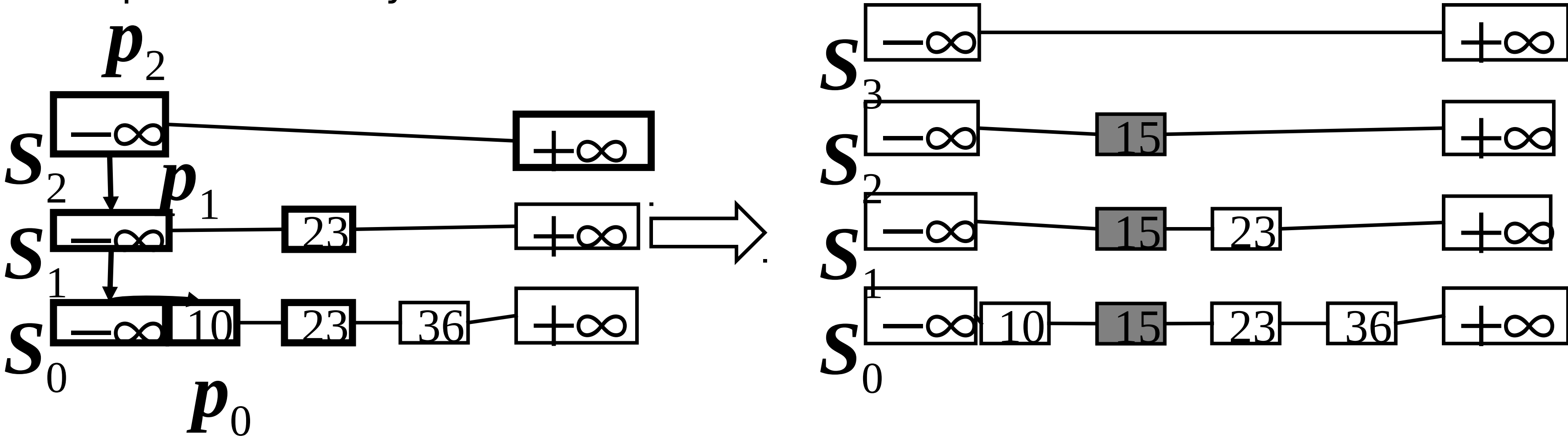


Insertion

To insert an item (***x***, ***o***) into a skip list, we use a randomized algorithm:

- We repeatedly toss a coin until we get tails, and we denote with ***i*** the number of times the coin came up heads
- If $i \geq h$, we add to the skip list new lists $S_{h+1}, \ldots, S_{i+1}$, each containing only the two special keys
- We search for ***x*** in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with largest key less than ***x*** in each list $S_0, S_1, \ldots, S_i$
- For $j \leftarrow 0, \ldots, i$, we insert item (***x***, ***o***) into list S_j after position p_j

Example: insert key 15, with $i = 2$



Space Usage

1. Fact 1: The probability of getting ***i*** consecutive heads when flipping a coin is $1/2^i$
2. Fact 2: If each of ***n*** items is present in a set with probability ***p***, the expected size of the set is ***np***

Consider a skip list with ***n*** items

By Fact 1, we insert an item in list S_i with probability $1/2^i$

By Fact 2, the expected size of list S_i is $n/2^i$

The expected number of nodes used by the skip list is

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

- Thus, the expected space usage of a skip list with ***n*** items is ***O***(***n***)

Search and Update Times

- The search time in a skip list is proportional to
  - the number of drop-down steps, plus
  - the number of scan-forward steps
- The drop-down steps are bounded by the height of the skip list and thus are $O(\log n)$ with high probability
- To analyze the scan-forward steps, we use yet another probabilistic fact:
  - Fact 4: The expected number of coin tosses required in order to get tails is 2
- When we scan forward in a list, the destination key does not belong to a higher list
  - A scan-forward step is associated with a former coin toss that gave tails
- By Fact 4, in each list the expected number of scan-forward steps is 2
- Thus, the expected number of scan-forward steps is $O(\log n)$
- We conclude that a search in a skip list takes $O(\log n)$ expected time
- The analysis of insertion and deletion gives similar results

Introduction to Algorithms — March 18, 2004
Massachusetts Institute of Technology — 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser — Handout 14

Lecture Notes on Skip Lists

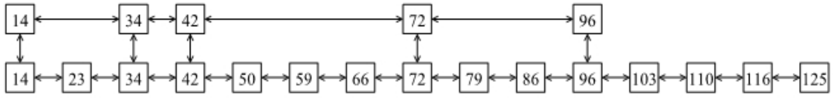
Lecture 12 — March 18, 2004

Erik Demaine

- Balanced tree structures we know at this point: B-trees, red-black trees, treaps.
- Could you implement them right now? Probably, with time... but without looking up any details in a book?
- Skip lists are a simple randomized structure you'll never forget.

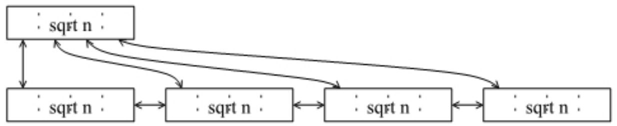
Starting from scratch

- Initial goal: *just searches* — ignore updates (Insert/Delete) for now
- Simplest data structure: linked list
- Sorted linked list: $\Theta(n)$ time
- 2 sorted linked lists:
  - Each element can appear in 1 or both lists
  - How to speed up search?
  - **Idea:** Express and local subway lines
  - **Example:** $\boxed{14}$, 23, $\boxed{34}$, $\boxed{42}$, 50, 59, 66, $\boxed{72}$, 79, 86, $\boxed{96}$, 103, 110, 116, 125
    (What is this sequence?)
  - $\boxed{\text{Boxed}}$ values are "express" stops; others are normal stops
  - Can quickly jump from express stop to next express stop, or from any stop to next normal stop
  - Represented as two linked lists, one for express stops and one for all stops:

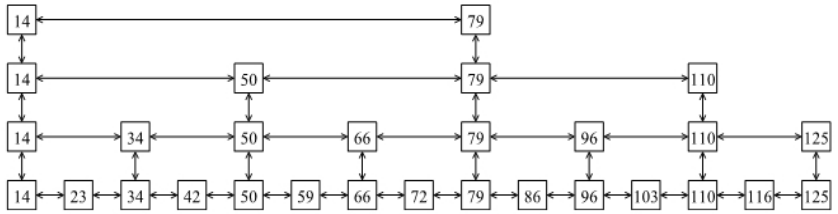


  - Every element is in linked list 2 (LL2); some elements also in linked list 1 (LL1)
  - Link equal elements between the two levels
  - To search, first search in LL1 until about to go too far, then go down and search in LL2

- Cost:
$$\text{len}(\text{LL1}) + \frac{\text{len}(\text{LL2})}{\text{len}(\text{LL1})} = \text{len}(\text{LL1}) + \frac{n}{\text{len}(\text{LL1})}$$
- Minimized when
$$\begin{aligned} & \text{len}(\text{LL1}) = \frac{n}{\text{len}(\text{LL1})} \\ \Rightarrow\ & \text{len}(\text{LL1})^2 = n \\ \Rightarrow\ & \text{len}(\text{LL1}) = \sqrt{n} \\ \Rightarrow\ & \text{search cost} = 2\sqrt{n} \end{aligned}$$
- Resulting 2-level structure:



- 3 linked lists: $3 \cdot \sqrt[3]{n}$
- k linked lists: $k \cdot \sqrt[k]{n}$
- $\lg n$ linked lists: $\lg n \cdot \sqrt[\lg n]{n} = \lg n \cdot \underbrace{n^{1/\lg n}}_{=2} = \Theta(\lg n)$
  - Becomes like a binary tree:



  - **Example:** Search for 72
    - ∗ Level 1: 14 too small, 79 too big; go down 14
    - ∗ Level 2: 14 too small, 50 too small, 79 too big; go down 50
    - ∗ Level 3: 50 too small, 66 too small, 79 too big; go down 66
    - ∗ Level 4: 66 too small, 72 spot on

Insert

- New element should certainly be added to bottommost level
  (Invariant: Bottommost list contains all elements)
- Which other lists should it be added to?
  (Is this the entire balance issue all over again?)
- **Idea:** Flip a coin
  - With what probability should it go to the next level?
  - To mimic a balanced binary tree, we'd like half of the elements to advance to the next-to-bottommost level
  - So, when you insert an element, flip a fair coin
  - If heads: add element to next level up, and flip another coin (repeat)
- Thus, on average:
  - $1/2$ the elements go up 1 level
  - $1/4$ the elements go up 2 levels
  - $1/8$ the elements go up 3 levels
  - Etc.
- Thus, "approximately even"

Example

- Get out a real coin and try an example
- You should put a special value ∞ at the beginning of each list, and always promote this special value to the highest level of promotion
- This forces the leftmost element to be present in every list, which is necessary for searching

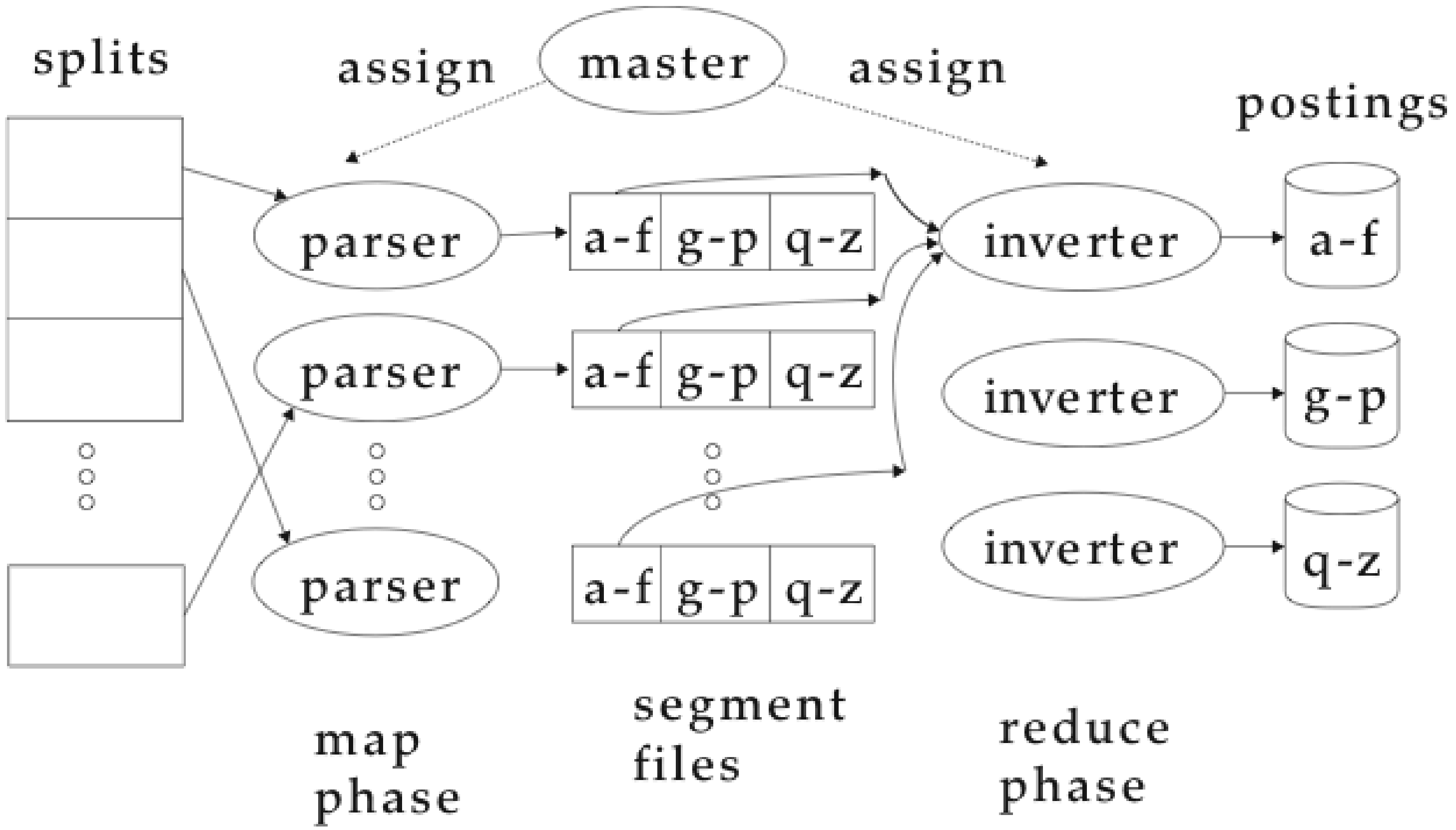
... many coins are flipped ...
(Isn't this easy?)

- The result is a skip list.
- It probably isn't as balanced as the ideal configurations drawn above.
- It's clearly good on average.
- Claim it's really really good, almost always.

Properties of Skip Lists

- A skip list is a data structure for dictionaries that uses a randomized insertion algorithm
- In a skip list with $\boldsymbol{n}$ items
  - The expected space used is $\boldsymbol{O}(\boldsymbol{n})$
  - The expected search, insertion and deletion time is $\boldsymbol{O}(\log \boldsymbol{n})$
- Using a more complex probabilistic analysis, one can show that these performance bounds also hold with high probability
- Skip lists are fast and simple to implement in practice

MapReduce for index construction



Why compression? (in general)

- Use less disk space (saves money)
- Keep more stuff in memory (increases speed)
- Increase speed of transferring data from disk to memory (again, increases speed)
- [read compressed data and decompress in memory] is faster than [read uncompressed data]
- Premise: Decompression algorithms are fast.
- This is true of the decompression algorithms we will use.

Why compression in information retrieval?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk
- Note: Large search engines keep significant part of postings in memory
- We will devise various compression schemes for dictionary and postings.

Lossy vs. lossless compression

- Lossy compression: Discard some information
- Several of the preprocessing steps we frequently use can be viewed as lossy compression:
- downcasing, stop words, porter, number elimination
- Lossless compression: All information is preserved.
- What we mostly do in index compression

Model collection: The Reuters collection

symbol	statistics	value
N	Documents	800,000
L	avg. # tokens per document	200
M	word types	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
T	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term (= word type)	7.5
	non-positional postings	10^8

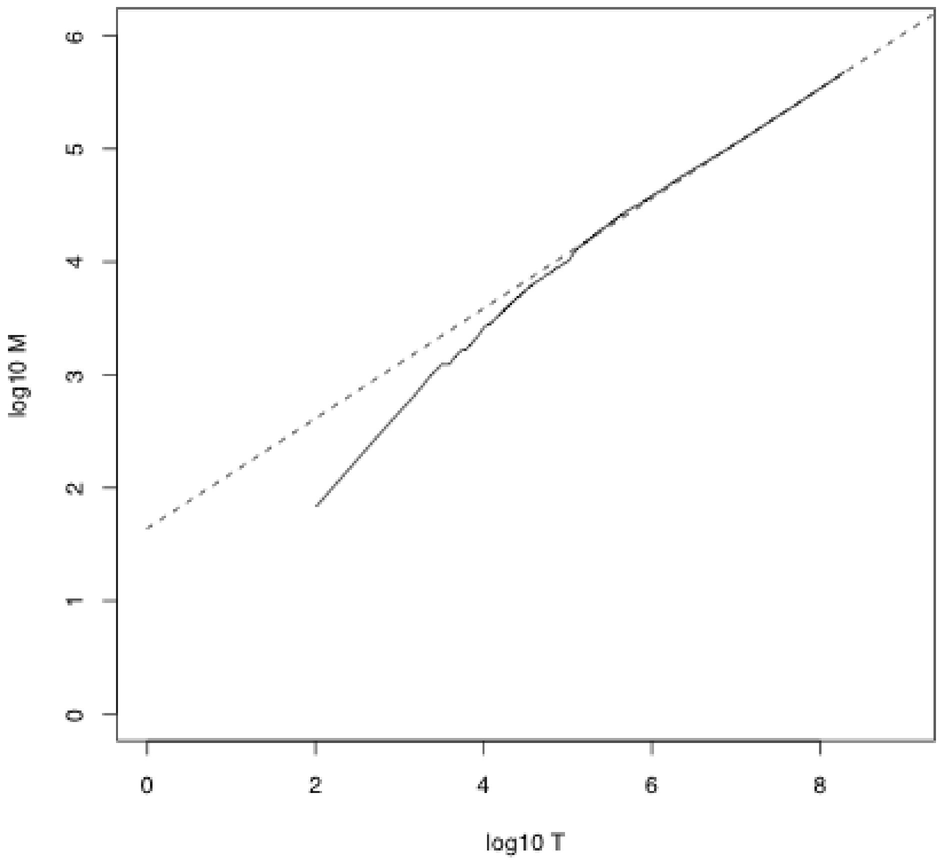
How big is the term vocabulary?

- That is, how many distinct words are there?
- Can we assume there is an upper bound?
- The vocabulary will keep growing with collection size

  Heaps' law: $M = kT$^b
- M is the size of the vocabulary, T is the number of tokens in the collection
- Typical values for the parameters k and b are: $30 \leq k \leq 100$ and

  $b \approx 0.5$
- Heaps' law is linear in log-log space
- It is the simplest possible relationship between collection size and vocabulary size in log-log space
- Empirical law

Heaps' law for Reuters



Vocabulary size M as a
function of collection size
T (number of tokens) for
Reuters-RCV1. For these
data, the dashed line
$\log 10 M =$
$0.49 * \log 10\ T +$
1.64 is the
best least squares fit.
Thus, $M =$

Empirical fit for Reuters

- Good, as we just saw in the graph
- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

  $$44 \times 1{,}000{,}020\text{^}(0.49) \approx 38{,}323$$
- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general

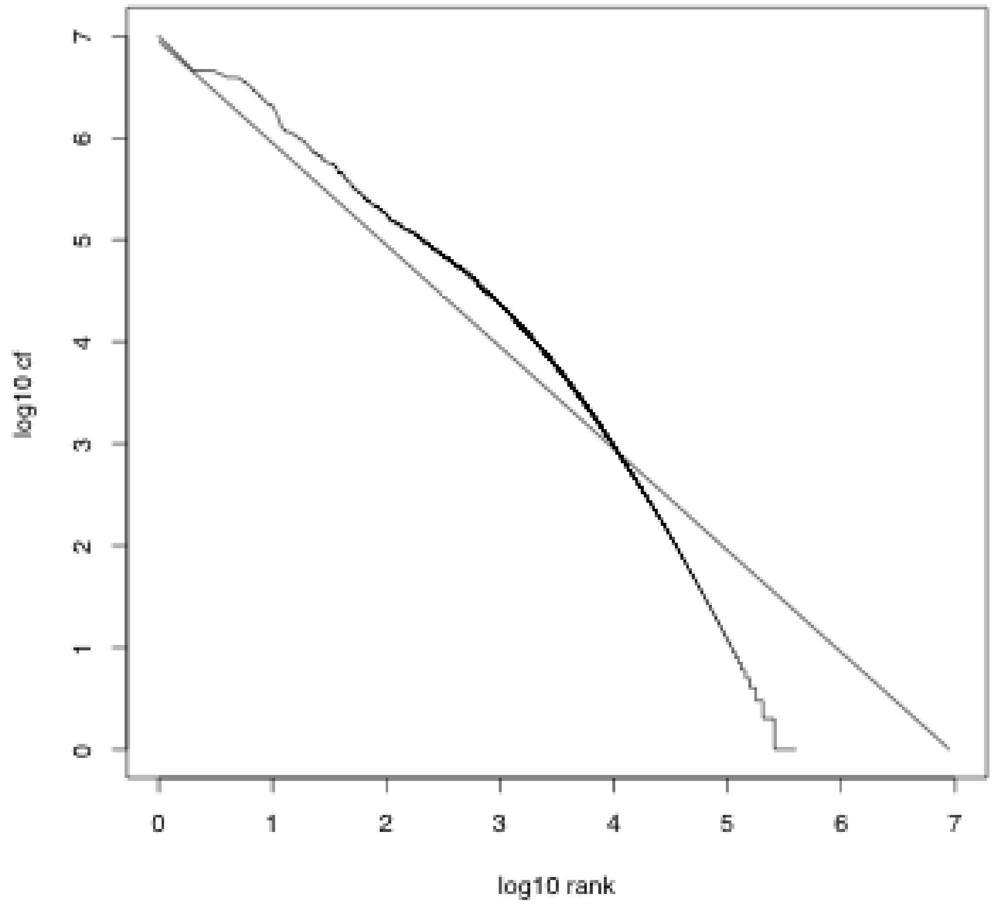
Zipf’s law

- Now we have characterized the growth of the vocabulary in collections.
- We also want to know how many frequent vs. infrequent terms we should expect in a collection.
- In natural language, there are a few very frequent terms and very many very rare terms.
- Zipf’s law: The *i*th most frequent term has frequency cf*i* proportional to 1/*i* .
- cf*i* is collection frequency: the number of occurrences of the te $\mathrm{cf}_i \propto \frac{1}{i}$ in the collection.

Zipf’s law

- Zipf’s law: The *i*th most frequent term has frequency propor $\mathrm{cf}_i \propto \frac{1}{i}$ to 1/*i* .
- cf is collection frequency: the number of occurrences of the term in the collection.
- So if the most frequent term (*the*) occurs cf1 times, then the second most frequent term (*of*) $\mathrm{cf}_2 = \frac{1}{2}\mathrm{cf}_1$ . . .s many occurrences
- . . . and the third most frequent term (*and*) has a third as many oc $\mathrm{cf}_3 = \frac{1}{3}\mathrm{cf}_1$ es
- Equivalent: cf$i$ = ci^k and log cf$i$ = log c + k log i (for $k = -1$)
- Example of a power law

Zipf’s law for Reuters



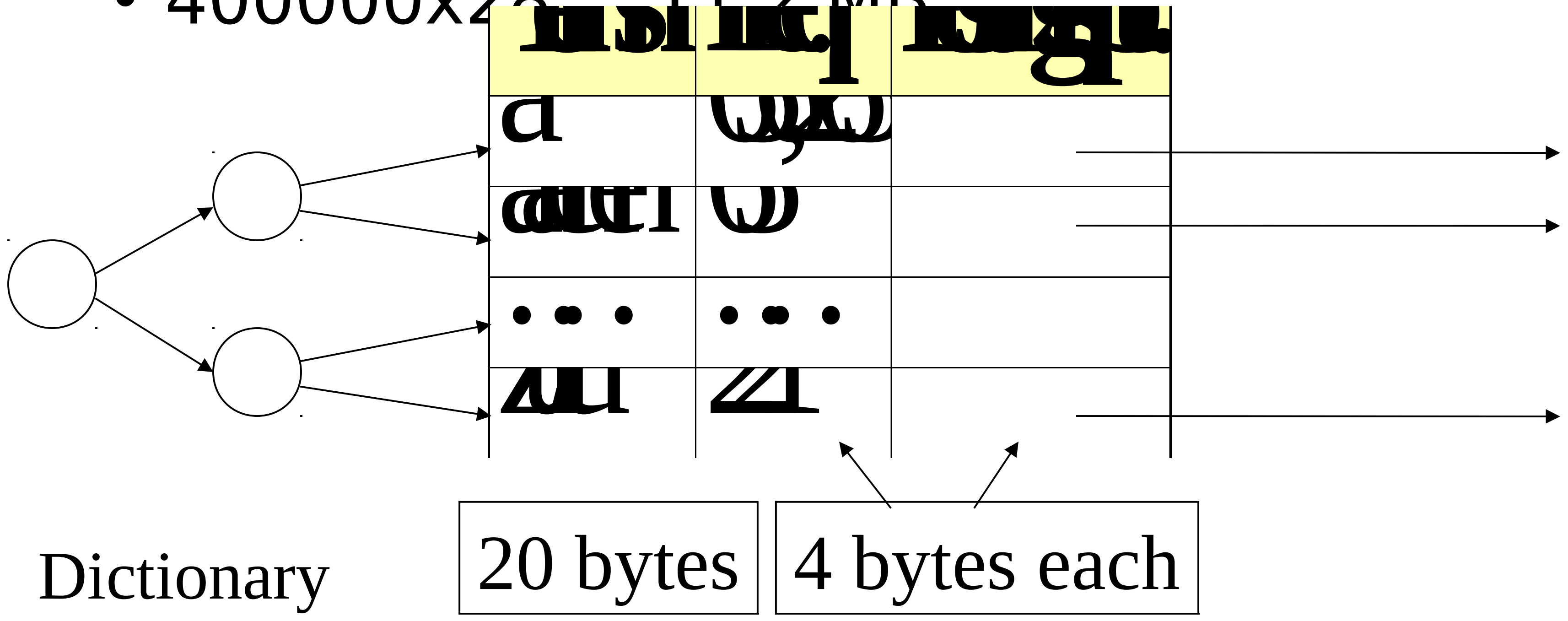
Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.

Compression of Inverted Indexes

- Dictionary
  - Make it small enough to keep in main memory
  - Make it so small that you can keep some postings lists in main memory too
- Postings file(s)
  - Reduces disk space needed
  - Decreases time needed to read postings lists from disk
  - Large search engines keep a significant part of the postings in memory.

Dictionary storage – fixed width

- Array of fixed-width entries
  - ~400,000 terms; 28 (20 + 4 +4) bytes/term;
  - 400000x28 = 11.2 MB



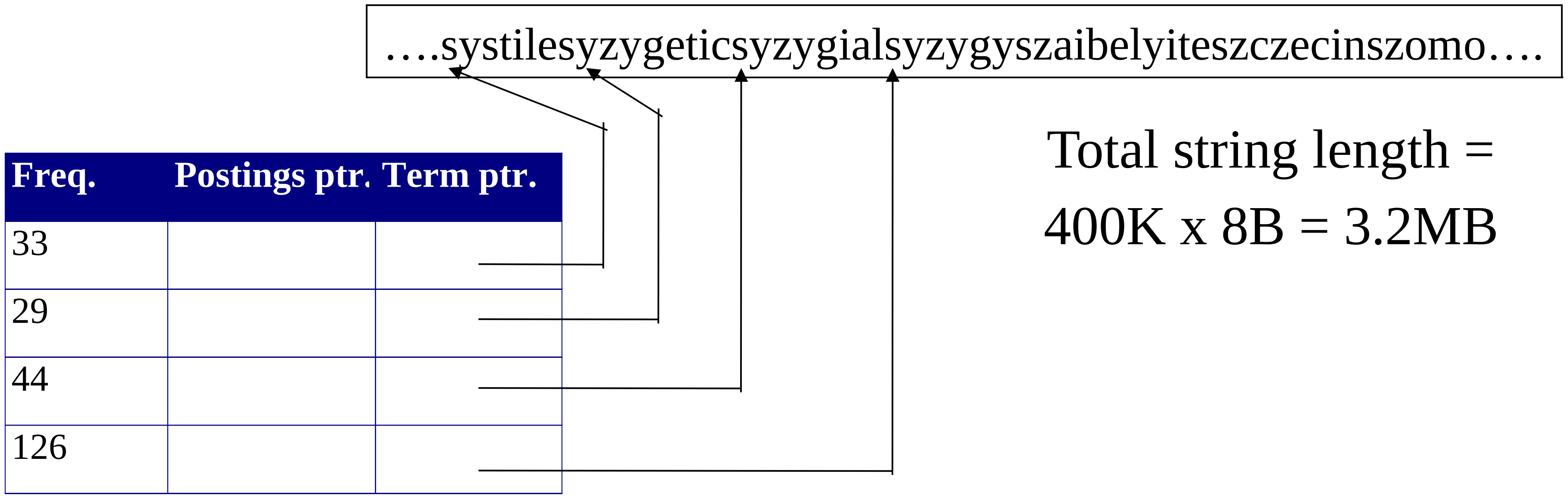
[illegible]	[illegible]	[illegible]
a	[illegible]	
[illegible]	[illegible]	
…	…	
[illegible]	[illegible]	

Problem with Fixed-Width Terms

- Most of the bytes in the **Term** column are wasted – we allot 20 bytes for even one letter terms.
  - And we still can't handle *a term like hydrochlorofluorocarbons.*
- Average dictionary word in English: ~8 characters
  - Can we use ~8 characters per dictionary term?

Store the Dictionary as a String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Can save up to 60% of dictionary space.



Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length =

400K x 8B = 3.2MB

Pointers resolve 3.2M positions:

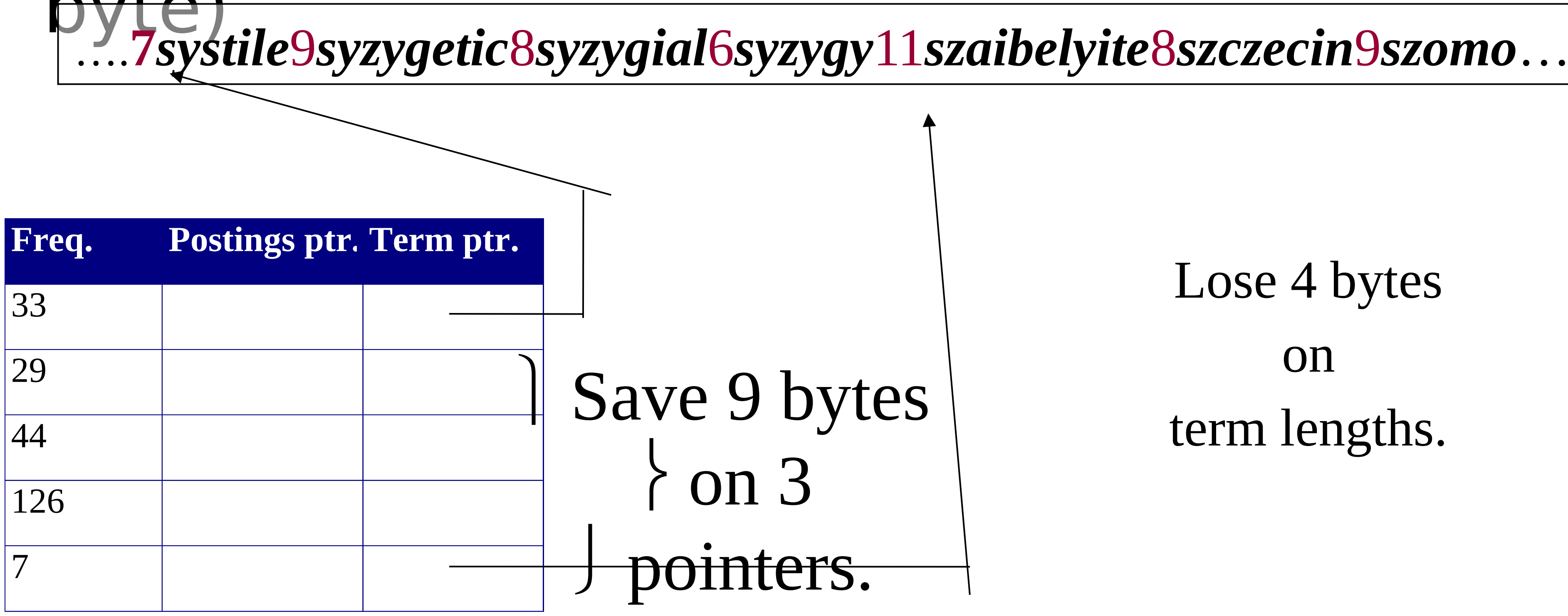
$\log_2 3.2M = 22 bits = 3 bytes$

Space for dictionary as a string

- 4 bytes per term for Document Frequency
- 4 bytes per term for postings pointer
- 3 bytes per term pointer
- Avg. 8 bytes per term in term string
- 400K terms x 19 (4+4+3+8)⇒ 7.6 MB (against 11.2MB for fixed

Term Pointer to a Block of strings

- Store pointers to every *k*th term string.
  - Example: *k*=4.
- Need to store length of the term (1 extra byte)



….7***systile***9***syzygetic***8***syzygial***6***syzygy***11***szaibelyite***8***szczecin***9***szomo***….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

Save 9 bytes } on 3 pointers.

Lose 4 bytes on term lengths.

Front coding

- Front-coding:
  - Lexicographically Sorted words commonly have long common prefix – store differences only
  - (for last *k-1* in a block of *k*)

8***automata***8***automate***9***automatic***10***automation***

→8***automat****a1◊e2◊ic3◊ion**

Encodes ***automat***

Extra length beyond ***automat.***

RCV1 dictionary compression summary

Fixed width	11.2
Dictionary-as-String with pointers to every term	7.6
Blocking $k = 4$	7.1
Blocking + front coding	5.9

Postings compression

- The postings file can typically be larger than the dictionary, by a factor of at least 10.
- A posting for simplicity is a docID.
- For Reuters (800,000 documents), we use 32 bits per docID when using 4-byte integers.
- We can use $\log_2 800{,}000 \approx 20$ bits per docID.
- Can we use a lot less than 20 bits per

Postings: Document Frequency of Terms

- A term like ***cryptology*** occurs in maybe one doc out of a million – we would like to store this posting using $\log_2$ 1M ~ 20 bits.
- A term like ***the*** occurs in virtually every doc, so 20 bits/posting is too expensive.
  - Prefer 0/1 bitmap vector in this case

Document Gaps

- We store the list of docs containing a term in increasing order of docID.
  - ***computer***: 33,47,154,159,202 …
- <u>Consequence</u>: it suffices to store *gaps*.
  - 33,14,107,5,43 …
- <u>Possibility</u>: gaps of freq. terms are small can be encoded/stored with far fewer than 20 bits; infreq. terms can have large gaps, but few.

Postings Entries: Example

Term	Encoding	postings				
THE	docID	...	123101	123102	123103	123104
	GAPS			1	1	1
COMPUTER	docID	...	120222	120328	120331	120355
	GAPS			106	3	24
CRYPTOLOGY	docID	110201	234432			
	GAPS	110201	124231			

Variable length encoding

- Aim:
  - For ***cryptology***, use ~20 bits/gap entry.
  - For ***the***, use ~1 bit/gap entry.
- If the average gap for a term is *G*, we use ~$\log_2 G$ bits/gap entry.
- Need to encode every gap with about as few bits as needed for that integer.
- This requires a *variable length encoding*
- Variable length codes use short codes for small numbers and long codes for large

Example

docIDs	568	574	214121
gaps		6	213547
VB code	00000100 10111000	10000110	00001101 00000100 10101011

Postings stored as the byte concatenation

000001001011100010000110000011010000010010101011

- VB-encoded postings are uniquely prefix-decodable.
- For a small gap of 6, VB encoding uses a whole byte!

GOOGLE

- ***From googol – (10)100***
- Prefers pages in which query terms are near
- Automatic “and” queries; “OR” and “not” (-)
- Searches are not case sensitive
- Stop words are ignored; can add(+) them
- Does not use stemming
- Phrase searches are permitted; “
- Advanced operators
  - Cache (cached version of webpage)
  - Link (restrict search to pages linking-link:iisc)
  - Allintitle:web mining – docs having both in title