Information Retrieval

M.Tech. CS Elective

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

Tentative Course Outline: Part 1

- Introduction to IR. Applications. Brief history.
- MapReduce and Hadoop.
- Apache Spark and tutorial.
- Terms and documents, term-document matrix, inverted index. Ranked retrieval, Vector space model, basic term weighting. Index structure, index creation.
- Query processing using inverted index, skip lists, champion lists, gd-ordering. Index compression.
- Tokenisation, stemming, lemmatization, stopword removal. Dictionary structure, tolerant retrieval.
- Apache lucene tutorial. Discussion on Solr and ElasticSearch.
- Evaluation: precision, recall, average precision, NDCG, other metrics, test collections, evaluation forums, sound experimental methods.
- Relevance feedback, pseudo relevance feedback, query expansion. Latent semantic indexing.
- Probabilistic models for IR. Okapi BM25. Language models for IR.
- Web search: crawling and indexing.
- Advertising on the web: Adwords.
- Deduplication: min-hashing, locality sensitive hashing.
- Link analysis: term-spam, Markov chain, PageRank, link spam, hubs and authorities.

Tentative Course Outline: Part 2

- Machine learning basics. Classification methods: Naive Bayes, SVM. (Assumed to be covered in ML1)
- Clustering basics: hierarchical clustering, point-assignment clustering, k-means. Search result clustering. (ML1)
- Logistic regression, Deep neural network, gradient descent.
- Overview of deep learning optimization techniques.
- Recurrent neural networks, LSTM, GRU.
- Convolutional neural networks and its applications on text data.
- Word embeddings: neural LM, word2vec, GloVe, ELMo, FastText.
- Attention, Transformer.
- BERT, fine tuning BERT for transfer learning. Text classification and applications in IR.
- Learning to Rank.
- Query suggestion.
- Recommender systems.
- Personalization of search results.
- Text summarization.
- Question answering.
- Summary and overview of recent trends in IR.

References

- Textbooks (for part 1)
  - Introduction to Information Retrieval, by Manning, Raghavan and Schütze.
  - Mining of Massive Datasets, by Leskovec, Rajaraman and Ullman.
  - Both books are available in PDF online.
- References for part 2 will include several other sources and will be given during the course.

Information Retrieval

Introduction and a Brief History

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

Information Retrieval

User needs some information.

An information retrieval system tries to bridge this gap.

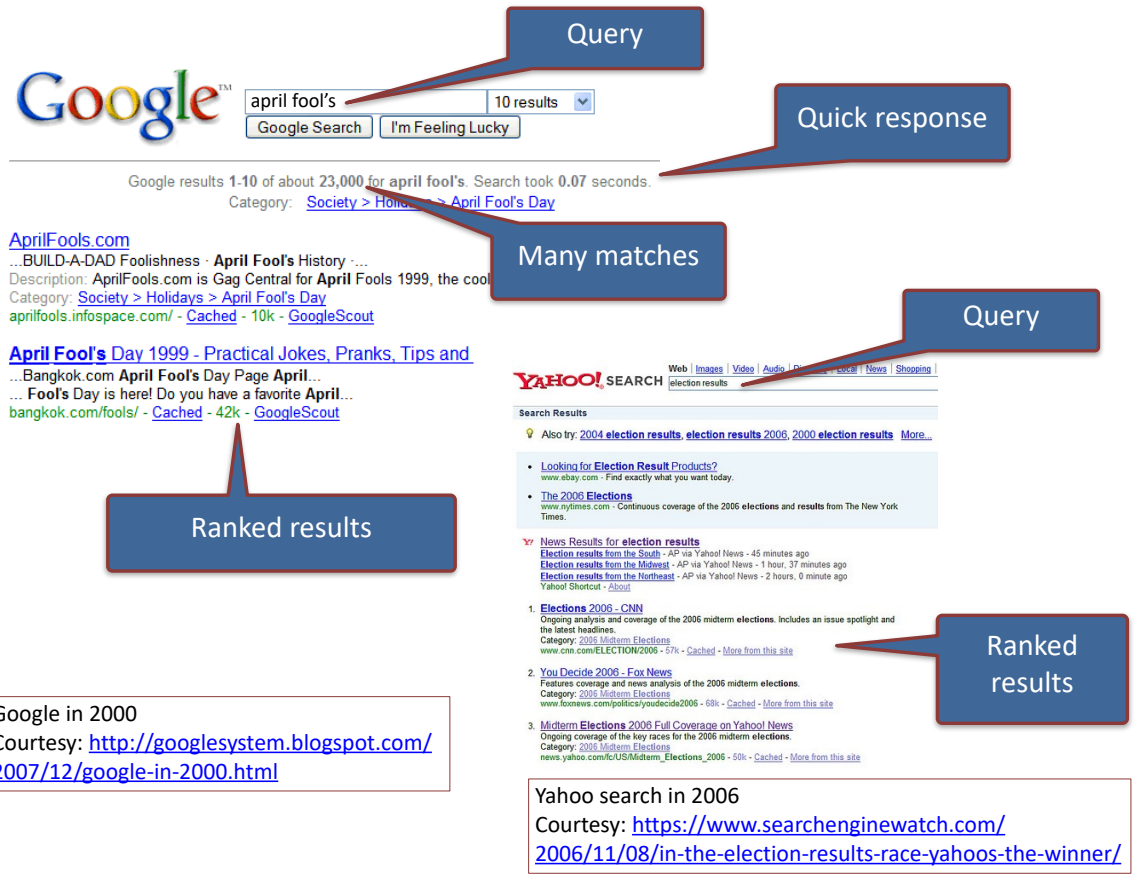
Assumption: the required information is present somewhere.

The goal of an information retrieval system is to satisfy user's information need.

Basic example

User expresses the information need in the form of a query.

The system returns a (ranked) list of results.



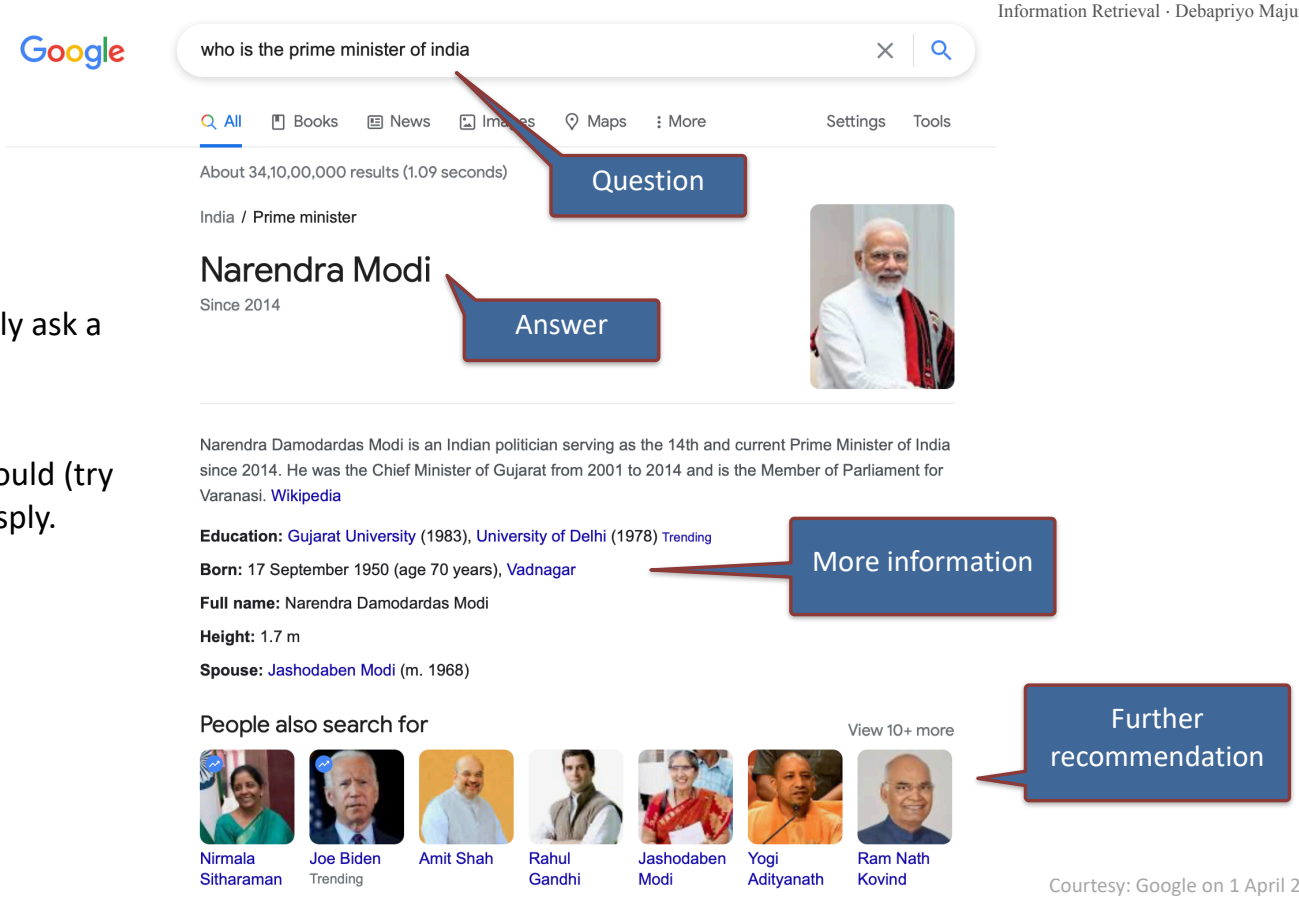
Google in 2000
Courtesy: http://googlesystem.blogspot.com/2007/12/google-in-2000.html

Yahoo search in 2006
Courtesy: https://www.searchenginewatch.com/2006/11/08/in-the-election-results-race-yahoos-the-winner/

Question Answering

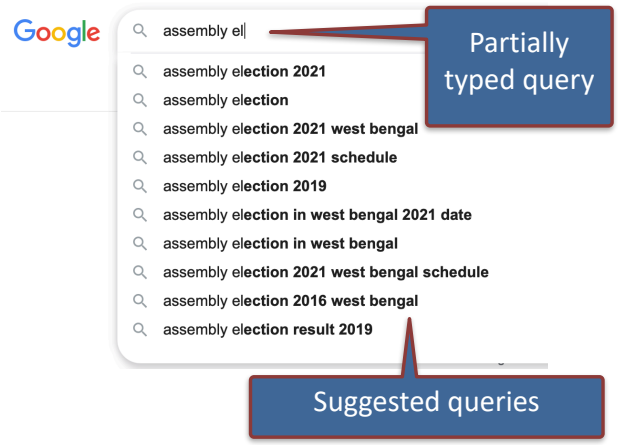
User can simply ask a question.

The system would (try to) answer crisply.



Courtesy: Google on 1 April 2021

Query suggestions



User may not be sure what would be a good query.

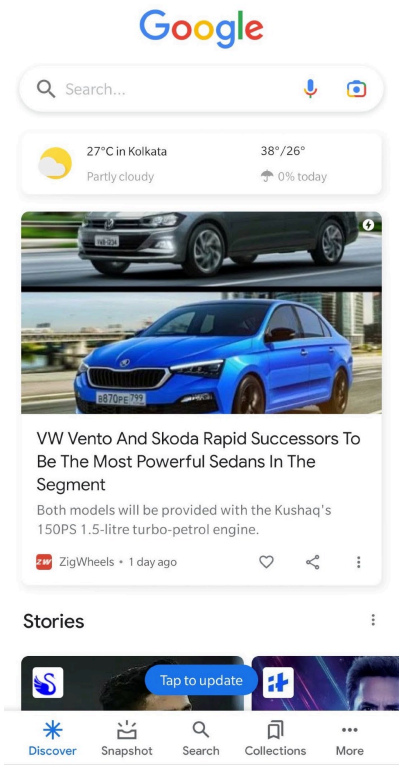
The system helps the user to formulate a query as the user types on.

Interactive IR.

Recommendation

User may not even have to ask, the system tries to "feed" information which may be "helpful".

Proactive IR.



Courtesy: Google on 1 April 2021

Brief history

- 1800s and till 1930s: only librarians or paralegals had to *retrieve* information by searching (manually).
- 1950s: use of computer for information retrieval started.
  - IR started as a discipline: how to index documents, and how to retrieve them.
  - From Boolean retrieval to ranked retrieval: not all *matches* are equally good.
- 1978: ACM SIGIR conference started.
- Standard test collections became important to measure success of IR methods.
  - 1992: Text Retrieval Conference (TREC) started.
- 1990s: The world wide web started growing and web search engines came up: Altavista, Yahoo.
- 1998: Google originated from Stanford University with the invention of PageRank algorithm.
- 2000s: Search advertising made Google a tech giant.
  - Also, internet reached a huge portion of the world's population, search became a necessity.
  - Machine learning (later deep learning and reinforcement learning) became integral parts of IR.

Sanderson, Mark, and W. Bruce Croft. "The history of information retrieval research." *Proceedings of the IEEE* 100, no. Special Centennial Issue (2012): 1444-1451.

References

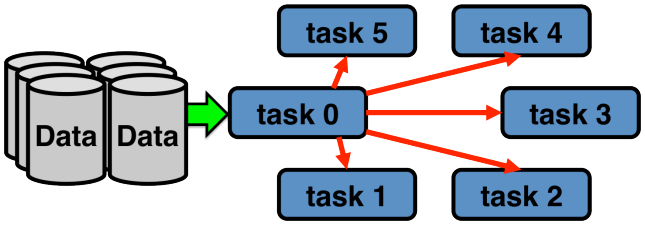
- Sanderson, Mark, and W. Bruce Croft. "The history of information retrieval research." *Proceedings of the IEEE* 100, no. Special Centennial Issue (2012): 1444-1451.
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. "Introduction to Information Retrieval", Cambridge University Press. 2008.

MapReduce

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

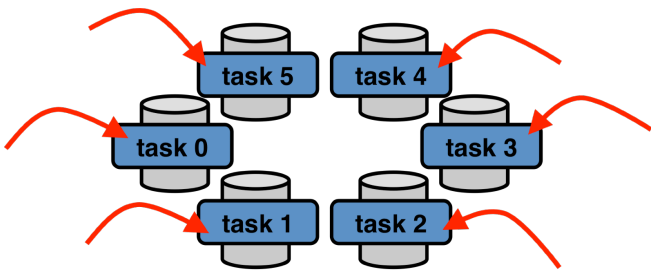
Computer(s) and Data

- Modern data mining: process immense amount of data quickly
- Exploit parallelism



Traditional parallelism

Bring data to the computer

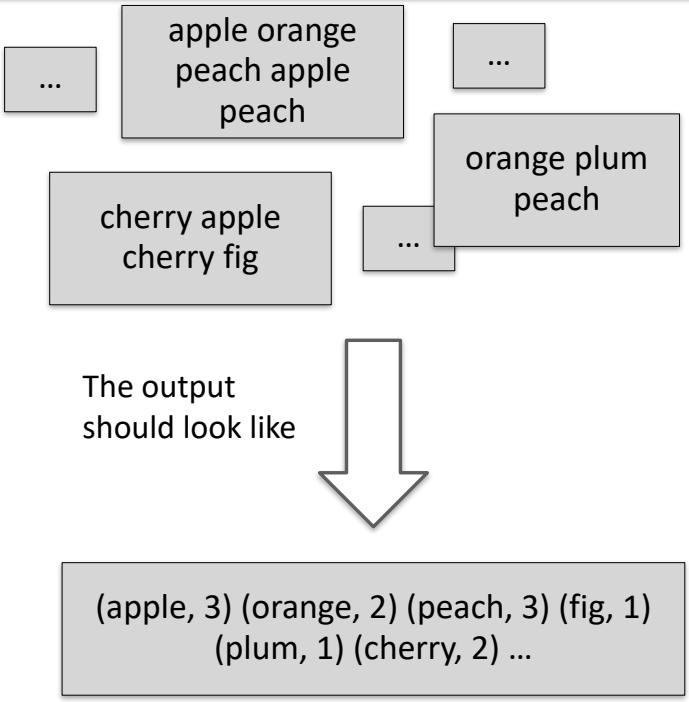


MapReduce

Compute where the data is present

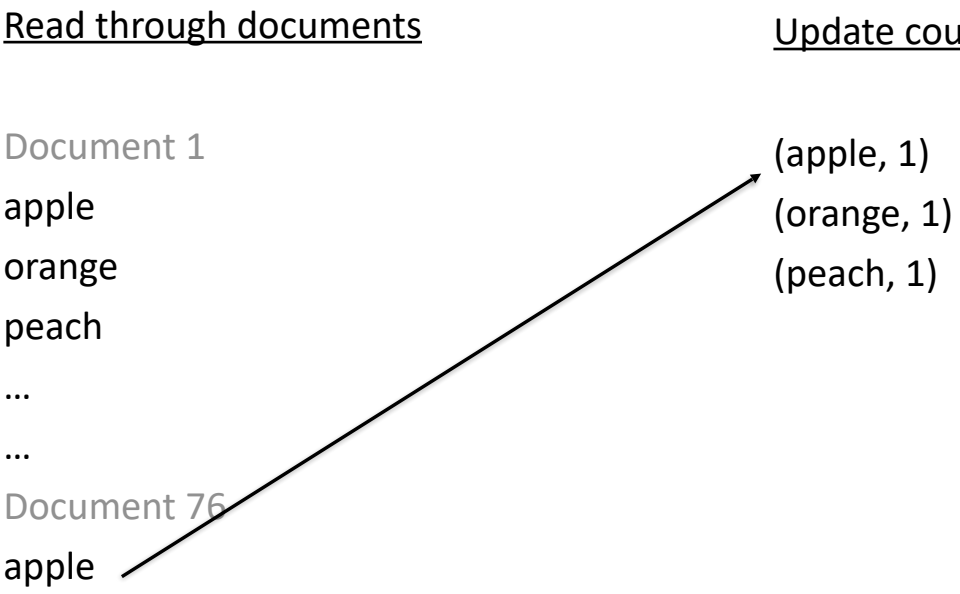
Pictures courtesy: Glenn K. Lockwood, glennklockwood.com

The "Hello World" problem of Big Data



- Many documents, say millions, billions
  - Each with several words
- Goal: word count
  - Count total number of occurrences for each word

Normally, how would we do it?



Normally, how would we do it?

Read through documents

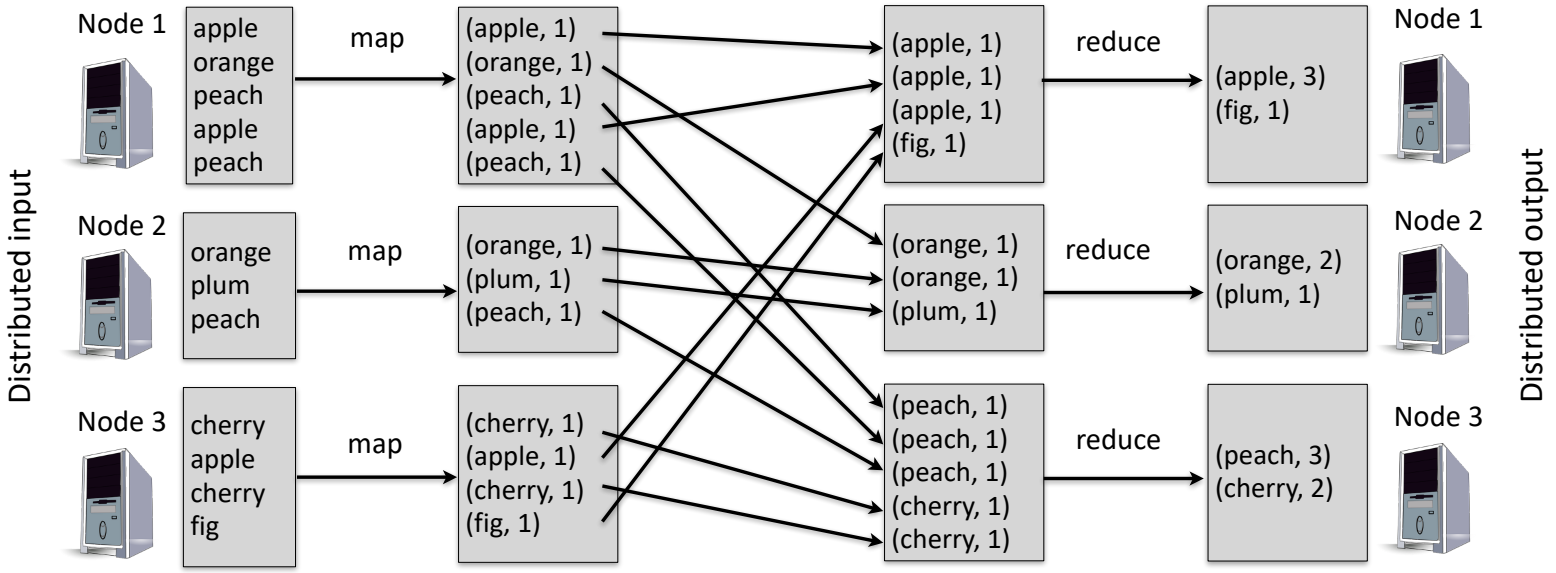
Update counters in memory

Document 1
apple
orange
peach
…
…
Document 76
apple

(apple, **2**)
(orange, 1)
(peach, 1)

Word frequency using MapReduce

Given a collection of documents, count the number of times each word occurs in the collection

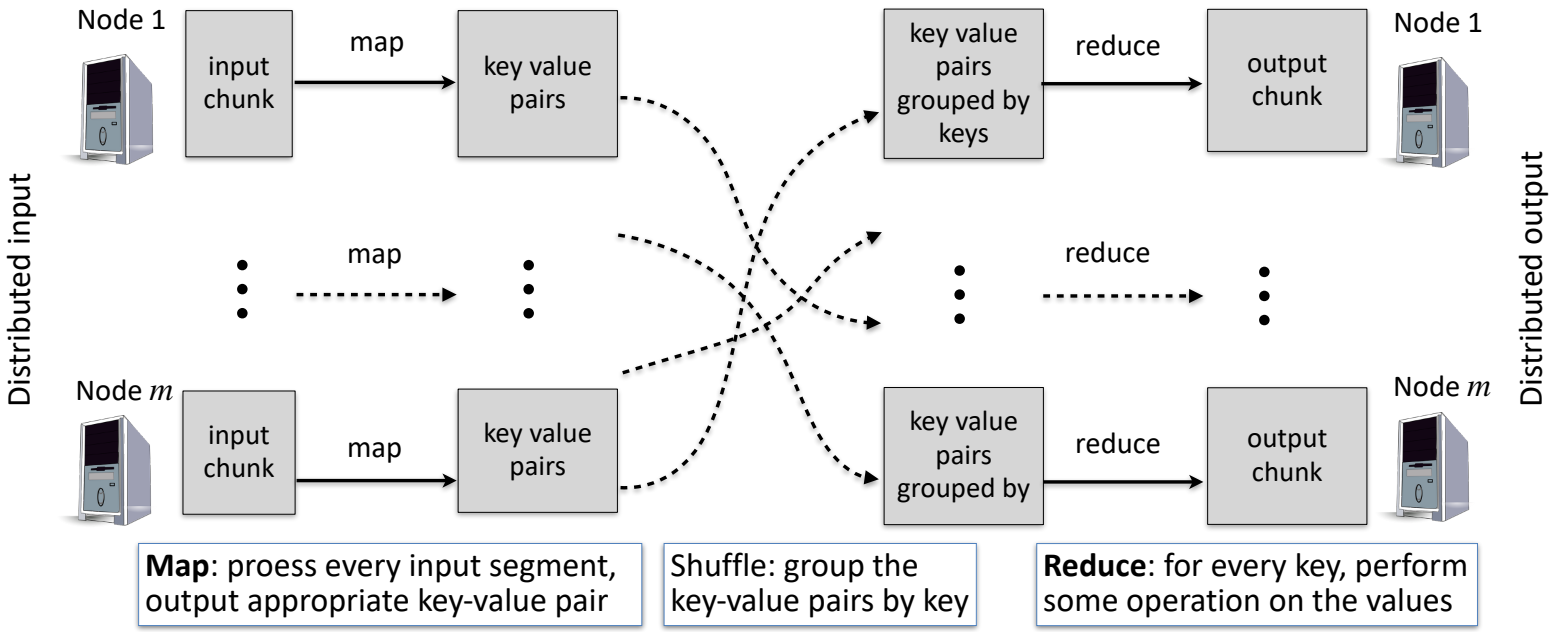


Map: for every word w, output the key-value pair $(w,1)$

Shuffle: group the key-value pairs by key

Reduce: for every key, sum the values

The MapReduce Paradigm



The user (programmer) needs to write the **map** and the **reduce** functions

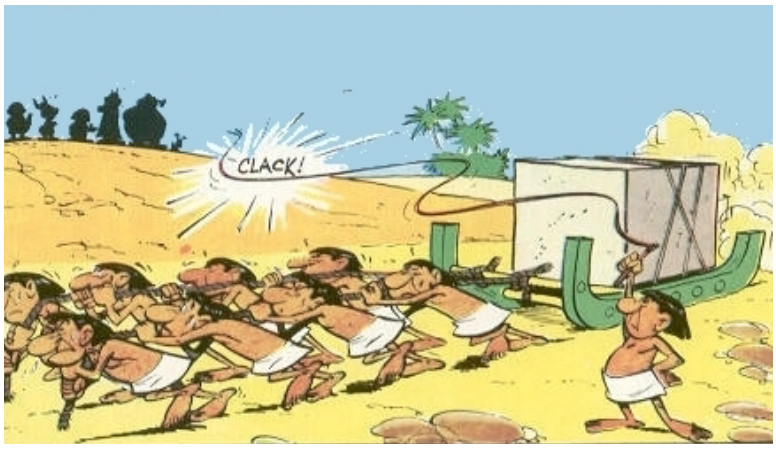
Apache Hadoop

An open source MapReduce framework

Hadoop

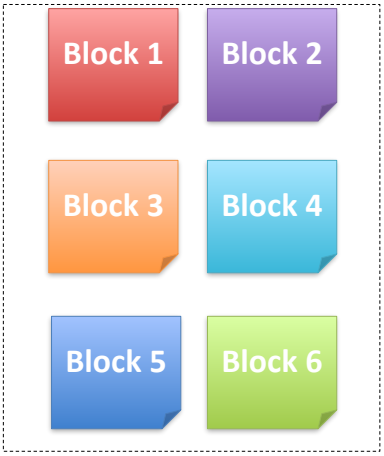
Hadoop

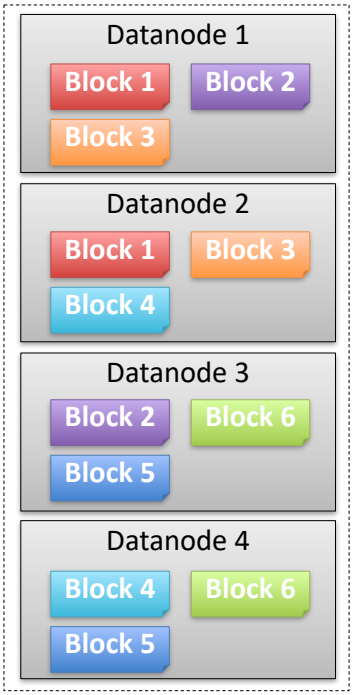
- Two main components
  - **Hadoop Distributed File System** (HDFS): to store data
  - **MapReduce engine**: to process data
- Master – slave architecture using commodity servers



- The HDFS
  - Master: Namenode
  - Slave: Datanode
- MapReduce
  - Master: JobTracker
  - Slave: TaskTracker

HDFS: Blocks



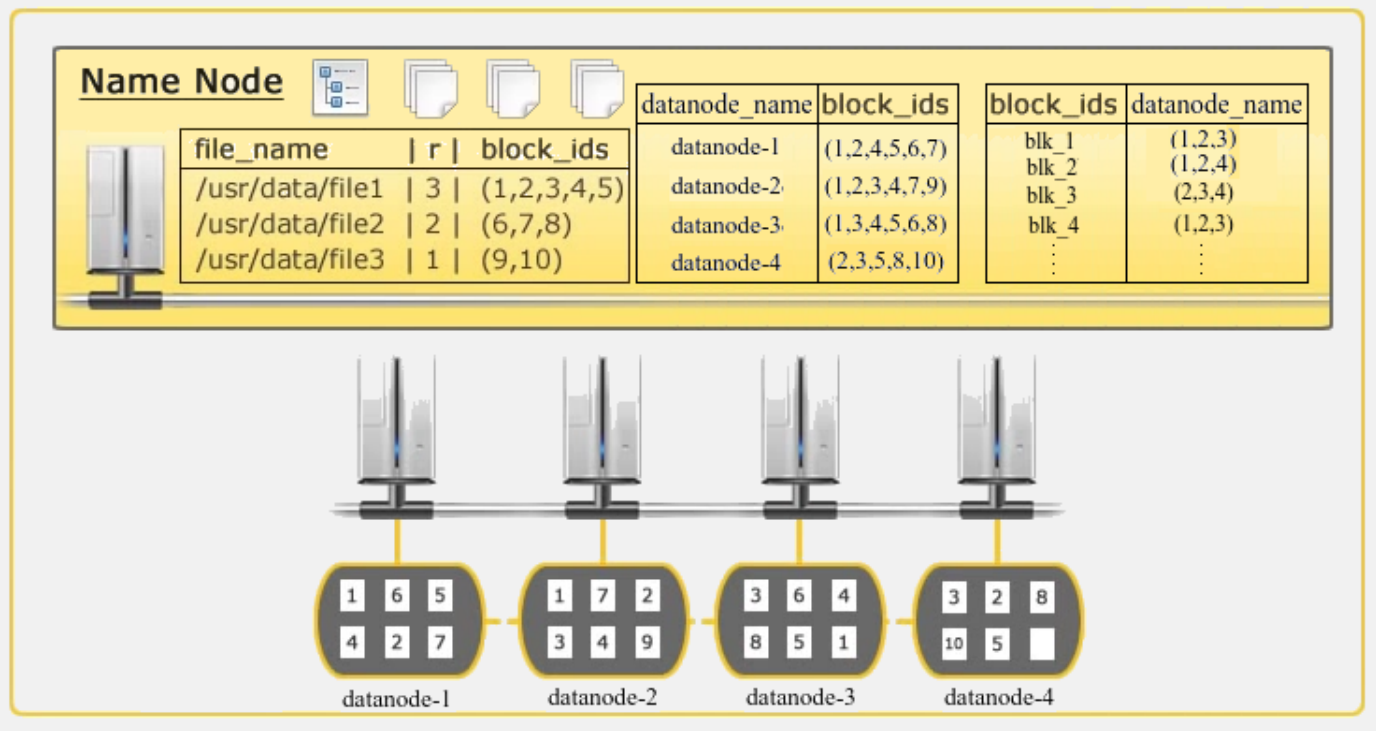


- Runs on top of existing filesystem
- Blocks are 64MB (128MB recommended)
- Single file can be > any single disk
- POSIX based permissions
- Fault tolerant

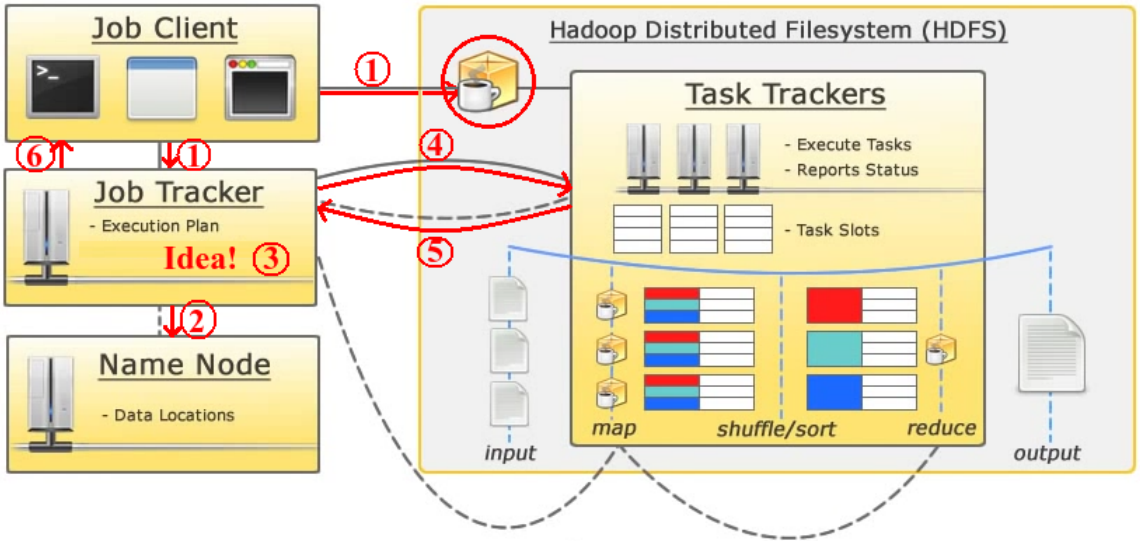
HDFS: Namenode and Datanode

- Namenode
  - Only one per Hadoop Cluster
  - Manages the filesystem namespace
  - The filesystem tree
  - An edit log
  - For each block block i, the datanode(s) in which block i is saved
  - All the blocks residing in each datanode
- Secondary Namenode
  - Backup namenode
- Datanode
  - Many per Hadoop cluster
  - Controls block operations
  - Physically puts the block in the nodes
  - Does the physical replication

HDFS: an example



MapReduce: JobTracker and TaskTracker



1. JobClient submits job to JobTracker; Binary copied into HDFS
2. JobTracker talks to Namenode
3. JobTracker creates execution plan
4. JobTracker submits work to TaskTrackers
5. TaskTrackers report progress via heartbeat
6. JobTracker updates status

Fault Tolerance

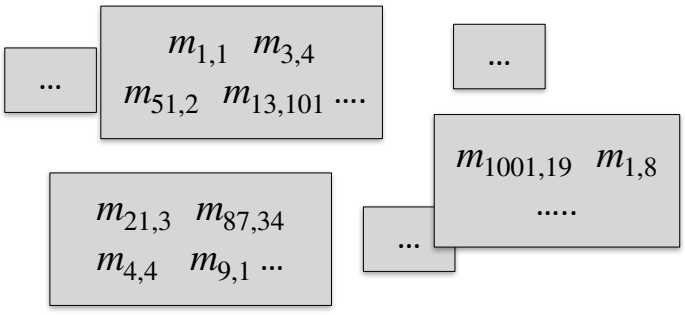
- If the master fails
  - MapReduce would fail, have to restart the entire job
- A map worker node fails
  - Master detects (periodic ping would timeout)
  - All the map tasks for this node have to be restarted
    - Even if the map tasks were done, the output *were* at the node
- A reduce worker fails
  - Master sets the status of its currently executing reduce tasks to idle
  - Reschedule these tasks on another reduce worker

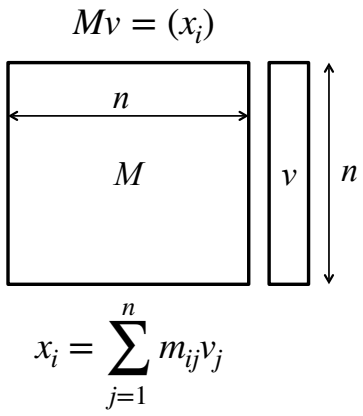
Some algorithms using MapReduce

Using MapReduce

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)

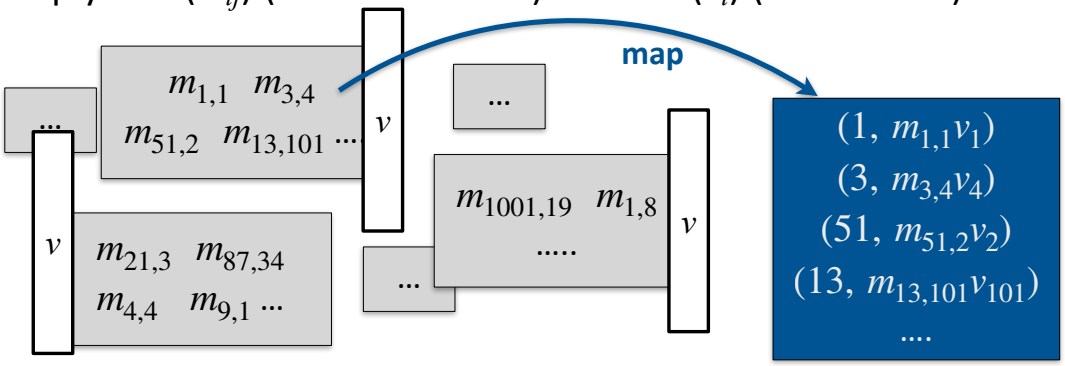




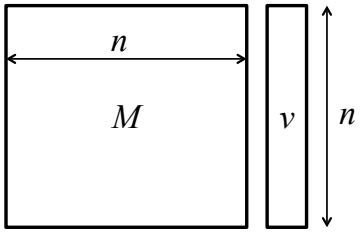
- If $n = 1000$ (small enough), no need of MapReduce!
- When n is very large, how are the entries stored (possibly in a distributed filesystem)?
  - May not be in an order convenient for making any assumption
  - We will not assume anything about the arrangement of the entries
  - Approach: the map should consider each m_{ij}, one at a time

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)



$$Mv = (x_i)$$



$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Need all $m_{ij}v_j$ values together for the same i

So, i needs to be the **key**

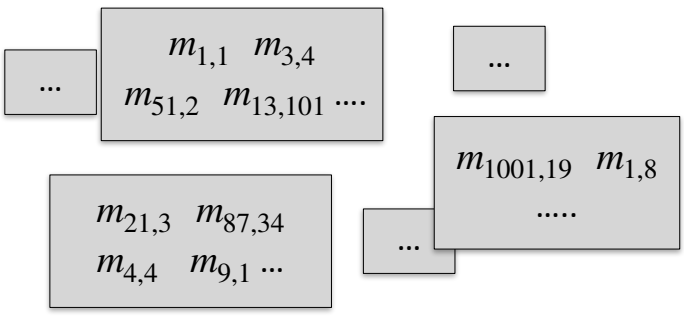
Case 1: Large n, M does not fit into main memory, but v does

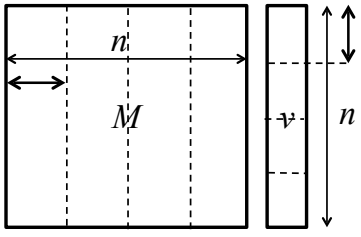
- Since v fits into main memory, all of v can be copied to every map worker
- ***Map***: for each matrix element m_{ij}, emit key value pair $(i,\ m_{ij}v_j)$
- *Shuffle*: groups all $m_{ij}v_j$ values together for the same i
- ***Reduce***: sum $m_{ij}v_j$ for all j for the same i

Matrix – Vector Multiplication

- Multiply $M = (m_{ij})$ (an $n \times n$ matrix) and $v = (v_i)$ (an n-vector)

$$Mv = (x_i)$$





$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Case 2: Very large n, even v does not fit into main memory of the nodes

- For every map, many accesses to disk (for parts of v) required!
- Solution:
  - Partition v so that each partition of v fits into memory
  - Take dot product of one partition of v and the corresponding partition of M
  - *Map* and *reduce* same as before, but several times to complete the computation

Relational Algebra

- Relation $R(A_1, A_3, \ldots, A_n)$ is a relation with attributes A_i
- Schema: set of attributes
- Operations
  - Selection on condition C: apply C on each tuple in R, output only those which satisfy C
  - Projection on a subset S of attributes: output the components for the attributes in S
  - Union, Intersection, Join…

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

Example: Links between URLs

URL1	*URL2*
url1	url2
url2	url1
url3	url5
url1	url3

Example: A relation or table **people_data**

Attributes

rows (records / tuples)

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Selection: example

select * from people_data where **age < 30**

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Projection: example

select **name**, **phone** from people_data

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	29372933	29
4892-4877	Sergio Marquina	879 CED Street	89378277	42
9092-2817	Andrés de Fonollosa	15A First Avenue	92891091	46
8827-3388	Anibal Cortes	76 Second Lane	78399022	19

Name	Phone
Silene Oliveira	29372933
Sergio Marquina	89378277
Andrés de Fonollosa	92891091
Anibal Cortes	78399022

(Inner) Join: example

people_data

Unique_ID	Name	Address	Phone	Age
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29
4892-4877	Sergio Marquina	879 CED Street	**89378277**	42
9092-2817	Andrés de Fonollosa	15A First Avenue	**92891091**	46
8827-3388	Anibal Cortes	76 Second Lane	**78399022**	19

car_data

Car_number	Phone
XBT 2301	**29372933**
BGA 2891	**78399022**
NMAH 938	**29372933**
MGB 7829	**68279482**

people_data **natural join** car_data

Unique_ID	Name	Address	Phone	Age	Car_number
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29	XBT 2301
2839-3847	Silene Oliveira	123 ABC Road	**29372933**	29	NMAH 938
8827-3388	Anibal Cortes	76 Second Lane	**78399022**	19	BGA 2891

Selection using MapReduce

- SQL: select * from R where C
- Trivial using MapReduce
- Map: For each tuple t in R, test if t satisfies condition C. If so, produce the key-value pair (t, 1).
- Reduce: The identity function. It simply passes each key t to the output.

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

Union using MapReduce

- Union of two relations R and S
- Suppose R and S have the same schema
- Map tasks are generated from chunks of both R and S
- Map: For each tuple t, produce the key-value pair $(t, 1)$
- Reduce: Only need to remove duplicates
  - For all key t, there would be either one (if the tuple is present in one relation) or two values (if the tuple is present in both)
  - Output t in either case

R

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyz	1	true
xyz	def	1	false
bcd	def	2	true

S

$Attr_1$	$Attr_2$	$Attr_3$	$Attr_4$
xyz	abc	1	true
abc	xyw	1	false
xyz	def	1	false
bcd	def	2	true

Natural join using MapReduce

- Join $R(A, B)$ with $S(B, C)$ on attribute B
- Map:
  - For each tuple $t = (a,b)$ of R, emit key value pair $(b, (R, a))$
  - For each tuple $t = (b,c)$ of S, emit key value pair $(b, (S, c))$
- Reduce:
  - Each key b would be associated with a list of values that are of the form (R, a) or (S, c)
  - Construct all pairs consisting of one with first component R and the other with first component S, say (R,a) and (S,c). The output from this key and value list is a sequence of key-value pairs

R

A	B
x	m
y	n
z	p
w	q

S

B	C
m	1
p	3
q	4
r	7

$R \bowtie S$

A	B	C
x	m	1
z	p	3
w	q	4

Grouping and Aggregation using MapReduce

- Group and aggregate on a relation $R(A, B)$ using aggregation function $\gamma(B)$, group by A
- Map:
  - Grouping comes for free with MapReduce, so the attribute to "group by" needs to be the key
  - Map is essentially the identity function
  - For each tuple $t = (a,b)$ of R, emit key value pair (a, b)
- Reduce:
  - Reduce does the aggregation part
  - For the group $\{(a, b_1), \ldots, (a, b_m)\}$ represented by a key a, apply γ to obtain $b_a = \gamma(b_1, \cdots, b_m)$
  - Output (a, b_a)

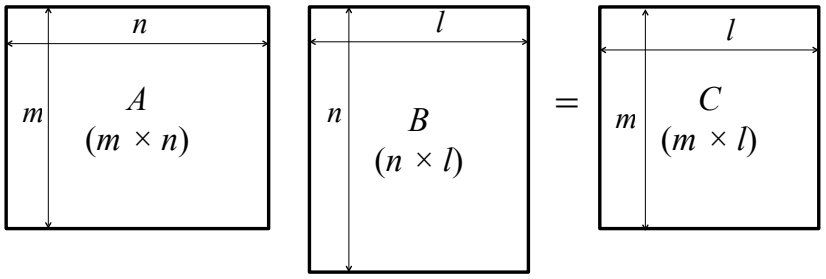
R

A	B
x	2
y	1
z	4
z	1
x	5

```
select A, sum(B)
from R group by A;
```

A	$sum(B)$
x	7
y	1
z	5

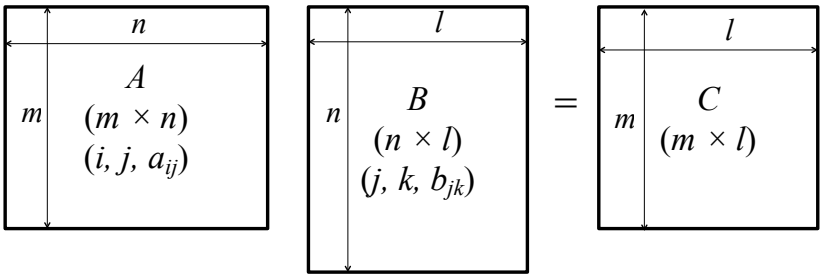
Matrix multiplication using MapReduce



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

- Think of a matrix as a relation with three attributes
- For example matrix A is represented by the relation $A(I, J, V)$
  - For every non-zero entry (i, j, a_{ij}), the row number is the value of I, column number is the value of J, the entry is the value in V
  - Also advantage: usually most large matrices would be sparse, the relation would have less number of entries
- The product is *analogous to* a natural join followed by a grouping with aggregation

Matrix multiplication using MapReduce



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

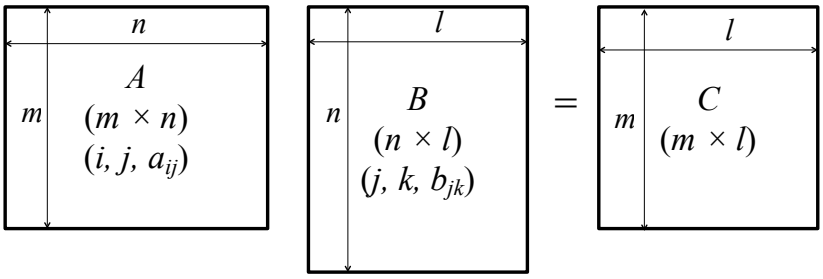
- Natural join of (I, J, V) and (J, K, W) produces tuples $(i, j, k, a_{ij}, b_{jk})$
- Map:
  - For every (i, j, a_{ij}), emit key value pair $(j, (A, i, a_{ij}))$
  - For every (j, k, b_{jk}), emit key value pair$(j, (B, k, b_{jk}))$
- Reduce:

  for each key j

  for each value (A, i, a_{ij}) and (B, k, b_{jk})

  produce a key value pair $((i, k), (a_{ij} b_{jk}))$ [could also output $i, k, a_{ij} b_{jk}$, but doing some work in advance]

Matrix multiplication using MapReduce



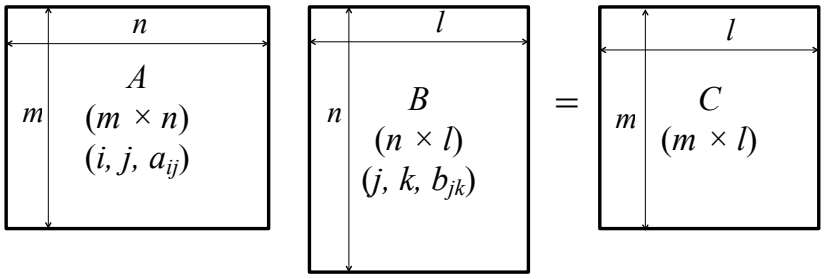
$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

- First MapReduce process has produced key value pairs $((i, k), (a_{ij} b_{jk}))$
- Another MapReduce process to group and aggregate
  - Map: identity, just emit the key value pair $((i, k), (a_{ij} b_{jk}))$
  - Reduce:

    for each key (i, k)

    produce the sum of the all the values for the key: $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$

Matrix multiplication using MapReduce: Method 2



$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Idea: the final keys have to be ik pairs, so create all possible ik combinations as keys upfront

- A method with one MapReduce step
- Map:
  - For every (i, j, a_{ij}), emit for all $k = 1,\ldots, l$, the key value $((i,k), (A, j, a_{ij}))$
  - For every (j, k, b_{jk}), emit for all $i = 1,\ldots, m$, the key value $((i,k), (B, j, b_{jk}))$
- Reduce:

  for each key (i,k)

  sort values (A, j, a_{ij}) and (B, j, b_{jk}) by j to group them by j

  for each j multiply a_{ij} and b_{jk}

  sum the products for the key (i,k) to produce $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$

May not fit in main memory and may have to perform expensive external memory sort. If so, method 1 is more practical.

Combiners

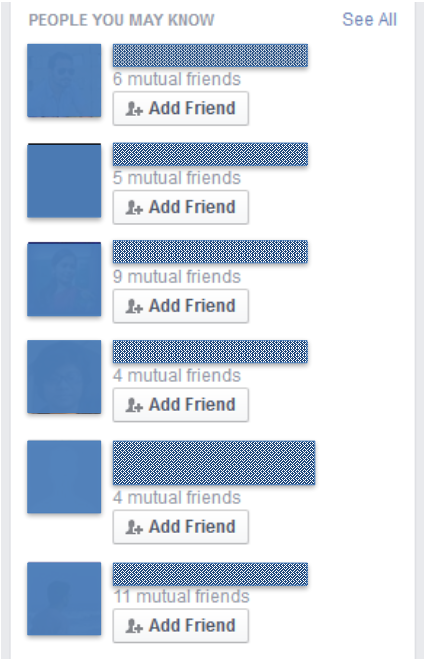
- If the aggregation operator in the reduce algorithm is associative and commutative
  - Associative: $(a * b) * c = a * (b * c)$
  - Commutative: $(a * b) = (b * a)$
  - It does not matter in which order we aggregate
- We can do some work for reduce in the map step
- Example: the word count problem
  - Instead of emitting $(w,1)$ for each word
  - We can emit (w, n) for each document or each map task where n is the number of times the word w appears in the whole input chunk

A problem: find people you may know

A Social Network

- Task: suggest friends
- There are several factors to consider
- A very important one is by number of mutual friends
- How to compute a list of suggestions?
- Consider a simple criterion:

If X and Y have at least 3 mutual friends then Y is a suggested friend of X



The Data

- For every member, there is a list of friends

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

Naïve approach:

- For every friend X of A $O(n)$
  - For every friend Y of X $O(n)$
    - Compute number of mutual friends between Y and A
    - Intersect the friend-list of Y and friend-list of A $O(n)$

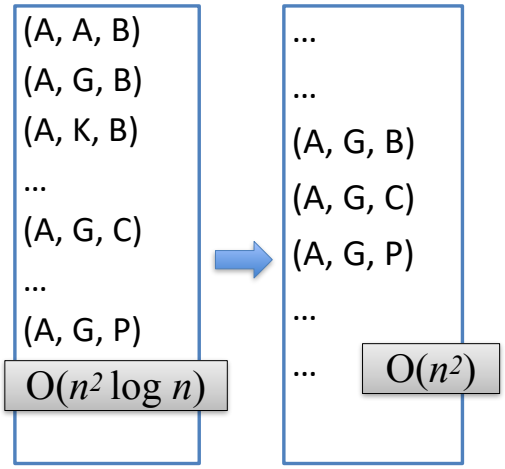
N members.
Every member has n friends.
$N \gg n$.

$O(n^3)$ per member. Total $O(N\, n^3)$!!

A better method

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

- For each friend X of A $O(n)$
  - For each friend Y of X $O(n)$
    - Write: (A, Y, X) $O(n^2)$
- What do we get?
- Now sort by first two entries
  - All the (A,G)'s are together
- For each (A, Y), count how many times it occurs
  - Number of mutual friends between A and Y



$O(n^2 \log n)$ per member

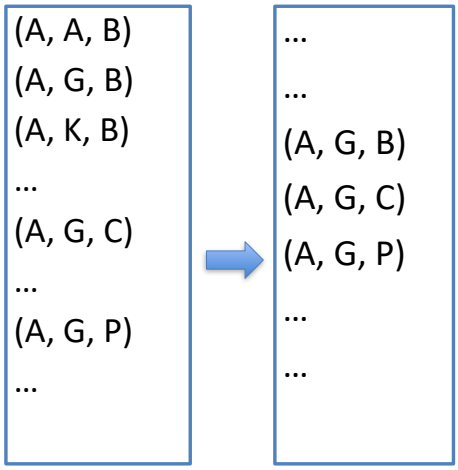
Challenges

A	→	B	C	L	…
B	→	A	G	K	…
C	→	A	G	M	…

- For each friend X of A
  - For each friend Y of X
    - Write: (A, Y, X)
- Now sort by first two entries
  - All the (A,G)'s are together
- For each (A, Y), count how many times it occurs

Data may be too huge, distributed

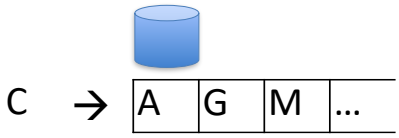
Friendlist of A in one part of storage
Friendlist of friend of A in another part



Using MapReduce

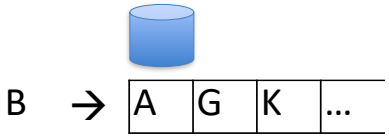
Map:

- For each member X, fetch his/her friendlist $Y_1, Y_2, \dots, Y_n$
  - For each pair (Y_1, Y_2), emit the key value pair $\{(Y_1, Y_2), X\}$



Map

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
...	...



Map

Key	Value
(A,G)	B
(A,K)	B
(G,K)	B
...	...

Using MapReduce

Reduce:

- For each Key = (X,Y), determine Count(X,Y) = # of mutual friends

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
...	...

Key	Value
(A,G)	B
(A,K)	B
(G,K)	B
...	...

Shuffle and sort

Key	Value
(A,G)	B
(A,G)	C
(A,G)	P
...	...

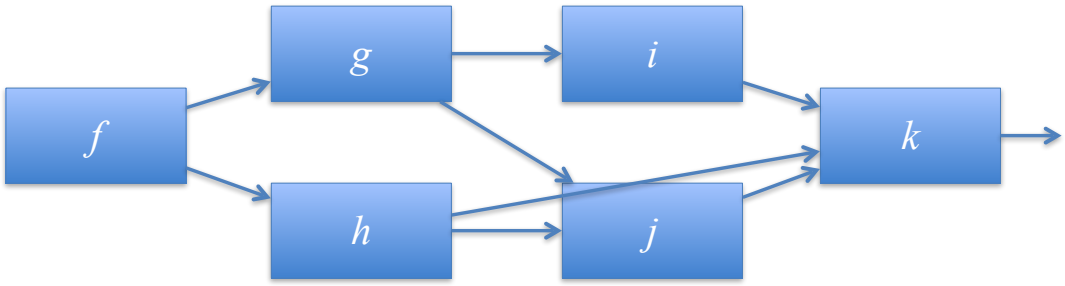
Reduce → Count(A,G)

Key	Value
(S,T)	Q
(S,T)	V
(S,T)	P
...	...

Reduce → Count(S,T)

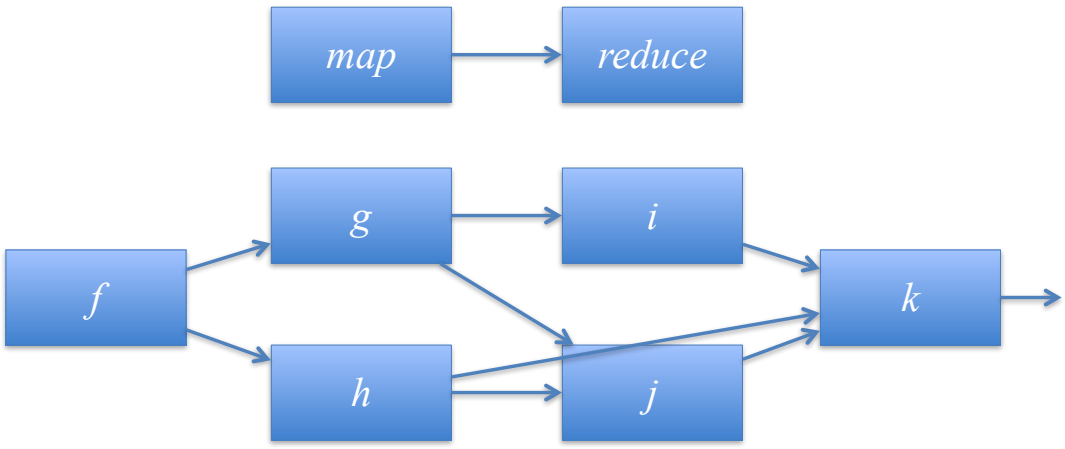
Extensions of MapReduce

- MapReduce: two-step workflow
- Extension: any collection of functions: acyclic graph representing the workflow



- Each function of the workflow can be executed by many tasks
- A master controller divides the work among the tasks
- Each task's output goes to the successor as input
- Two experimental systems: Clustera (University of Wisconsin), Hyracks (University of California at Irvine)
- Similarly, recursive use of MapReduce

Cost of MapReduce based algorithms



- Computation in the nodes → Computation cost
- Transferring of data through network → Communication cost
- Communication cost is most often the bottleneck. Why?

Why communication cost?

- The algorithm executed by each task is usually very simple, often linear in the size of its input
- Network speed within cluster ∼ 1Gigabit/s
  - But processor executes simple map or reduce tasks even faster
  - In a cluster several network transfers may be required at the same time → overload
- Data typically stored in disk
  - Reading the data into main memory may take more time than to process it in map or reduce tasks

Communication cost

- Communication cost of a task = The size of the input to the task
- Why size of the input?
  - Most often size of the input ≥ size of the output
  - Why?
  - Because unless summarized, the output can't be used much
  - If output size is large, then it is the input for another task (counting input size suffices)

History

MapReduce was first popularized as a programming model in 2004 by Jeffery Dean and Sanjay Ghemawat of Google (Dean & Ghemawat, 2004). In their paper, "MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS," they discussed Google's approach to collecting and analyzing website data for search optimizations. Google's proprietary MapReduce system ran on the Google File System (GFS).

MapReduce helped in two key tasks Google needed to perform day in and day out at that time. One is creation of an inverted index for search, and another is computing PageRank.

Apache, the open source organization, began using MapReduce in the "Nutch" project, which is an open source web search engine that still is active today. Hadoop began as a subproject in the Apache Lucern project, which provides text search capabilities across large databases.

In 2006, Doug Cutting, an employee of Yahoo!, designed Hadoop, naming it after his son's toy elephant. While it was originally a subproject, Cutting released Hadoop as an open source Apache project in 2007. (Hadoop, 2011). In 2008, Hadoop became a top level project at Apache. On July 2008, an experimental 4000 node cluster was created using Hadoop, and in 2009 during a performance test, Hadoop was able to sort a terabyte of data in 17 hours.

https://mindmajix.com/mapreduce/history-and-advantages-of-hadoop-mapreduce-programming

References and acknowledgements

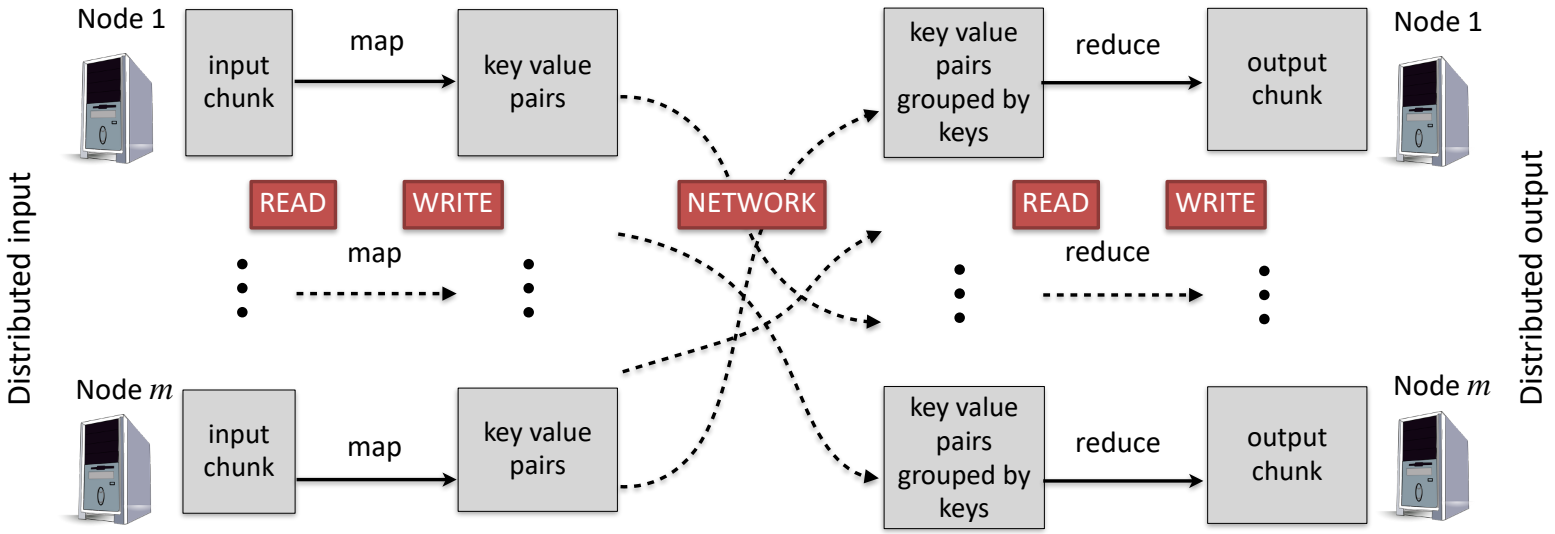
- Primary reference: Mining of Massive Datasets, by Leskovec, Rajaraman and Ullman, Chapter 2
- Source for some slides on Hadoop: Dwaipayan Roy

Apache Spark

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

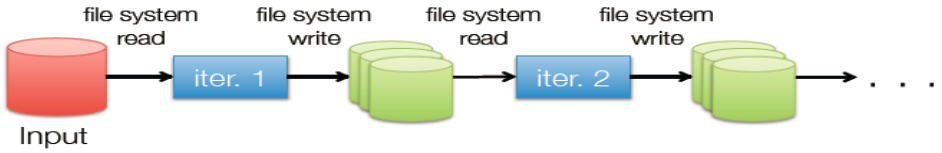
The MapReduce Paradigm

Vanilla MapReduce is based on acyclic data flow



- *Inefficient* for applications that repeatedly reuse a working set of data. For example:
  - Iterative algorithms (machine learning, graphs)
  - Interactive data mining tools (functionality similar to R, Python): with Hadoop, apps need to reload data from stable storage for each query

Limitations of Vanilla MapReduce



- Map-reduce is fine for one-pass computation, but inefficient for multi-pass algorithms
  - Examples: k-means, PageRank
- No efficient mechanism for data sharing
  - State between steps goes to distributed file system
  - Slow due to replication & disk storage
- Not interactive or flexible
  Have to write *map* and *reduce* for any task
- Commonly spend 90% of time doing I/O

Apache Spark

- Goals
  - Extend the MapReduce model to better support two common classes of analytics apps:
    - Iterative algorithms (machine learning, graph)
    - Interactive data mining
  - Enhance programmability

- Approach: **Resilient Distributed Dataset (RDD)**
  - Allow apps to keep working sets in memory as long as possible
  - Retain the advantages of MapReduce (fault tolerance, data locality, scalability)
  - Support a wide range of applications

RDDs

- An RDD is a read-only , partitioned collection of records
- Can only be created by :
  - (1) Data in stable storage
  - (2) Other RDDs (transformation , lineage)
- Each RDD include:
  - 1) A set of partitions (atomic pieces of datasets)
  - 2) A set of dependencies on parent RDDs
  - 3) A function for computing the dataset based on its parents
  - 4) Metadata about its partitioning scheme
  - 5) Data placement
- An RDD has enough information about how it was derived from other datasets(its lineage)
  - Fault tolerance
- Users can control two aspects of RDDs
  - (1) Persistence (in RAM, reuse)
  - (2) Partitioning (hash, range, [<k, v>])
- Transformations are lazy, they don't compute right away. Just remember the transformations applied to datasets(lineage). Only compute when an action require.

Programming languages

- Spark is implemented in Scala
- Allows interactive use from Scala interpreter
- Scala
  - High-level language for JVM
  - Object-oriented + Functional programming
  - Statically typed
  - Comparable in speed to Java
  - No need to write types due to type inference
  - Interoperates with Java
  - Can use any Java class, inherit from it, etc;
  - Can also call Scala code from Java
- Python (Pyspark)
  - Slower than Java / Scala in performance (about 2-3 times)
  - But gaining a lot of popularity because python is used widely nowadays

Memory Management

- Spark provides three options for persist RDDs:
  - In-memory storage as deserialized Java Objects
    - fastest, JVM can access RDD natively
  - In-memory storage as serialized data
    - space limited, choose another efficient representation, lower performance cost
  - On-disk storage
    - RDD too large to keep in memory, and costly to recompute
    - Defaults to (almost) vanilla MapReduce for certain tasks

References and Acknowledgments

- Credits for some of the slides go to Wen Zhiguang and Jiaul Paik

Basics of Information Retrieval

Term-Document Matrix, Inverted Index and Boolean Retrieval

Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

Information Retrieval

User needs some information.

An information retrieval system tries to bridge this gap.

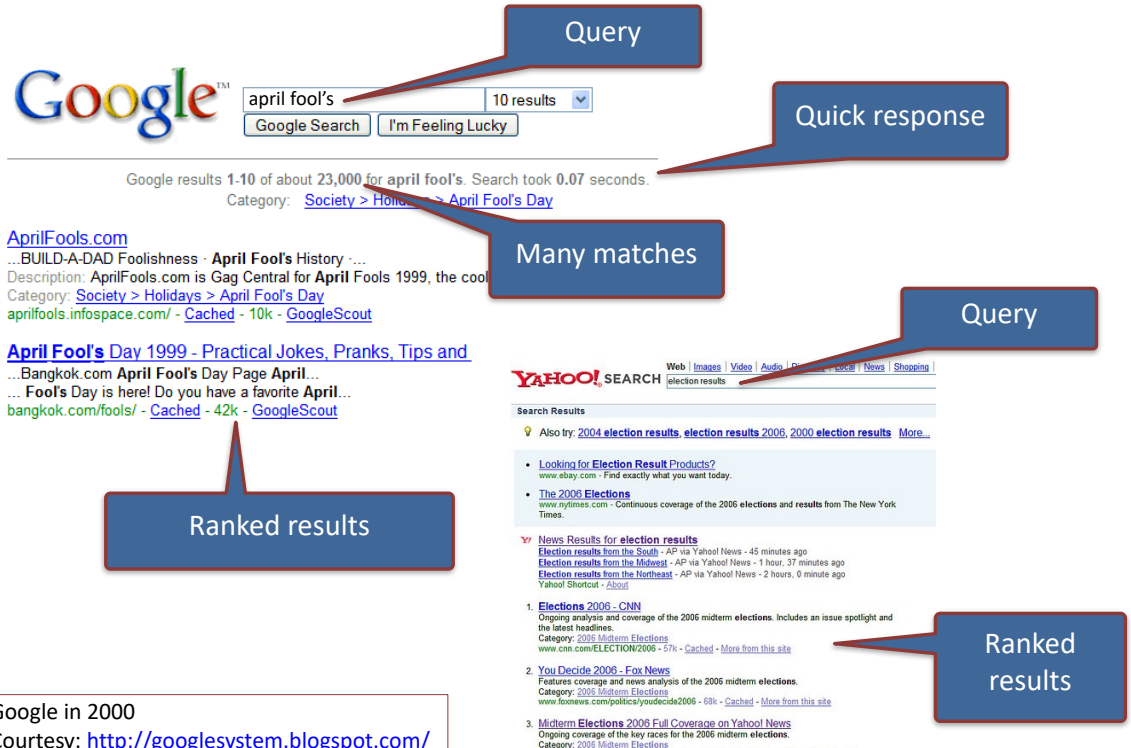
Assumption: the required information is present somewhere.

The goal of an information retrieval system is to satisfy user's information need.

Basic example

User expresses the information need in the form of a query.

The system returns a (ranked) list of results.



Google in 2000
Courtesy: http://googlesystem.blogspot.com/2007/12/google-in-2000.html

Yahoo search in 2006
Courtesy: https://www.searchenginewatch.com/2006/11/08/in-the-election-results-race-yahoos-the-winner/

Collection and Documents

The curse of the black pearl Ship Captain Jack Sparrow Caribbean Elizabeth Gun Fight	**Finding Nemo** Ocean Fish Nemo Reef Animation	**Tintin** Ocean Animation Ship Captain Haddock Tintin	**Titanic** Ship Rose Jack Atlantic Ocean England Sink Captain
The Dark Knight Bruce Wayne Batman Joker Harvey Gordon Gun Fight Crime	**Skyfall** 007 James Bond MI6 Gun Fight	**Silence of the Lambs** Hannibal Lector FBI Crime Gun Cannibal	**The Ghost Ship** Ship Ghost Ocean Death Horror

- Document: unit of retrieval
- Collection: the group of documents from which we retrieve
  - Also called the ***corpus*** (a body of text)

Boolean retrieval

- Find all documents containing a word w
- Find all documents containing a word w_1 but not containing the word w_2
- Queries in the form of any Boolean expression
- Query: **Jack**

Boolean retrieval

- Find all documents containing a word w
- Find all documents containing a word w_1 but not containing the word w_2
- Queries in the form of any Boolean expression
- Query: **Jack**

Term – document matrix

	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1	0	1	1	0	0	0	1
Jack	1	0	0	1	0	0	0	0
Bond	0	0	0	0	0	1	0	0
Gun	1	0	0	0	1	1	1	0
Ocean	1	1	1	1	0	0	0	1
Captain	1	0	1	1	0	0	0	0
Batman	0	0	0	0	1	0	0	0
Crime	0	0	0	0	1	0	1	0

- The entry (*w*, *d*) = 1 if and only if the word *w* is present in document *d*
- Terms are dimensions of this matrix (*units of index; we will discuss later*)
- Commonly called the ***term – document matrix***
- Term and word are not same, though often words are used as terms

Boolean retrieval

	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1	0	1	1	0	0	0	1
Jack	1	0	0	1	0	0	0	0
Bond	0	0	0	0	0	1	0	0
Gun	1	0	0	0	1	1	1	0
Ocean	1	1	1	1	0	0	0	1
Captain	1	0	1	1	0	0	0	0
Batman	0	0	0	0	1	0	0	0
Crime	0	0	0	0	1	0	1	0

- Query: Jack
- Results: 10010000

Boolean retrieval

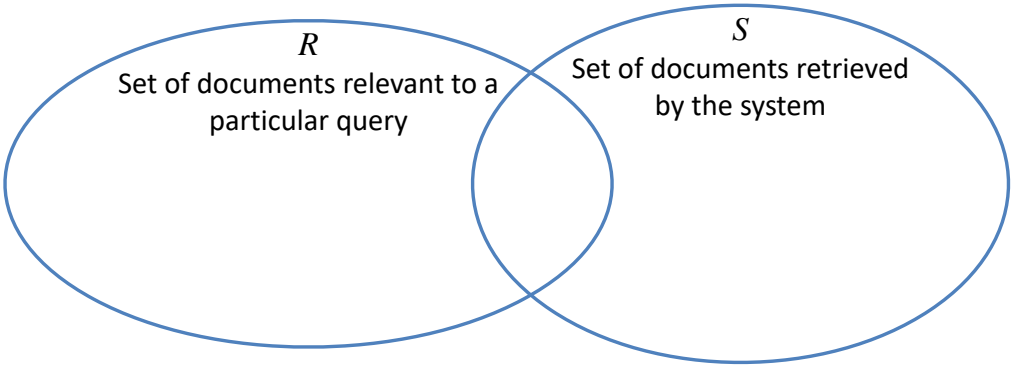
	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1	0	1	1	0	0	0	1
Jack	1	0	0	1	0	0	0	0
Bond	0	0	0	0	0	1	0	0
Gun	1	0	0	0	1	1	1	0
Ocean	1	1	1	1	0	0	0	1
Captain	1	0	1	1	0	0	0	0
Batman	0	0	0	0	1	0	0	0
Crime	0	0	0	0	1	0	1	0

- Query: Captain AND Gun
- Results: 10110000 && 10001110 = 10000000

Query and relevant documents

- Query: given by user, represents the *information need*
  - Information need is the topic, conceptually what the user wants to know
  - Query is the representation of information need that the user conveys to the retrieval system
- Relevant document: a document that satisfies the information need, as perceived by the user
  - Merely matching the query terms does not mean a document is relevant
  - A relevant document must satisfy the actual information need

Precision and recall



- What fraction of the returned results are relevant?

  Precision $= \dfrac{|R \cap S|}{|S|}$

- What fraction of the relevant documents in the collection were returned by the system?

  Recall $= \dfrac{|R \cap S|}{|R|}$

What if the collection is “large”?

	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1	0	1	1	0	0	0	1
Jack	1	0	0	1	0	0	0	0
Bond	0	0	0	0	0	1	0	0
Gun	1	0	0	0	1	1	1	0
Ocean	1	1	1	1	0	0	0	1
Captain	1	0	1	1	0	0	0	0
Batman	0	0	0	0	1	0	0	0
Crime	0	0	0	0	1	0	1	0

- About 1 million documents (still not so large)
- About 500,000 distinct terms
- A term – document matrix of 500,000 × 1 million Boolean entries ~ 500GB

What if the collection is “large”?

	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1	0	1	1	0	0	0	1
Jack	1	0	0	1	0	0	0	0
Bond	0	0	0	0	0	1	0	0
Gun	1	0	0	0	1	1	1	0
Ocean	1	1	1	1	0	0	0	1
Captain	1	0	1	1	0	0	0	0
Batman	0	0	0	0	1	0	0	0
Crime	0	0	0	0	1	0	1	0

Sparse matrix ⟶ inverted index

	Black pearl	Finding Nemo	Tintin	Titanic	Dark Knight	Skyfall	Silence of lambs	Ghost ship
Ship	1		1	1				1
Jack	1			1				
Bond						1		
Gun	1				1	1	1	
Ocean	1	1	1	1				1
Captain	1		1	1				
Batman					1			
Crime					1		1	

- In reality term – document matrices are very sparse
- Most terms are NOT present in most documents
- For every term, store only the documents where the term is present

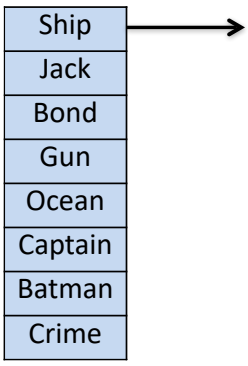
Sparse matrix ⟶ inverted index

	1 Black pearl	2 Finding Nemo	3 Tintin	4 Titanic	5 Dark Knight	6 Skyfall	7 Silence of lambs	8 Ghost ship
Ship	1		1	1				1
Jack	1			1				
Bond						1		
Gun	1				1	1	1	
Ocean	1	1	1	1				1
Captain	1		1	1				
Batman					1			
Crime					1		1	

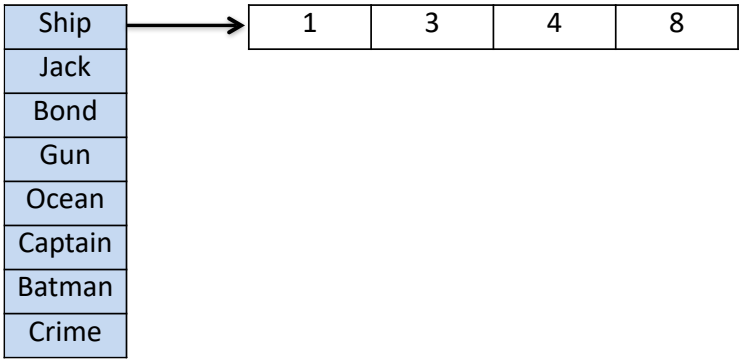
- Represent documents by document IDs

Sparse matrix ⟶ inverted index

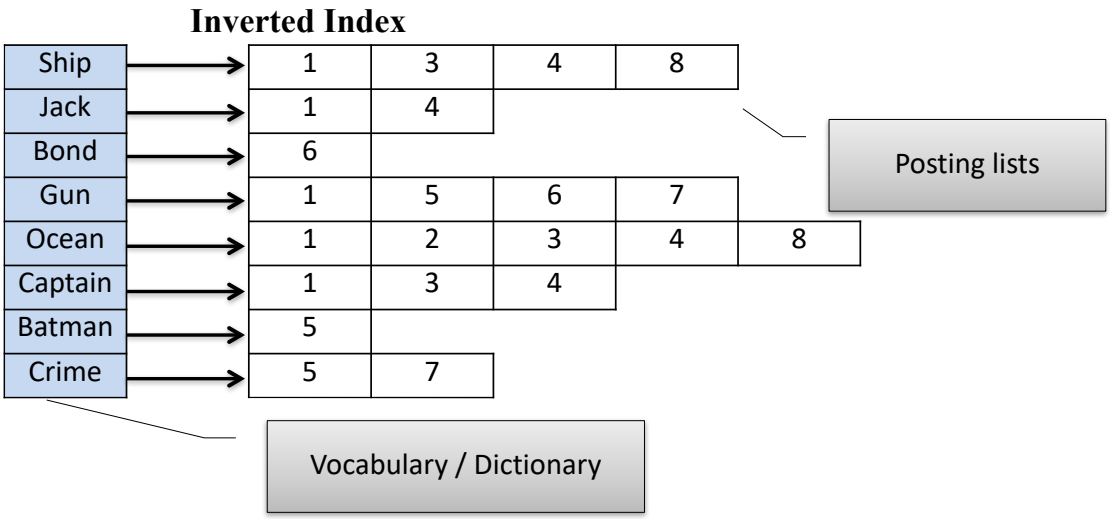
1 Black pearl	2 Finding Nemo	3 Tintin	4 Titanic	5 Dark Knight	6 Skyfall	7 Silence of lambs	8 Ghost ship



Sparse matrix ⟶ inverted index



Sparse matrix ⟶ inverted index



Creating an inverted index: basic idea

- For each document, write out pairs (term, docid)
- Sort by term, then group

Term	docId
Ship	1
Captain	1
Jack	1
…	…
Ship	3
Tintin	3
…	…
Jack	4
…	…

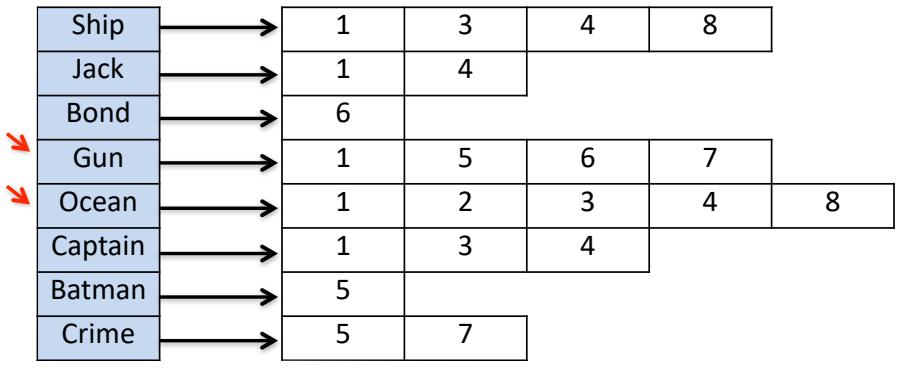
Group by term

Term	docId
…	…
Captain	1
…	…
Jack	1
Jack	4
…	…
Ship	1
Ship	3
…	…

Term	docId	docId	docId
Captain	1	…	
Jack	1	4	…
Ship	1	3	…
…	…		

Boolean query processing

Query: Gun OR Ocean

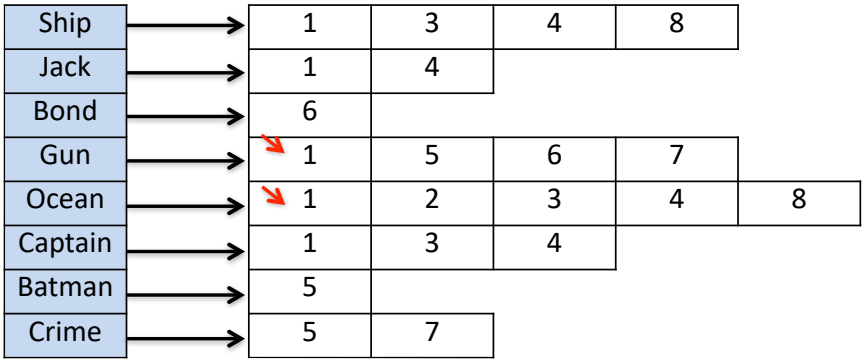


Need to perform **merge union** of the two lists sorted by document ID

Start with a pointer at the beginning of each list

Boolean query processing

Query: Gun OR Ocean

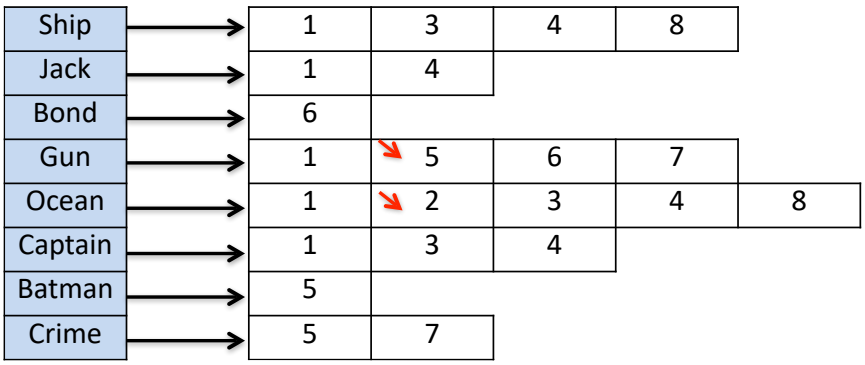


Both doc IDs are same $\Longrightarrow$ add to result list and advance pointers in both lists

Results: 1

Boolean query processing

Query: Gun OR Ocean

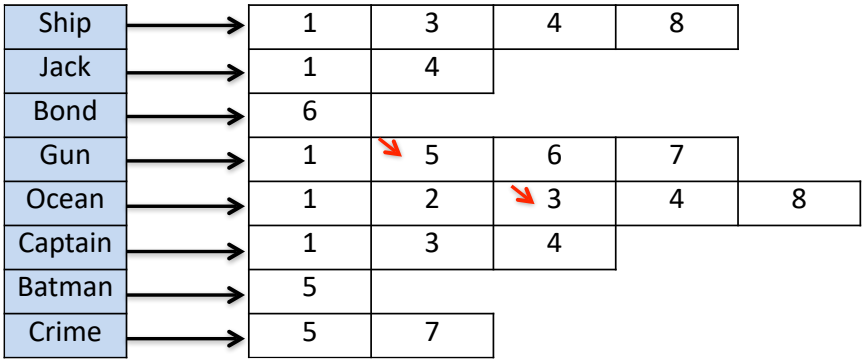


Doc IDs are not same $\Longrightarrow$ add the smaller ID to result list and advance only in that list

Results: 1 2

Boolean query processing

Query: Gun OR Ocean

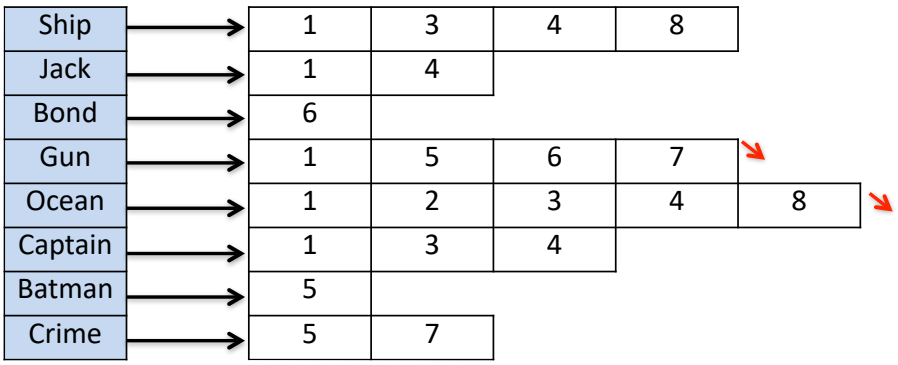


Doc IDs are not same $\Longrightarrow$ add the smaller ID to result list and advance only in that list

Results: 1 2 3

Boolean query processing

Query: Gun OR Ocean



Final result

$O(n)$ algorithm if lists are of length $O(n)$

Merge intersection works in a similar way

Results: 1 2 3 4 5 6 7 8

Boolean retrieval use cases

Westlaw (www.westlaw.com)

- Largest commercial legal document search
- Tens of TB of text data
- Half a million users, million queries a day

Examples of queries:

- Information need: cases about a host's responsibility for drunk guests
- Example query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

The Vector Space Model

Basics of Ranking

- Boolean retrieval models simply return documents satisfying the Boolean condition(s)
  - Among those, are all documents equally "good"?
  - No
- Case: single term query
  - Not all documents containing the term are equally associated with that term
- From Boolean model to term weighting
  - Weight of a term in a document is 1 or 0 in Boolean model
  - Use more granular term weighting
- Weight of a term in a document: represent how much the term is important in the document and vice versa

Term weighting – TF.iDF

- How important is a term t in a document d
- Intuition 1: More times a term is present in a document, more important it is
  - Term frequency (TF)
- Intuition 2: If a term is present in many documents, it is less important particularly to any one of them
  - Document frequency (DF)
- Combining the two: TF.iDF (term frequency × inverse document frequency)
  - Many variants exist for both TF and DF

Formulation of term frequency (TF)

1. Simplest term frequency: Number of times a term t occurs in a document d: $\text{freq}(t, d)$
   - If a term a is present 10 times, b is present 2 times, is a 5 times more important than b in that document?
   - No, importance does not grow linearly with frequency
2. Logarithmically scaled frequency: $1 + \log(\text{freq}(t, d))$, or even $1 + \log(1 + \log(\text{freq}(t, d)))$
   - Still, long documents on same topic would have the more frequency for the same terms
3. Augmented frequency: avoid bias towards longer documents

Half the score for just being present

Rest is a function of frequency

$$TF(t,d) = 0.5 + \frac{0.5 \times \text{freq}(t,d)}{\max\{\text{freq}(w,d) \mid w \in d\}}$$

for all t in d; 0 otherwise

(Inverse) document frequency (iDF)

- Inverse document frequency of a term t

$$\text{iDF}(t) = \log\left[\frac{N}{\text{DF(t)}}\right],$$

where N = total number of documents

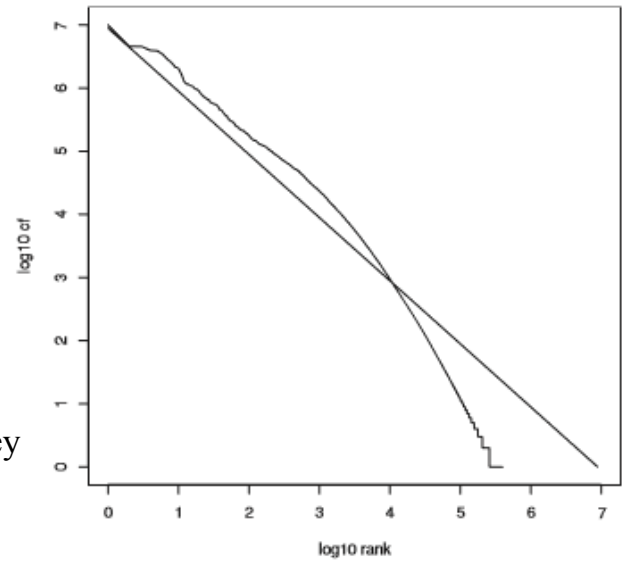
$\text{DF}(t)$ = number of documents in which t occurs

Distribution of terms

- Zipf's law: Let $T = \{t_1, \ldots, t_m\}$ be the terms, sorted by decreasing order of the number of documents in which they occur. Then

$$\text{DF}(t_i) \propto \frac{1}{i}$$

- In other words, $\log \text{DF}(t_i) = \log c + (-1) \cdot \log i$ for some constant c



Zipf's law fitting for Reuter's RCV1 collection

Vector space model

	d_1	d_2	d_3	d_4	d_5	q
diwali	0.5	0	0	0	0	
india	0.2	0.2	0	0.2	0.1	1
flying	0	0.4	0	0	0	
population	0	0	0	0.5	0	
autumn	0	0	1	0	0	
statistical	0	0.3	0	0	0.2	1

Each term represents a dimension

Documents are vectors in the term-space

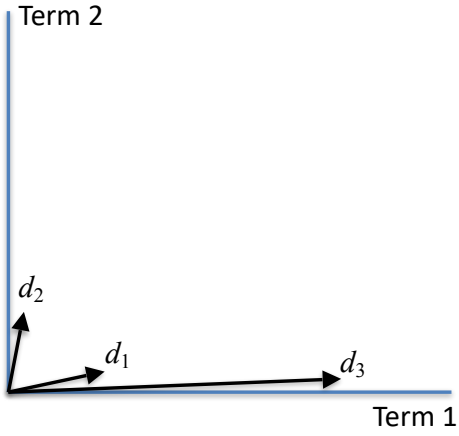
Term-document matrix: a very sparse matrix

Entries are scores of the terms in the documents (Boolean → Count → Weight)

Query is also a vector in the term-space

- Vector similarity: inverse of "distance"
- Euclidean distance?

Problem with Euclidean distance



Problem

- Topic-wise d_3 is closer to d_1 than d_2 is
- Euclidean distance wise d_2 is closer to d_1

Dot product seems to solves this problem

Vector space model

	d_1	d_2	d_3	d_4	d_5	q
diwali	0.5	0	0	0	0	
india	0.2	0.2	0	0.2	0.1	1
flying	0	0.4	0	0	0	
population	0	0	0	0.5	0	
autumn	0	0	1	0	0	
statistical	0	0.3	0	0	0.2	1
$q^T d$	0.2	0.5	0	0.2	0.3	

Each term represents a dimension

Documents are vectors in the term-space

Term-document matrix: a very sparse matrix

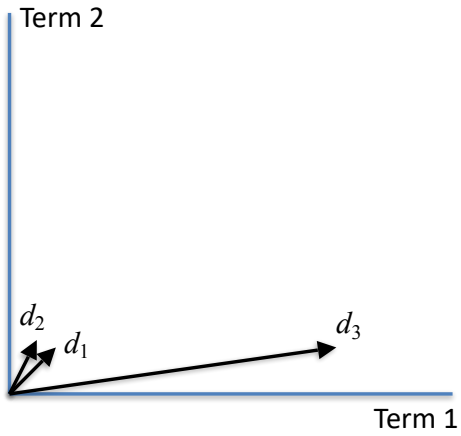
Entries are scores of the terms in the documents (Boolean → Count → Weight)

Query is also a vector in the term-space

Vector similarity: dot product

$$sim(q,d) = q^T d$$

Problem with dot product



Problem

- Topic-wise d_2 is closer to d_1 than d_3 is
- Dot product of d_3 and d_1 is greater because of the length of d_3

- Consider angle
  - Cosine of the angle between two vectors
  - Same direction: 1 (similar)
  - Orthogonal: 0 (unrelated)
  - Opposite direction: -1 (opposite)

Vector space model

	d_1	d_2	d_3	d_4	d_5	q
diwali	0.5	0	0	0	0	
india	0.2	0.2	0	0.2	0.1	1
flying	0	0.4	0	0	0	
population	0	0	0	0.5	0	
autumn	0	0	1	0	0	
statistical	0	0.3	0	0	0.2	1

Each term represents a dimension

Documents are vectors in the term-space

Term-document matrix: a very sparse matrix

Entries are scores of the terms in the documents (Boolean → Count → Weight)

Query is also a vector in the term-space

cosine of the angle between the vectors

$$\text{sim}_{\cos}(q, d) = \cos(\theta_{q,d}) = \frac{q^T d}{\|q\| \|d\|}$$

References

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. "Introduction to Information Retrieval", Cambridge University Press. 2008.

Indexing

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.5 µs)
Processor's clock rate	10^9 per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

Caching

- Accessing data in memory is a few times faster than from disk
- Keep as much information as possible in main memory

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.5 µs)
Processor's clock rate	10^9 per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

Reading from disk

- About 10 MB data in one contiguous chunk on disk
  - One seek (~5 ms) + Read (~ 0.2 s) ≈ 0.2 s
- About 10 MB data in 1000 non- contiguous chunks on disk
  - Thousand seeks (~5 s) + Read (~ 0.2 s) ≈ 5.2 s

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 µs)
Processor's clock rate	10^9 per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

- The OS reads and writes data in blocks
  - Block sizes may be 8, 16, 32 or 64KB
  - Reading 1 byte data takes same time as the entire block
  - Buffer: part of main memory where a block is stored after reading or while writing

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 µs)
Processor's clock rate	10^9 per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

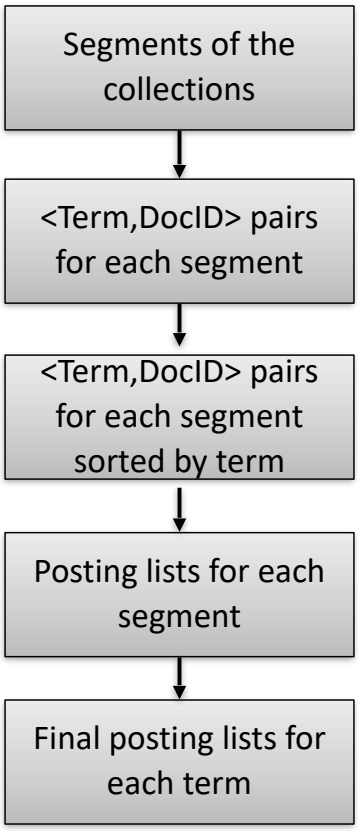
- Simultaneous reading and processing
  - Disk → main memory data transfer is done by system bus
  - Processor is free, can work while reading data
  - The system can read data and decompress at the same time

Hardware parameters

Parameter	HDD	SSD
Average seek time	5×10^{-3} s (5 ms)	0.1 ms
Average time to read a byte from disk	2×10^{-8} s (50MB/s)	5×10^{-9} s (200 MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 µs)	
Processor's clock rate	10^9 per second	
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)	

- The parameters change a lot for SSDs
  - Seek: about 50 times faster than HDD
  - Read: about 4 times faster than HDD

Blocked sort based indexing



Step 1: partition the collection into segments

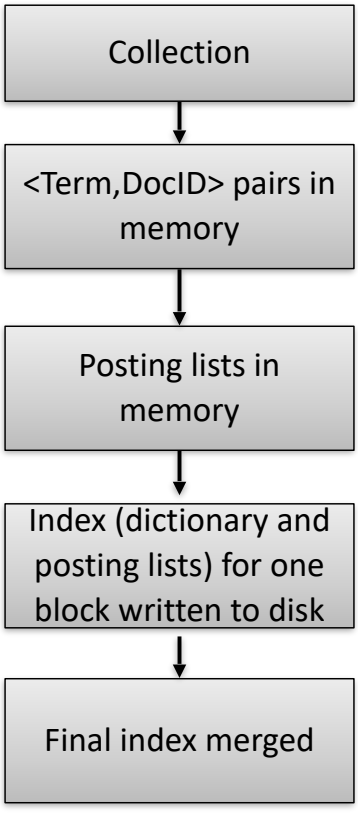
Step 2: make a pass over all documents, write out <term, docId> pairs

Step 3: separately in each segment, sort the <term,docId> pairs by term. May require an external memory sort.

Step 4: separately in each segment, create the posting lists for each term

Step 5: for each term, merge the posting lists from all segments and create one single posting list

Single-pass in memory indexing



Step 1: Keep parsing documents, convert to <term,docId> pairs

Step 2: invert <term,docId> pairs in memory to make posting lists; immediately available for searching as well

Step 3: When memory is exhausted, write out the whole index to disk after sorting the posting lists and dictionary

Step 4: merge the index files for different blocks

Dynamic indexing

- For fast access, posting lists are written in contiguous blocks
- Changes to existing index requires a lot of moving of the data
- Approach: create an auxiliary index for incremental changes, keep growing that index
  - Searches are performed in both old and auxiliary indices, results are merged
  - When auxiliary index grows significantly large, merge it with the original one (costly operation, but not done often)
- Ease of merging
  - If each posting list can be one file → Good
  - Bad idea! The OS cannot handle too many files (too many terms) well
  - Tradeoff between #of files and #of posting lists per file: keep some posting lists in one file, but not all in one

Security

Search in enterprise data

- Not all users may be allowed to view all documents
- Approach 1: use access control lists
  - For each user / group of users, have a posting list of documents the user / group can access
  - Intersect that list with the search result list
  - Problem: difficult to maintain when access changes
  - Access control lists may be very large
- Approach 2: check at query time
  - From the retrieved results, filter out those which the user is not allowed to access
  - May slow down retrieval

References

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/

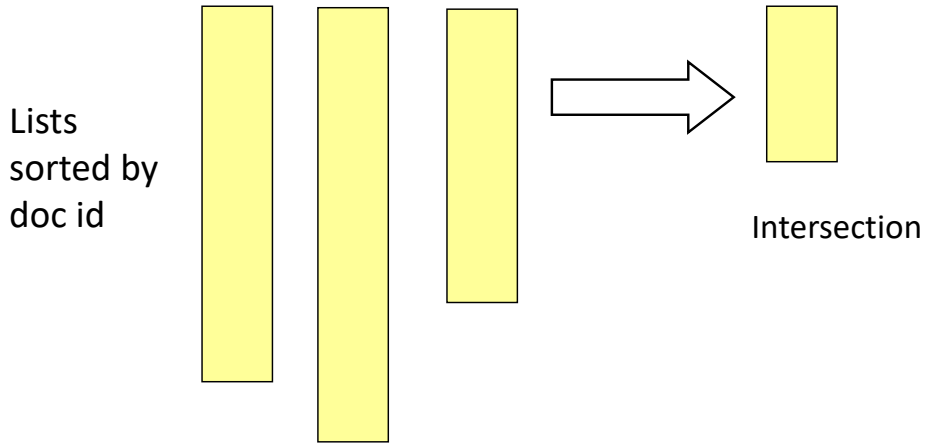
Index Structures and Query Processing

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Merge union or intersection

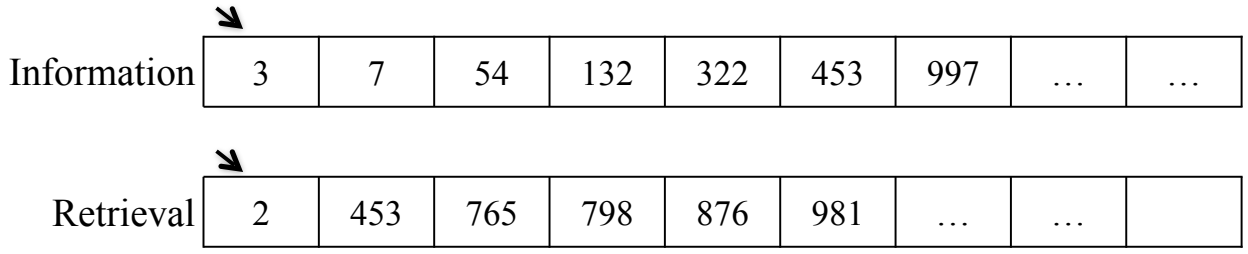
Have to scan the lists fully: $O(n)$, that can be bad!

The lists can be VERY large.



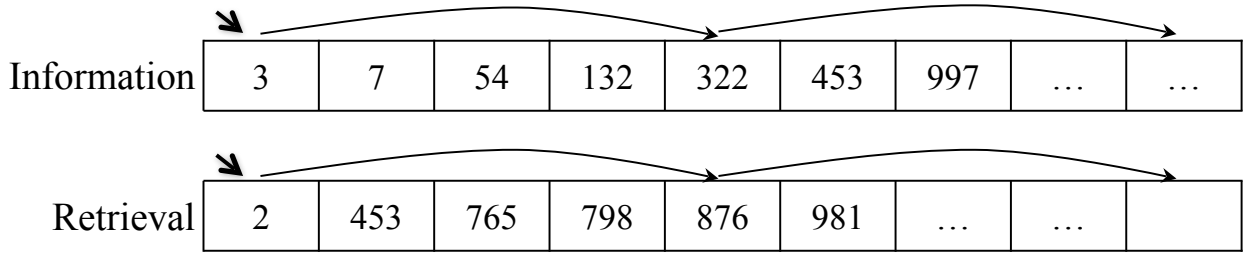
- Most real life queries are AND queries
  - User wants all the terms to be present

Skip lists: intersection



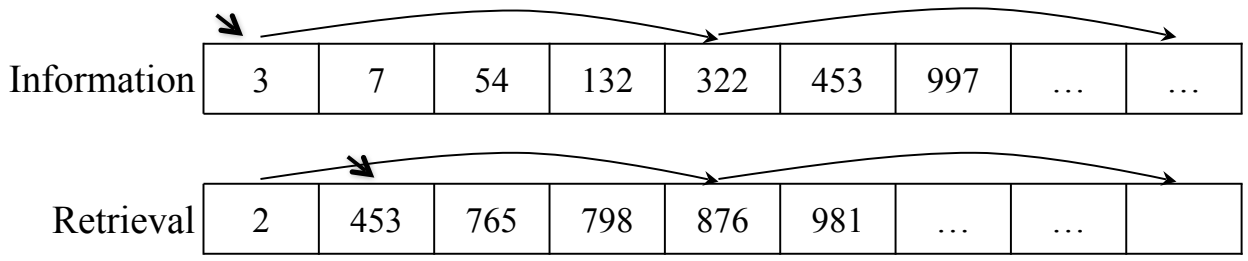
- Linear intersection
  - Start from the beginning
  - Determine the smaller id, move on that list
  - If ids on both lists match, add to result list
  - May have to advance pointer in one list alone and keep comparing
  - Could we skip them?

Skip lists: intersection



- Skip lists
  - Start from the beginning, determine the smaller id (2 < 3)
  - Is the id following the skip pointer on list 2 also smaller than 3?
  - No, so move to 453 on list 2

Skip lists: intersection



- Skip lists
  - Start from the beginning, determine the smaller id (2 < 3)
  - Is the id following the skip pointer on list 2 also smaller than 3?
  - No, so move to 453 on list 2
  - Now 3 < 453, the next id 7 also may be < 453, may be even the next
  - Check: is the next id following the skip pointer (322) < 453? Yes!
  - Skip to 322, skipping some part of the list
  - Continue ☺

Skip lists: discussion

- Built in indexing time
- Where to place the skip pointers?
  - More skip pointers: more comparison overhead
  - Less skip pointers: less skips
  - Empirical tradeoff: for a list of size n, keep $\sqrt{n}$ evenly spaced skip pointers
- Maintaining
  - Building at indexing time is easy
  - Maintaining is difficult if the index is updated frequently
  - In particularly, need to be careful with deletion

Phrase queries

- The simple term → doc ids posting list cannot answer phrase queries such as: “indian statistical institute”
- Bi-word index
  - Keep an index with all *pairs of consecutive word*s as keys

indian statistical	3	7	54	132	322	453	997	…	…
statistical institute	2	453	765	798	876	981	…	…	

- Does not exactly correspond to all documents corresponding to the query phrase, there would be some false positives [why?]
- But works fairly well in practice

Positional index

- Together with the term → document ids posting list, store the positions where the term occurs in each document
- Each entry in the posting list:
  `Doc_ID :: Score :: Positions`
- Intersection as usual on posting lists
- In addition to matching docIds, also match the positions

diwali:	d_3::0.3::<1>		
indian:	d_2::0.2::<4,8>	d_3::0.4::<7>	d_7::0.9::<1>
institute:	d_1::0.1::<2,9>	d_2::0.8::<10>	
population:	d_7::0.5::<2>		
autumn:	d_4::0.4::<3>		
statistical:	d_1::0.3::<1>	d_2::0.6::<9>	

Parametric and zone indexes

- Thus far, a document has been a sequence of terms
- In fact documents have multiple parts, some with special semantics:
  - Author
  - Title
  - Date of publication
  - Language
  - Format
  - etc.
- These constitute the *metadata* about a document

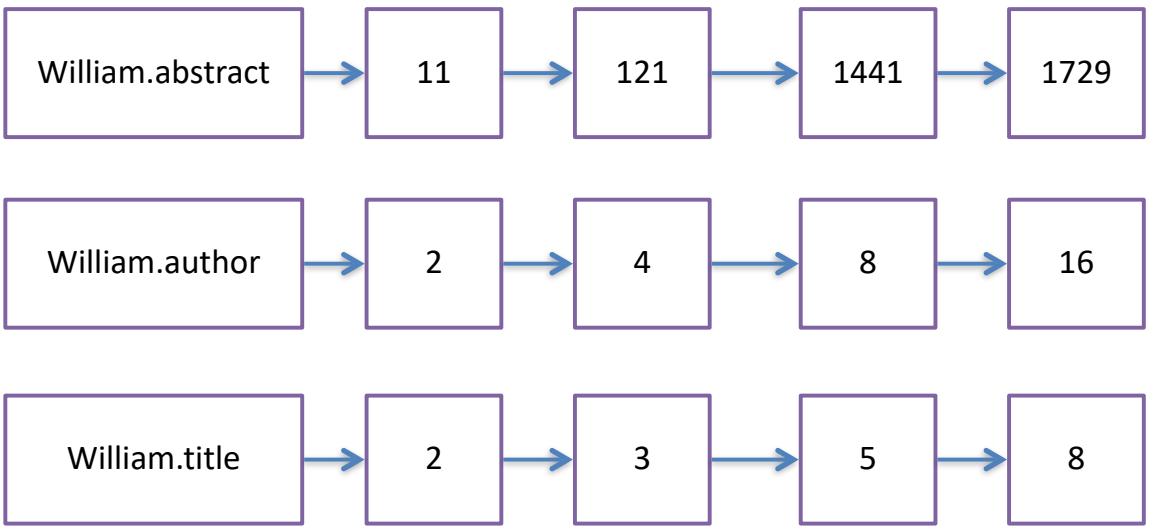
Fields

- We sometimes wish to search by these metadata
  - E.g., find docs authored by William Shakespeare in the year 1601, containing *alas poor Yorick*
- Year = 1601 is an example of a field
- Also, author last name = shakespeare, etc.
- Field or parametric index: postings for each field value
  - Sometimes build range trees (e.g., for dates)
- Field query typically treated as conjunction
  - (doc *must* be authored by shakespeare)

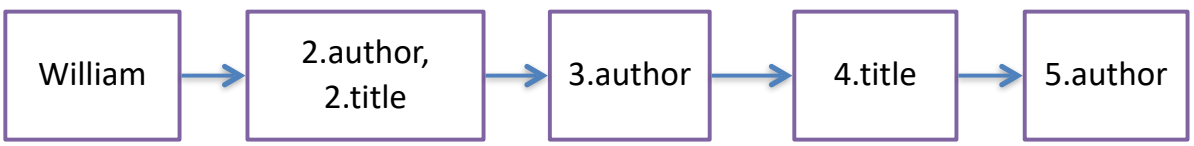
Zone

- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
  - Title
  - Abstract
  - References …
- Build inverted indexes on zones as well to permit querying
- E.g., "find docs with *merchant* in the title zone and matching the query *gentle rain*"

Example: Zone Index



Zone index and posting lists



Basics of Ranking

- Single term query
  - The score (may be TF-iDF) for the document corresponding to the term = the score for the document corresponding to the query
- Queries with more than one term: how do we combine scores across multiple posting lists?

Indian	D1 :: 0.5	D2 :: 0.2	D3: 0.7	D4 :: 0.2	D5 :: 0.1	D6 :: 0.2
Statistical	D2 :: 0.7	D5 :: 0.3				
Institute	D2 :: 0.2	D4 :: 0.3	D5 :: 0.8	D6 :: 0.2		

- Simple assumption: sum the scores (analogous to dot product)
- Usual merge union / intersection, add the scores along the way

Query processing

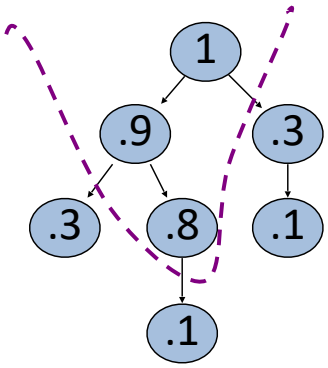
- How to compute cosine similarity and find top-k?
- Similar to merge union or intersection using inverted index
  - The aggregation function changes to compute cosine
  - We are interested only in ranking, no need of normalizing with query length
  - If the query words are unweighted, this becomes very similar to merge union
  - Exercise: work out the details
- Need to find top-k results after computing the union list with (cosine) scores

Partial sort using heap for selecting top k

Usual sort takes $O(n \log n)$ operations (may be n = 1 million)

Partial sort

- Binary tree in which each node's value > the values of children
- Takes $O(n)$ operations to construct, then read each of top k in $O(\log n)$ steps.
- For n =1 million, k = 100, this is about 10% of the cost of sorting



Length normalization and query – document similarity

Term – document scores

TF.iDF or similar scoring. May already apply some normalization to eliminate bias on long documents

Document length normalization

Ranking by cosine similarity is equivalent to further normalizing the term – document scores by document length. Query length normalization is redundant.

Similarity measure and aggregation

- Cosine similarity
- Sum the scores with union
- Sum the scores with intersection
- Other possible approaches

Top-k algorithms

- If there are millions of documents in the lists
  - Can the ranking be done without accessing the lists fully?
- Exact top-k algorithms (used more in databases)
  - Family of threshold algorithms (Ronald Fagin et al)
  - Threshold algorithm (TA)
  - No random access algorithm (NRA) [we will discuss, as an example]
  - Combined algorithm (CA)
  - Other follow up works

NRA (No Random Access) Algorithm

Fagin's NRA Algorithm:

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

read one doc from every list

NRA (No Random Access) Algorithm

Fagin's NRA Algorithm: round 1

0.6 + 0.6 + 0.9 = 2.1

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

current score

lists sorted by score

Candidates

best-score

Candidates
doc 83 [0.9, **2.1**]
doc 17 [0.6, **2.1**]
doc 25 [0.6, **2.1**]

min top-2 score: 0.6

maximum score for unseen docs: **2.1**

min-top-2 < best-score of candidates

read one doc from every list

NRA (No Random Access) Algorithm

Fagin's NRA Algorithm: round 2

0.5 + 0.6 + 0.7 = 1.8

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

Candidates

Candidates
doc 17 [1.3, **1.8**]
doc 83 [0.9, **2.0**]
doc 25 [0.6, **1.9**]
doc 38 [0.6, **1.8**]
doc 78 [0.5, **1.8**]

min top-2 score: 0.9

maximum score for unseen docs: **1.8**

min-top-2 < best-score of candidates

read one doc from every list

NRA (No Random Access) Algorithm

0.4 + 0.6 + 0.3 = 1.3

Fagin's NRA Algorithm: round 3

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

read one doc from every list

Candidates

doc 83 [1.3, **1.9**]
doc 17 [1.3, **1.7**]
doc 25 [0.6, **1.5**]
doc 78 [0.5, **1.4**]

min top-2 score: 1.3

maximum score for unseen docs: 1.3

min-top-2 < best-score of candidates

no more new docs can get into top-2

but, extra candidates left in queue

NRA (No Random Access) Algorithm

0.3 + 0.6 + 0.2 = 1.1

Fagin's NRA Algorithm: round 4

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

read one doc from every list

Candidates

doc 17 **1.6**
doc 83 [1.3, **1.9**]
doc 25 [0.6, **1.4**]

min top-2 score: 1.3

maximum score for unseen docs: 1.1

min-top-2 < best-score of candidates

no more new docs can get into top-2

but, extra candidates left in queue

NRA (No Random Access) Algorithm

0.2 + 0.5 + 0.1 = 0.8

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

read one doc from every list

Fagin's NRA Algorithm: round 5

Candidates

doc 83 **1.8**
doc 17 **1.6**

min top-2 score: **1.6**

maximum score for unseen docs: 0.8

no extra candidate in queue

Done!

More approaches:

- Periodically also perform random accesses on documents to reduce uncertainty (CA)
- Sophisticated scheduling on lists
- Crude approximation: NRA may take a lot of time to stop. Just stop after a while with approximate top-k – who cares if the results are perfect according to the scores?

Inexact top-k retrieval

- Does the exact top-*k* matter?
  - How much are we sure that the 101st ranked document is less important than the 100th ranked?
  - All the scores are simplified models for what information may be associated with the documents
- Suffices to retrieve *k* documents with
  - Many of them from the exact top-*k*
  - The others having score close to the top-*k*

Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest score in *t*'s posting list
  - Ideally $k < r << n$ (n = size of the posting list)
  - Champion list for *t* (or fancy list or top docs for *t*)
- Note: *r* has to be chosen at index build time
  - Thus, it's possible that $r < k$
- At query time, only compute scores for docs in the champion list of some query term
  - Pick the *k* top-scoring docs from amongst these

Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - (Pagerank)

Modeling authority

- Assign a *query-independent* quality score in [0,1] to each document d
  - Denote this by $g(d)$
- Consider a simple total score combining cosine relevance and authority
- combined-score(q,d) = $g(d)$ + cosine(q,d)
  - Can use some other linear combination
  - Indeed, any function of the two "signals" of user happiness – more later
- Now we seek the top k docs by the combined score

Top-k by combined score – fast methods

- First idea: Order all postings by $g(d)$
- Key advantage: this is a common ordering for all posting lists
- Thus, can concurrently traverse query terms' postings for
  - Postings intersection
  - Cosine score computation
- Under $g(d)$-ordering, top-scoring docs likely to appear early in postings traversal
- In time-bound applications (have to return whatever search results we can in 50 ms), this allows us to stop postings traversal early
  - Short of computing scores for all docs in postings

Champion lists in $g(d)$-ordering

- Can combine champion lists with $g(d)$-ordering
- Maintain for each term a champion list of the r docs with highest $g(d)$ + tf-idf(t, d)
- Seek top-k results from only the docs in these champion lists

High and low lists

- For each term, we maintain two postings lists called *high* and *low*
  - Think of *high* as the champion list
- When traversing postings on a query, only traverse *high* lists first
  - If we get more than k docs, select the top k and stop
  - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality $g(d)$ scores
- A means for segmenting index into two tiers

Impact-ordered postings

- We only want to compute scores for docs d for which $wf_{t,d}$, for query term t, is high enough
- Sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick top k?
  - Two ideas follow

1. Early termination

- When traversing t's postings, stop early after either
  - a fixed number of r docs
  - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
  - One from the postings of each query term
- Compute only the scores for docs in this union

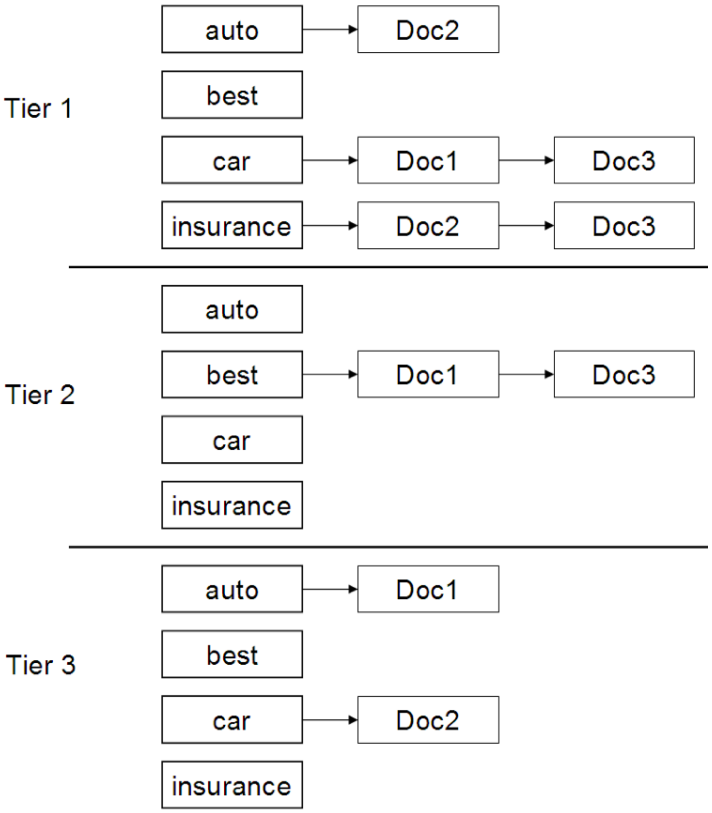
2. iDF-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
  - High idf terms likely to contribute most to score
- As we update score contribution from each query term
  - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

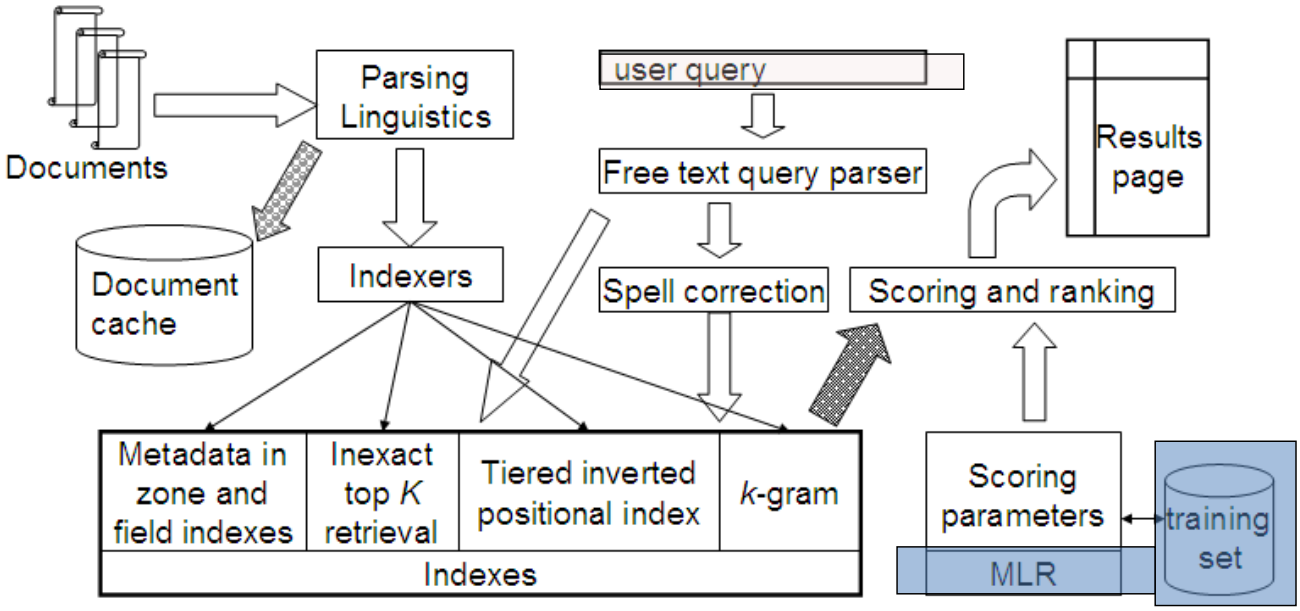
Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important
- Can be done by $g(d)$ or another measure
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
  - If so drop to lower tiers

Example tiered index



Putting it all together



Sources and Acknowledgements

- IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/
- Acknowledgment: Some slides are adapted from the slides by Prof. Nayak and Prof. Raghavan for their course in Stanford University

Index Compression

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Data compression

Represent the information using (hopefully) lesser number of bits than used by the original representation

- Lossy: may lose some "information"
  - Consequently, may not be able to get back the original
- Lossless: just a different representation, will be able to "decompress" and get back the original
  - We will consider only lossless compression now

Question:

Can a lossless compression algorithm "compress" any input data?

Index compression

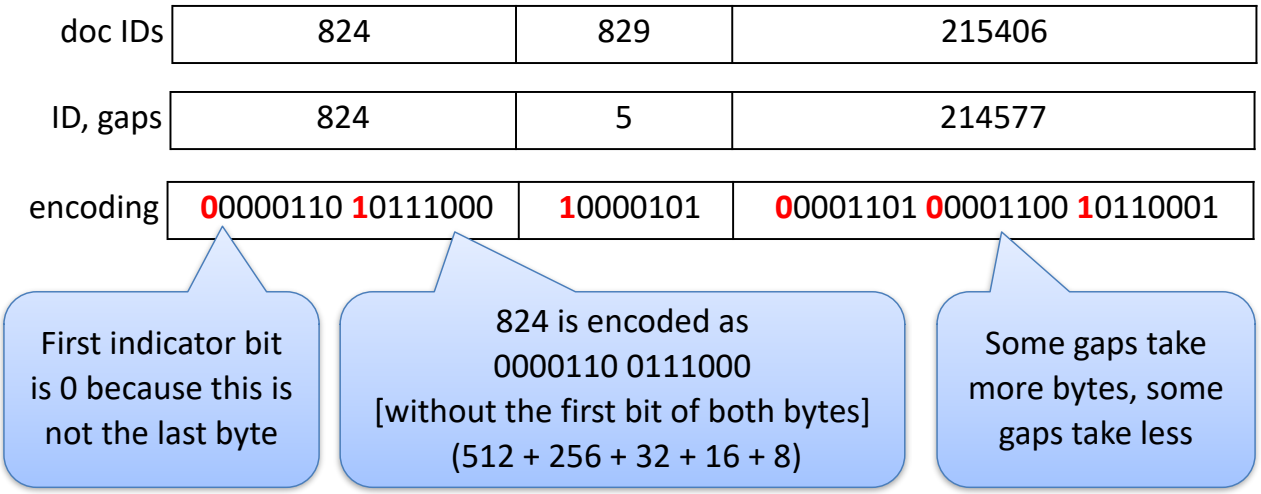
- Use less space to store
  - In disk or in memory
  - Saving is saving anyway
- Dictionary compression
  - May store the full or most of the dictionary in memory
- Posting list compression
  - Assumption: cannot store it in memory
  - Transfer data from disk faster
  - Typically can compress index by 4 times
- Need a fast decompression
  - Without compression: transfer 4*x* bytes from disk
  - With compression: transfer *x* bytes and decompress

The posting lists

zuckerberg	…	3429934	3478390	3689839	…
the	…	45	46	47	…
ashish	…	54892834	394099834	828493832	…

- For a "small" dataset, need 4 bytes (or even less) to store the doc IDs
- For a "very large" dataset, may need ~8 bytes to store the doc IDs
- Observation 1: the docIDs are large integers, but the gaps are not so large!
  - Idea: Store the first doc ID, and then store only the gaps (**Gap encoding**)
- Observation 2(a): for some posting lists, the gaps are very small
  - Very frequent terms
- Observation 2(b): For some the gaps can be large as well
  - Very infrequent terms
- Some posting lists may also have some large and some small gaps
  - Fixed size for gaps would not work!

Gap encoding: variable byte encoding



- Approach: store the first doc ID, then store the gaps
- Use 1 byte units
- First bit is the indicator, next 7 bits are payload (the number)
- First bit is 1 if this is the last byte, 0 otherwise

Elias γ encoding

- Unary code: $n \mapsto n$ 1s followed by a 0
  - For example $3 \mapsto 1110$
  - Some conventions use n 0s followed by a 1 (equivalent)
- γ encoding (due to Peter Elias)
  - Consider the binary representation of n
  - It starts with a 1; chop it off (we know it, so redundant)
  - Example: $9 \mapsto 1001 \mapsto 001$ (call this part offset)
  - Encode the length of the offset in unary
  - Example: length of offset of 9 is 3. Unary encoding: 1110
  - γ encoding of $9 \mapsto 1110001$

length of offset is 3 in unary

the number is leading 1 and the offset = 1001

Elias γ encoding

- Better than variable byte encoding because γ encoding is bit-level encoding (empirically 15% better in posting lists)
- Decompression is less efficient than variable byte encoding
- Observation: the unary encoding of length may use a lot of space
- Solution: encode length + 1 using γ encoding again (Elias δ encoding)
  - More effective for large numbers
  - Example:
    509 ↦ 111111110 11111101 (γ) ↦ 111000 11111101 (δ)
- Question: why length + 1?
- Omega encoding: recursively apply γ encoding on the length

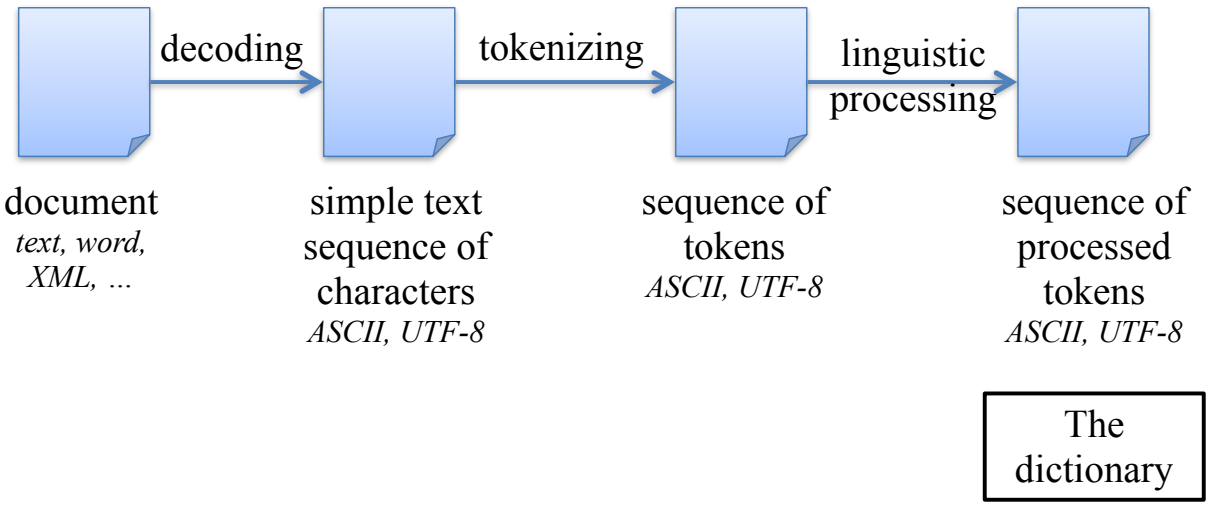
References

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/

Dictionaries and Tolerant Retrieval

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Pre-processing of a document



Tokenization

- Simplest: split sentences by whitespace
  - But, that's too naive, not good enough
- A punctuation may not mean end of sentence, or even a word
  - Acronyms: M.Sc., U.S.A., …
  - Dates: 08.02.2021, 08/02/21, …
  - Decimal numbers: interest rate 4.90%
  - URLs: https://www.isical.ac.in, email IDs, hashtags
  - Some phrases (multiple words) need to be treated as "one word"
  - New York, rock 'n' roll
- Some languages do not use spaces to separate words!
- Reasonable approach: tokenize text using *regular expressions*

Stop words

- Exclude the common words from the dictionary entirely
- Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques: the space for including stop words in a system is very small
  - Good query optimization techniques: mean you pay little at query time for including stop words.
  - We 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"

Normalization to terms

- We may need to "normalize" words in indexed text as well as query words into the same form
  - We want to match ***U.S.A.*** and ***USA***
- Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
  - deleting periods to form a term
    - ***U.S.A.*** → ***USA***
  - deleting hyphens to form a term
    - ***anti-discriminatory*** → ***antidiscriminatory***

Normalization: other languages

- Accents: e.g., French ***résumé*** vs. ***resume***
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent
- Most important criterion:
  - How are the users like to write their queries for these words?

- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - ***Tuebingen, Tübingen, Tubingen*** → ***Tubingen***

Normalization: other languages

- Normalization of different date formats
  - 02/08/2021, 08/02/2021, 08 Feb 2021, …
  - Note: more intelligence required here

- Tokenization and normalization may depend on the language and so is intertwined with language detection

- Normalization may depend on the language

Morgen will ich in MIT …

Is this MIT the institution, or the German word "mit" (means "with")

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - In search, often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

- CAT
  - Does it mean the animal cat?
  - Or the common admission test?

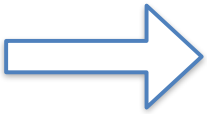
Lemmatization

- 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

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.



for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- *(m>1) EMENT* →
  - *replacement* → *replac*
  - *cement* → *cement*

Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
    - Single-pass, longest suffix removal (about 250 rules)
  - Paice/Husk stemmer
  - Snowball

- Full morphological analysis (lemmatization)
  - At most modest benefits for retrieval

Language-specificity

- The above methods embody transformations that are
  - Language-specific, and often
  - Application-specific
- These are “plug-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Does stemming help?

- Search
  - English: very mixed results. Helps recall for some queries but harms precision on others
    - E.g., operative (dentistry) ⇒ oper
  - Definitely useful for Spanish, German, Finnish, …
    - 30% performance gains for Finnish!
- NLP Tasks
  - Generic "classification tasks" aiming to predict the target by identifying "topic": yes
  - Machine translation: no!
  - Sentiment analysis: no!

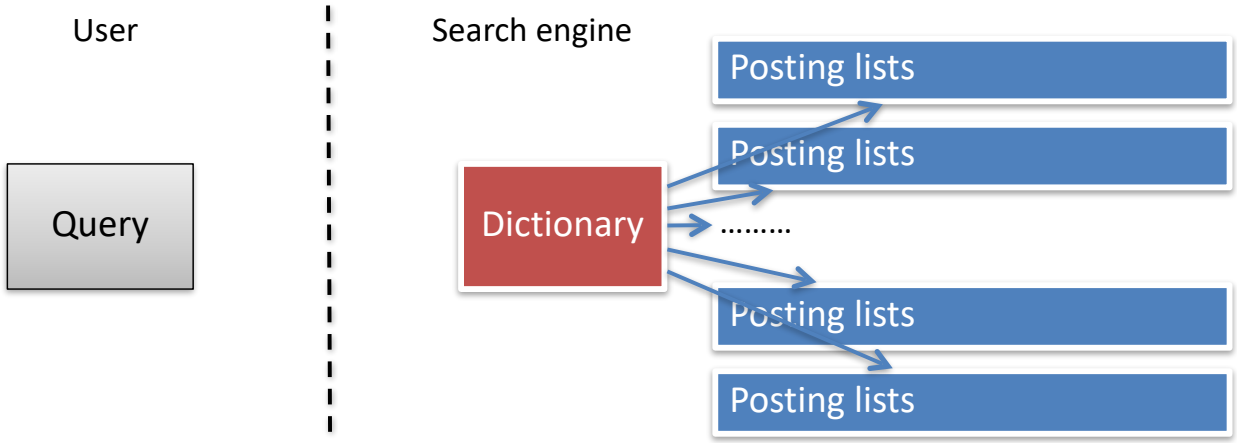
Stemming for sentiment analysis

Porter stemmer (suffix stripping)

Positive sentiment	Negative sentiment	Stem
defense	defensive	defens
affection	affect	affect
objective	objection	object
tolerant	tolerable	toler
extravagance	extravagant	extravag

- Different forms of the same stem may carry different sentiments
- WordNet stemmer does not have this problem, but still it removes comparative morphology, e.g. **happiest, happy** → **happy**

The dictionary



- User sends the query
- The engine
  - Determine the query terms
  - Determine if each query term is present in the dictionary
  - Dictionary: Lookup
  - Search trees or hashing

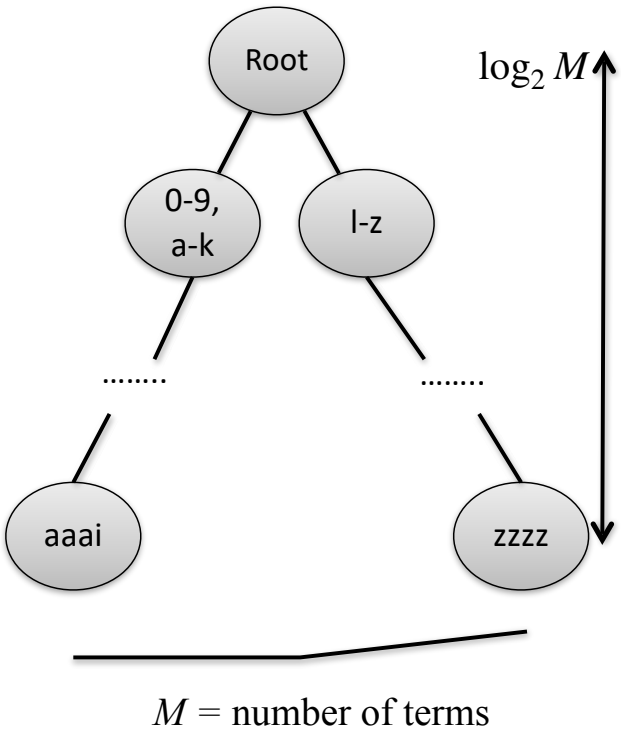
Binary search trees

Binary search tree

- Each node has two children
- O(log M) comparisons if the tree is balanced

Problem

- Balancing the tree when terms are inserted and deleted



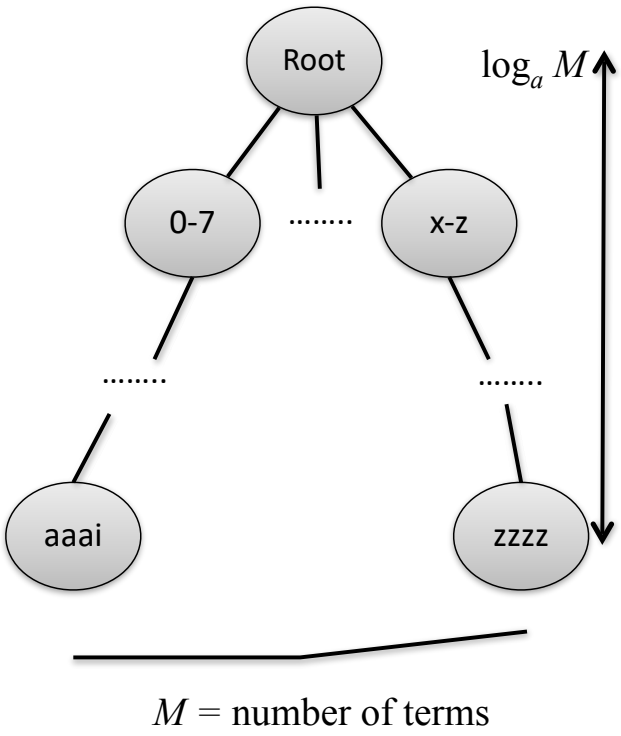
B-tree

B-tree

- Number of children for each node is between *a* and *b* for some predetermined *a* and *b*
- O($\log_a M$) comparisons
- Very few rebalancing required

B+ tree

- Similar to B-tree
- All data (pointers to posting lists) are in leaf nodes
- Linear scan of data easier



WILDCARD QUERIES

Wildcard queries

- Wildcard: is a character that may be substituted for any of a defined subset of all possible characters
- Wildcard queries: queries with wildcards
  - sydney/sidney: `s*dney`
  - debapriyo/debopriya/debopriyo: deb`*priy*`
  - judicial/judiciary: `judicia*`
- Trailing wildcard queries
  - Simplest: search trees work well
  - Determine the node(s) which correspond to the range of terms specified by the query
  - Retrieve the posting lists for the set *W* of all the terms in the entire sub-tree under those nodes

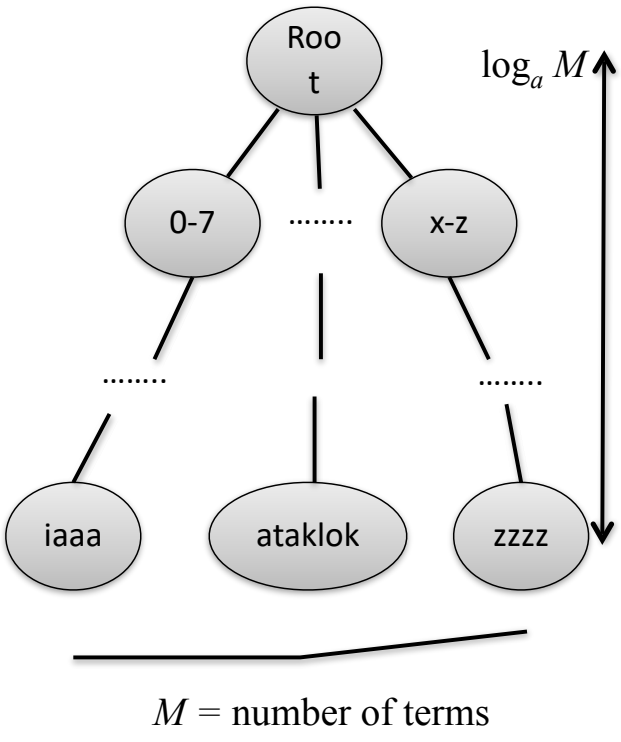
Trailing wildcard query

Queries with a single *

- Leading wildcard queries: `*ata`
  - Matching kolkata, bata, …
- Use a reverse B-tree
  - B-tree obtained by considering terms backwards
  - Consider the leading wildcard query backwards, it becomes a trailing wildcard query
  - Lookup as before for a trailing wildcard query on a normal B-tree

Queries with a single *

- Queries of the form: `s*dney`
  - Matching sydney, sidney, …
- Use a B-tree and a reverse B-tree
  - Use the B-tree to get the set *W* of all terms matching s*
  - Use the reverse B-tree to get the set *R* of all terms matching *dney
  - Intersect *W* and *R*



General wildcard queries

The permuterm index

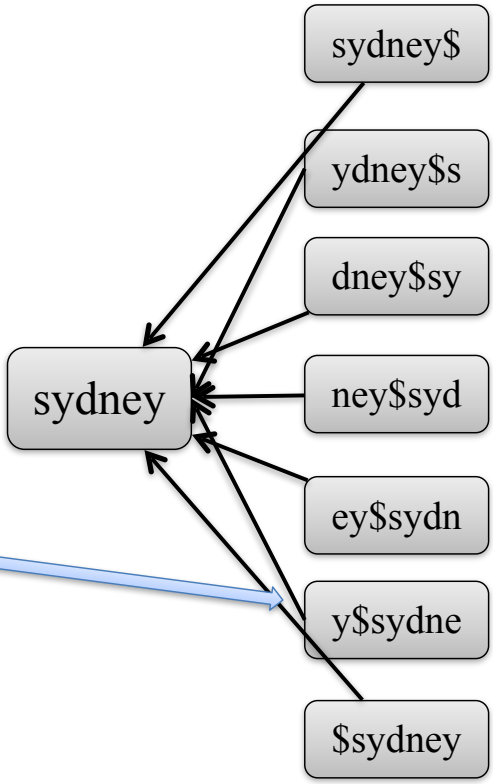
- Special character $ as end of term
- The term sydney → sydney$
- Enumerate all rotations of sydney$ and have all of them in the B-tree, finally pointing to sydney

Wildcard queries

- A single *: sy*ey
  - Query ey$sy* to the B-tree
  - One rotation of sydney$ will be a match
- General: s*dn*y
  - Query y$s* to the B-tree
  - Works equivalent to s*y, not all of the matches would have "dn" in the middle
  - Filter the others by exhaustive checks

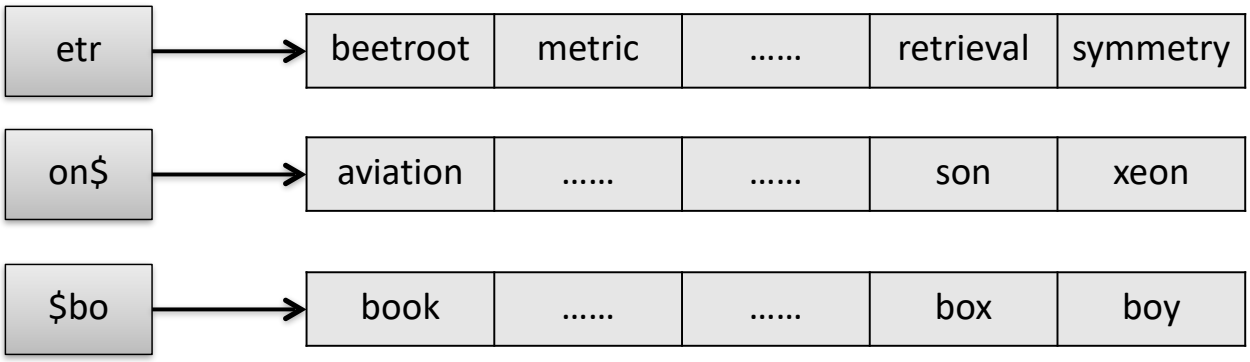
Problems

- Blows up the dictionary
- Empirically seen to be 10 times for English



k-gram index for wildcard queries

- *k*-gram: sequence of *k* characters
- *k*-gram index: <*k*-gram> → words in which the *k*-gram appears, sorted lexicographically
  - Consider all words with the beginning and ending marker $



k is predetermined

Wildcard queries with *k*-gram index

- User query: `re*ve`
  - Send the Boolean query $re AND ve$ to the *k*-gram index
  - Will return terms such as revive, remove, …
  - Proceed with those terms and retrieve from inverted index
- User query: `red*`
  - Send the query $re AND red to the 3-gram index
  - Returns all results starting with "re" and containing "red"
  - Post-processing to keep only the ones matching red*
- Exercise: more general wildcard query `s*dn*y`
  - Can we do this using the *k*-gram index (assume 3-gram)?

Discussion on wildcard queries

- Semantics
  - What does `re*d AND fe*ri` mean?
  - (Any term matching re*d) AND (Any term matching fe*ri)
  - Once the terms are identified, the operation on posting lists
  - (… Union …) Intersection (… Union …)
  - Expensive operations, particularly if there are many matching terms
- Expensive even without Boolean combinations
- Hidden functionality in search engines
  - Otherwise users would "play around" even when not necessary
  - For example, a query "s*" produces huge number of terms for which the union of posting lists need to be computed

Why search trees are better than hashing?

- Possible hash collision
- Prefix queries cannot be performed
  - red and re may be hashed to entirely different range of values
- Almost similar terms do not hash to almost similar integers
- A hash function designed now may not be suitable if the data grows to a much larger size

Did you mean?

SPELLING CORRECTIONS

Misspelled queries

- People type a lot of misspelled queries
  - britian spears, britney's spears, brandy spears, prittany spears → britney spears
- What to do?
  1. Among the possible corrections, choose the "nearest" one
  2. Among the possible "near" corrections, choose the most frequent one (probability of that being the user's intention is the highest)
  3. Context sensitive correction
  4. The query may not be actually incorrect. Retrieve results for the original as well as possible correction of the query
     - debapriyo majumder → returns results for debapriyo majumdar and majumder both
- Approaches for spelling correction
  - Edit distance
  - *k*-gram overlap

Edit distance

- Edit distance E(A,B) = minimum number of operations required to obtain B from A
  - Operations allowed: insertion, deletion, substitution
- Example: E(food, money) = 4
  - food → mood → mond → moned → money
- Computing edit distance in O(|A| . |B|) time
- Spelling correction
  - Given a (possibly misspelled) query term, need to find other terms (in the dictionary) with very small edit distance
  - Precomputing edit distance for all pairs of terms → absurd
  - Use several heuristics to limit possible pairs
    - Only consider pairs of terms starting with same letter

Computing edit distance

Observation:

- E(food, money) = 4
  - One sequence: food → mood → mond → moned → money
- E(food, moned) = 3
- Why?
  - If E(food, moned) < 3, then E(food, money) < 4

Prefix property: If we remove the last step of an optimal edit sequence then the remaining steps represent an optimal edit sequence for the remaining substrings

Computing edit distance

- Fix the strings A and B. Let |A| = m, |B| = n.
- Define: E(i, j) = E(A[1, … , i], B[1, … , j])
  - That is, edit distance between the length i prefix of A and length j prefix of B
- Note: E(m, n) = E(A,B)
- Recursive formulation

  (a) E(i, 0) = i

  (b) E(0, j) = j
- The last step: 4 possibilities
  - Insertion: E(i, j) = E(i, j – 1) + 1
  - Deletion: E(i, j) = E(i – 1, j) + 1
  - Substitution: E(i, j) = E(i – 1, j – 1) + 1
  - No action: E(i, j) = E(i – 1, j – 1)

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0						
1	M					
2	O					
3	N					
4	E					
5	Y					

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1				
2	O	2				
3	N	3				
4	E	4				
5	Y	5				

Backtrace:

Compute E(*i*, *j*) and also keep track of where E(*i*, *j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$
$$E(0,j) = j$$
$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable
P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1			
2	O	2				
3	N	3				
4	E	4				
5	Y	5				

Backtrace:
Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$
$$E(0,j) = j$$
$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable
P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1			
2	O	2	2			
3	N	3				
4	E	4				
5	Y	5				

Backtrace:
Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$
$$E(0,j) = j$$
$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable
P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2			
3	N	3	3			
4	E	4	4			
5	Y	5	5			

Backtrace:
Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$
$$E(0,j) = j$$
$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable
P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1		
3	N	3	3			
4	E	4	4			
5	Y	5	5			

Backtrace:
Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2		
4	E	4	4	3		
5	Y	5	5	4		

Backtrace:

Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$

$$E(0,j) = j$$

$$E(i,j) = \min \begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable

P = 1 if A[i] ≠ B[j], 0 otherwise

		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2	2	3
4	E	4	4	3	3	3
5	Y	5	5	4	4	4

Backtrace:

Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Computing edit distance: dynamic programming

The recursion

$$E(i,0) = i$$
$$E(0,j) = j$$
$$E(i,j) = \min\begin{cases} E(i,j-1)+1 \\ E(i-1,j)+1 \\ E(i-1,j-1)+|P| \end{cases}$$

P is an indicator variable
P = 1 if A[i] ≠ B[j], 0 otherwise

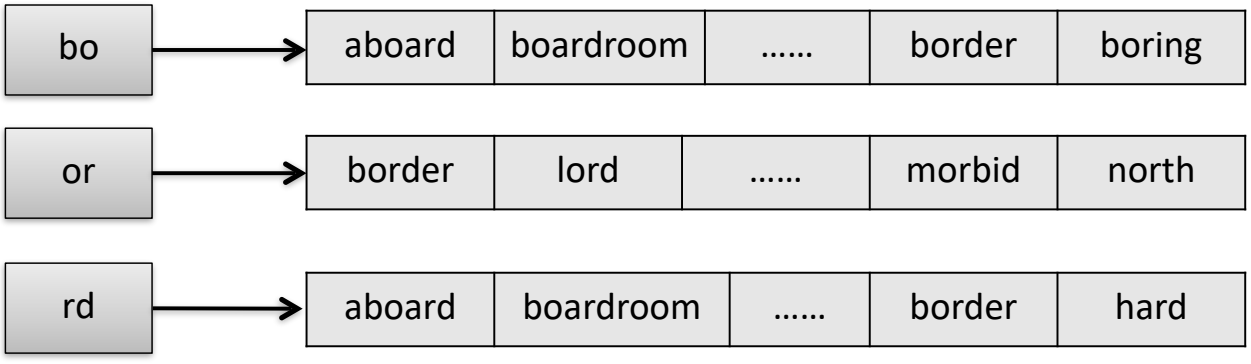
		0	1	2	3	4
			F	O	O	D
0		0	1	2	3	4
1	M	1	1	2	3	4
2	O	2	2	1	2	3
3	N	3	3	2	2	3
4	E	4	4	3	3	3
5	Y	5	5	4	4	4

Backtrace to find an optimal edit path

Backtrace:
Compute E(*i, j*) and also keep track of where E(*i, j*) came from

Spelling correction using *k*-gram index

- The *k*-grams are *small* portions of words
- Misspelled word would still have some *k*-grams intact
- Misspelled query: `bord`



- Intersect the list of words for *k*-grams
- Problem: long words which contain the *k*-grams but are not good corrections

Phonetic correction

- Some users misspell because they don't know the spelling
- Types as it "sounds"
- Approach for correction: use a phonetic hash function
  - Hash similarly sounding terms to the same hash value
- Soundex algorithm
  - Several variants

Soundex algorithm

1. Retain the first letter of the term
2. Change all
   A, E, I, O, U, H, W, Y → 0
   B, F, P, V → 1.
   C, G, J, K, Q, S, X, Z → 2.
   D,T to 3.
   L to 4.
   M, N to 5.
   R to 6.
3. Repeat: remove one of each pair of same digit
4. Remove all 0s. Pad the result with trailing 0s. Return the first 4 positions: one letter, 3 digits

Example: Hermann → H065055 → H06505 → H655

References and acknowledgements

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/
- The part on edit distance: lectures notes by John Reif, Duke University:https://www.cs.duke.edu/courses/fall08/cps230/Lectures/L-04.pdf

PageRank

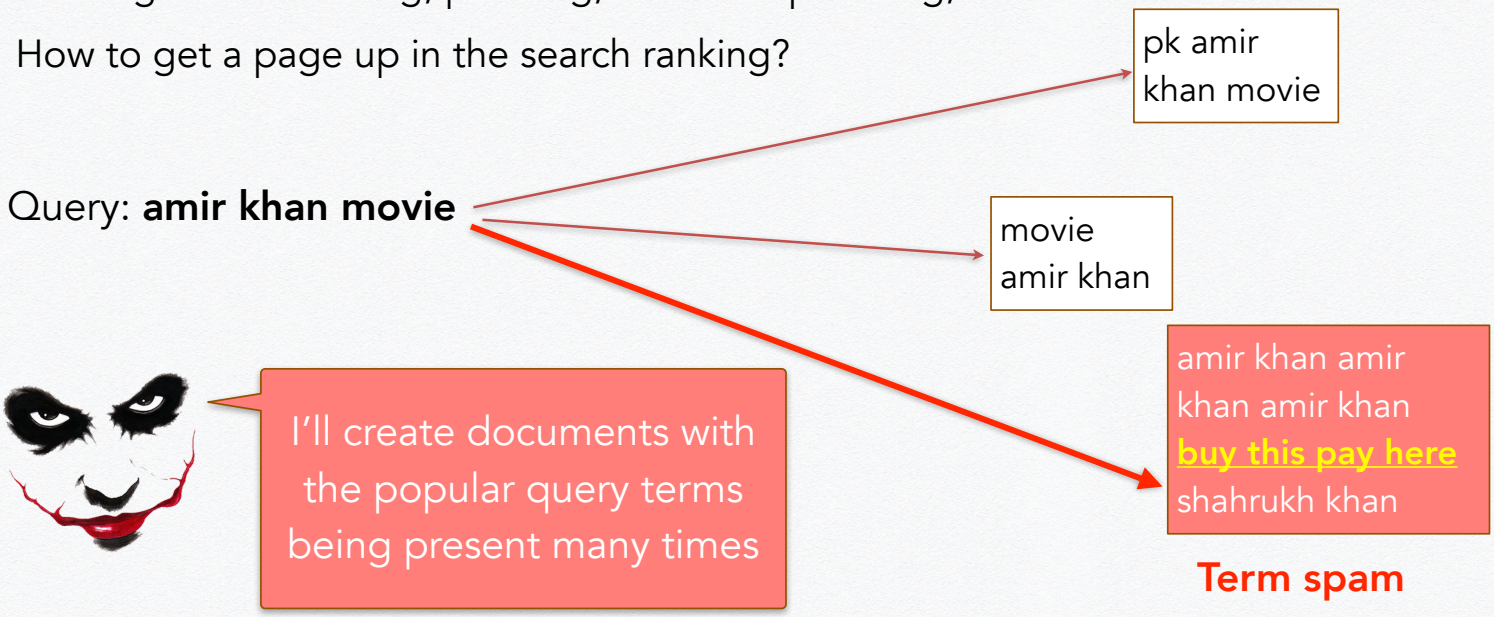
Debapriyo Majumdar
Indian Statistical Institute
debapriyo@isical.ac.in

Early search engines

- Main approach: matching terms (loosely speaking, words)
  - If query terms are present in a document, then the document is possibly relevant
  - Tf.IDF (and many other variants): assign a score for every term in a document
    - Intuition 1: more times a term is present in a document, the more important it is
      - Term frequency (TF)
    - Intuition 2: If a term is present in many documents, it is not *special* in any of the documents
      - (Inverse) document frequency (IDF)
  - Rank documents based on these scores
    - Higher Tf.IDF score $\Longrightarrow$ document showed up higher in search engine ranking

The spammer wants to exploit the ranking algorithm

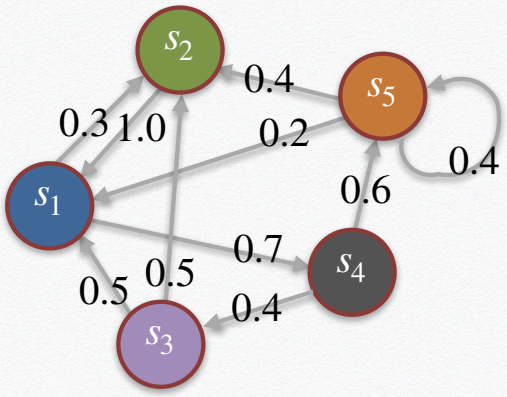
- Spammers' goal: get their pages to show up in the search results to receive clicks
  - End goal: advertising, phishing, malware spreading, …
- How to get a page up in the search ranking?



PageRank

- Assumption (a reasonable one)
  - Users of the web are largely reasonable people
  - They put (more) links to useful pages
- PageRank
  - Named after Larry Page (co-founder of Google Inc.)
  - Patented by Stanford University, later bought by Google
- Approach
  - Importance (PageRank) of a webpage is influenced by the number and quality of links into the page
  - Search results ranked by term matching as well as PageRank
  - Intuition – Random web surfer model: a random surfer follows links and surfs the web. More likely to end up at more important pages
- Advantage: term spam cannot ensure in-links into those pages
- Many variations of PageRank

$n = 5$ in this example



	1.0	0.5		0.2
0.3		0.5		0.4
			0.4	
0.7				
			0.6	0.4

Markov Chain

- A **discrete-time stochastic** (random) process
- Set $\mathcal{S} = \{s_1, \ldots, s_N\}$ of n states
- In any one of the states at any given time step t
- A probability distribution $p : S \times S \rightarrow [0,1]$ determines the probabilities of going to a state at the next time step $t + 1$
- Can define a transition matrix $M = (p_{ij})_{1 \le i,j \le n}$ as $p_{ij} := p(s_i \mid s_j) = p[S_{t+1} = s_i \mid S_t = s_j]$
  - If at state s_j now, the probability of going to state s_i in the next step is p_{ij}
- Markov property: $\sum_{i=1}^{n} p_{ij} = 1$, for all $j = 1, \ldots, n$.

Markov Chain and Stochastic Matrix

- A matrix $M = (p_{ij})_{1 \le i,j \le n}$ with the following properties

  All entries represent probabilities: $p_{ij} \in [0,1]$, and

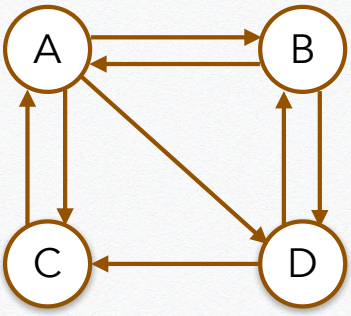
  Column sums are 1: $\sum_{i=1}^{n} p_{ij} = 1$, for all $j = 1, \ldots, n$.

  Is called a **left stochastic matrix**.
- Note: the formulation is also valid with a right stochastic matrix (transpose), where the row sums are 1.
- Property of a left stochastic matrix:
  - Largest eigenvalue is 1.
  - $Mv = v$ where v is the principal eigenvector.

	1.0	0.5		0.2
0.3		0.5		0.4
			0.4	
0.7				
			0.6	0.4

The random surfer model

A tiny web



- Web graph, links are directed edges
  - Assume equal weights in this example
  - If a surfer starts at A, with probability 1/3 each, may go to B, C, or D
  - If a surfer starts at B, with probability 1/2 each may go to A or D
  - Can define a transition matrix
- Markov process:
  - Future state solely based on present
  - $M_{ij} = P[j \rightarrow i \text{ in the next step} \mid \text{presently in } j]$

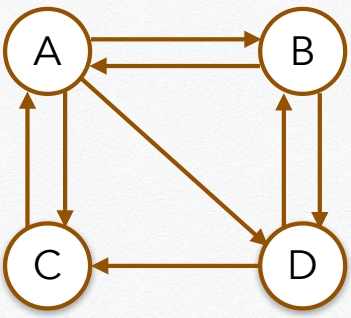
$M =$

A	B	C	D
0	1/2	1	0
1/3	0	0	1/2
1/3	0	0	1/2
1/3	1/2	0	0

Example courtesy: book by Leskovec, Rajaraman and Ullman

The random surfer model

A tiny web



- Random surfer: initially at any position, with equal probability $1/n$
- Distribution (column) vector $v = (1/n, \ldots, 1/n)$
- Probability distribution for her location after one step?
- Distribution vector: Mv
- Distribution vector after 2 steps: M^2v

Initially at A (1/4): A → A: not possible. Probability = 0
Initially at B (1/4): B → A (1/2). Overall probability = 1/8
Initially at C (1/4): C → A (1). Overall probability = 1/4
Initially at D (1/4): D →A (0). Overall probability = 0

$M =$

A	B	C	D
0	1/2	1	0
1/3	0	0	1/2
1/3	0	0	1/2
1/3	1/2	0	0

$v =$

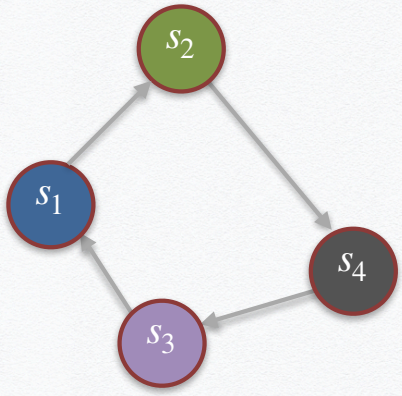
1/4
1/4
1/4
1/4

$Mv =$

0 + 1/8 + 1/4 + 0 = 9/24
1/12 + 0 + 0 + 1/8 = 5/24
1/12 + 0 + 0 + 1/8 = 5/24
1/12 + 1/8 + 0 + 0 = 5/24

Ergodic Markov Chain

- **Ergodic** Markov chain: if there exists a positive integer t_0 such that for all pairs of states s_i, s_j in the Markov chain, if it is started at time 0 in state s_i then for all $t > t_0$, the probability of being in state s_j at time t is greater than 0.
  - In other words, eventually, the probability of being at any state at any point of time is positive.
- Necessary conditions for ergodicity
  - (a) **Irreducibility**: there is a sequence of non-zero probabilities from any state to another.
  - (b) **Aperiodicity**: states are not partitioned into sets such that all transitions happen cyclically from one to another (not periodic).



Example of a *periodic* Markov chain

Perron — Frobenius theorem (1912)

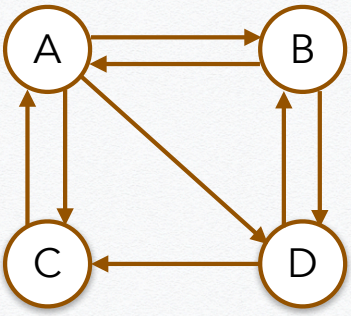
- For any ergodic Markov chain, there is a unique steady-state probability vector $\pi = [\pi_1, \pi_2, \ldots, \pi_n]^T$ that is the principal eigenvector of the transition matrix M, such that if $\eta(i, t)$ is the number of visits to state s_i in t steps, then

$$\lim_{t \to \infty} \frac{\eta(i,t)}{t} = \pi_i$$

- In other words, the probability distribution converges to a limiting distribution π with $M\pi = \pi$

PageRank

A tiny web



- PageRank: the stationary distribution (column) vector π
- π_i is the probably of the random surfer being at state s_i eventually
- Computation:

  Initialize $v := (1/n, \ldots, 1/n)$

  while (norm$(Mv - v) > \varepsilon$) {

  $v := Mv$

  }

M

0	1/2	1	0
1/3	0	0	1/2
1/3	0	0	1/2
1/3	1/2	0	0

v

1/4
1/4
1/4
1/4

Mv

9/24
5/24
5/24
5/24

M^2v

15/48
11/48
11/48
11/48

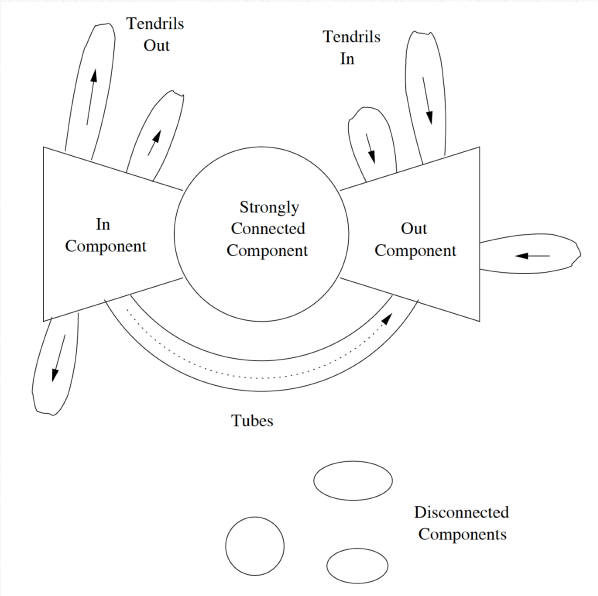
$\ldots \longrightarrow$

M^kv

3/9	PageRank of A
2/9	PageRank of B
2/9	PageRank of C
2/9	PageRank of D

Reality check: structure of the web

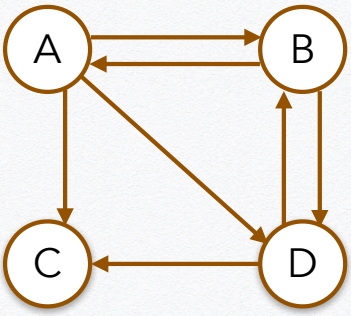
- The web is ***not*** strongly connected ☹
- The formulated Markov chain is not ergodic
- An early study of the web showed
  - One large strongly connected component
  - Several other components
- Requires modification to PageRank approach
- Two main problems
  1. Dead ends: a page with no outlink
  2. Spider traps: group of pages, outlinks only within themselves



Picture courtesy: book by Leskovec, Rajaraman and Ullman

Dead ends

A tiny web



- Let's make C a dead end
- *M* is not stochastic anymore, rather *substochastic*
  - The 3rd column sum = 0 (not 1)
- Now the iteration $v := Mv$ takes all probabilities to zero

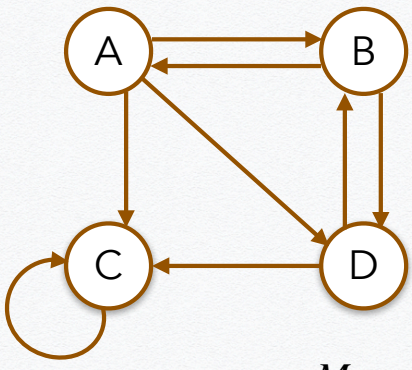
M

0	1/2	**0**	0
1/3	0	**0**	1/2
1/3	0	**0**	1/2
1/3	1/2	**0**	0

v	Mv	M^2v	...	
1/4	3/24	5/48	⟶	0
1/4	5/24	7/48		0
1/4	5/24	7/48		0
1/4	5/24	7/48		0

Spider traps

A tiny web



- Let C be a one node spider trap
- Now the iteration $v := Mv$ takes all probabilities to zero except the spider trap
- The spider trap gets all the PageRank

M

0	1/2	0	0
1/3	0	0	1/2
1/3	0	**1**	1/2
1/3	1/2	0	0

v	Mv	M^2v	...	
1/4	3/24	5/48	⟶	0
1/4	5/24	7/48		0
1/4	11/24	29/48		1
1/4	5/24	7/48		0

Teleportation

- Approach to handle dead-ends and spider traps
- Teleportation: the surfer may teleport to any other node with some probability
- Idealized PageRank: iterate $v_k = Mv_{k-1}$
- PageRank with teleportation

with probability β continue to an outlink

with probability $(1-\beta)$ teleport (leave and join at another node)

$$v_k = \beta M v_{k-1} + (1-\beta)\frac{\boldsymbol{e}}{n}$$

where β is a constant, usually between 0.8 and 0.9, and $\boldsymbol{e} = (1, \ldots, 1)$

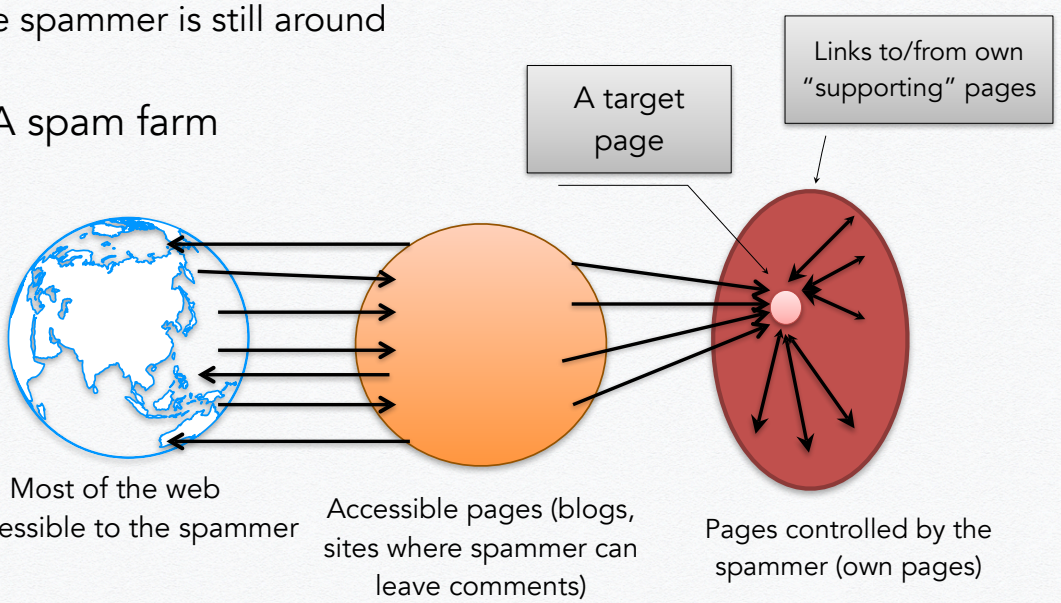
- Intuition: create artificial links to all other pages
- Then, the ergodicity property is also achieved

Link spam

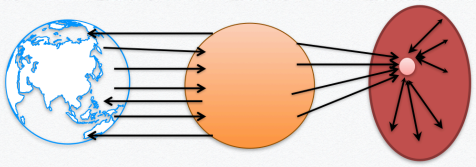
Google took care of the term spam, but …

The spammer is still around

A spam farm



Analysis of a spam farm



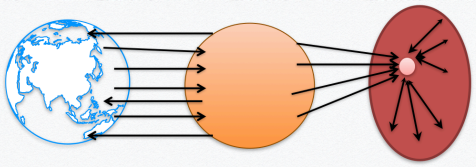
- Setting
  - Total #of pages in the web = n
  - Target page T, with m supporting pages
  - Let x be the PageRank contributed by accessible pages (sum of all PageRank of accessible pages times β)
  - How much y = PageRank of the target page can be?
- PageRank of every supporting page

$$\frac{\beta y}{m} + \frac{1-\beta}{n}$$

Contribution from the target page with PageRank y

Share of PageRank among all pages in the web

Analysis of a spam farm (continued)



- Three sources contribute to PageRank
  1. Contribution from accessible pages = x
  2. Contribution from supporting pages = $\beta\left(\frac{\beta y}{m} + \frac{1-\beta}{n}\right)$
  3. The n-th share of the fraction $(1-\beta)/n$ [negligible]
- So, we have $y = x + \beta m\left(\frac{\beta y}{m} + \frac{1-\beta}{n}\right)$
- Solving for y, we get $y = \frac{x}{1-\beta^2} + \frac{\beta}{1+\beta} \times \frac{m}{n}$
- If β = 0.85, then $y = 3.6x + 0.46\frac{m}{n}$
- External contribution up by 3.6 times, plus 46% of the fraction of the PageRank from the web

TrustRank and Personalied PageRank

- The teleportation scheme: $v_k = \beta \times M v_{k-1} + (1-\beta) \times \begin{bmatrix} 1/n \\ \vdots \\ 1/n \end{bmatrix}$

  Some nodes can be given higher priority by changing this vector here
- A set S of trustworthy pages where the spammers cannot place links
  - Wikipedia (after moderation), university pages, …
- Compute **TrustRank**: $v_k = \beta M v_{k-1} + (1-\beta)\frac{\boldsymbol{e}_S}{|S|}$

  where $\boldsymbol{e}_S$ is the vector with entry = 1 for all pages in S and 0 otherwise
- The random surfers are introduced only at trusted pages
- Spam mass = PageRank – TrustRank
- High spam mass $\Longrightarrow$ likely to be spam
- Similarly, **topic sensitive (personalized)** PageRank
  - For example, prioritize sports pages to create PageRank for users who like sports
  - Combine two or more topic-sensitive PageRanks making it suitable to user profiles

References and further reading

- Leskovec, Rajaraman and Ullman. Mining of Massive Datasets.
- Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Stanford InfoLab, 1999
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. "Introduction to Information Retrieval", Cambridge University Press. 2008.

Evaluation in Information Retrieval

Mandar Mitra

Indian Statistical Institute

Outline

1 Preliminaries

2 Metrics

3 Forums

4 Tasks
- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

Motivation

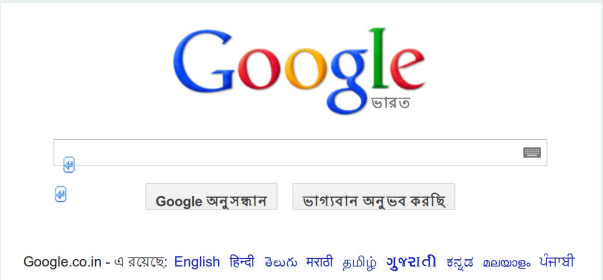
- Which is better: Heap sort or Bubble sort?

Motivation

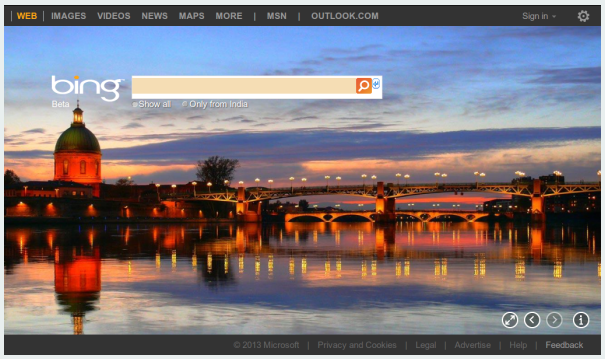
- Which is better: Heap sort or Bubble sort?

vs.

Which is better?



or



M. Mitra (ISI) Evaluation in Information Retrieval 3 / 57

Motivation

- IR is an *empirical* discipline.

Motivation

- IR is an *empirical* discipline.
- Intuition can be wrong!
  - "sophisticated" techniques need not be the best
    e.g. rule-based stemming vs. statistical stemming

Motivation

- IR is an *empirical* discipline.
- Intuition can be wrong!
  - "sophisticated" techniques need not be the best
    e.g. rule-based stemming vs. statistical stemming
- Proposed techniques need to be validated and compared to existing techniques.

Cranfield method (CLEVERDON ET AL., 60S)

Benchmark data

- Document collection
- Query / topic collection
- Relevance judgments - information about which document is relevant to which query

Cranfield method (CLEVERDON ET AL., 60S)

Benchmark data

- Document collection (syllabus)
- Query / topic collection (question paper)
- Relevance judgments - information about which document is relevant to which query (correct answers)

Cranfield method (CLEVERDON ET AL., 60S)

Benchmark data

- Document collection (syllabus)
- Query / topic collection (question paper)
- Relevance judgments - information about which document is relevant to which query (correct answers)

Assumptions

- relevance of a document to a query is objectively discernible
- all relevant documents contribute equally to the performance measures
- relevance of a document is independent of the relevance of other documents

Outline

1 Preliminaries

2 Metrics

3 Forums

4 Tasks
- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

Evaluation metrics

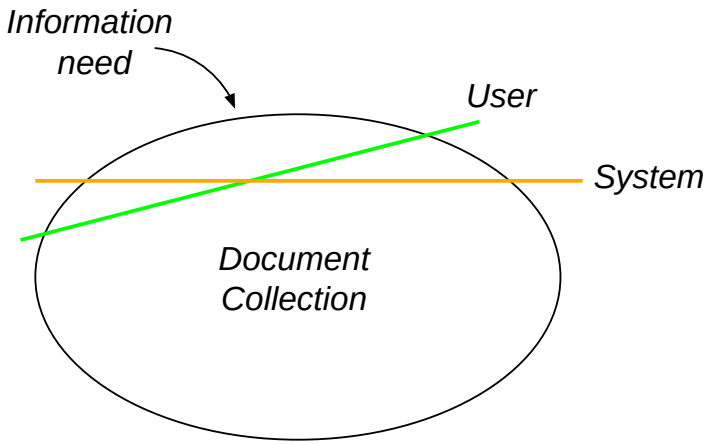
Background

- User has an information need.
- Information need is converted into a **query**.
- Documents are **relevant** or **non-relevant**.
- Ideal system retrieves all and only the relevant documents.

Evaluation metrics

Background

- User has an information need.
- Information need is converted into a **query**.
- Documents are **relevant** or **non-relevant**.
- Ideal system retrieves all and only the relevant documents.



Set-based metrics

$$\begin{aligned}\textbf{Recall} &= \frac{\#(\text{relevant retrieved})}{\#(\text{relevant})} \\ &= \frac{\#(\text{true positives})}{\#(\text{true positives + false negatives})}\end{aligned}$$

$$\begin{aligned}\textbf{Precision} &= \frac{\#(\text{relevant retrieved})}{\#(\text{retrieved})} \\ &= \frac{\#(\text{true positives})}{\#(\text{true positives + false positives})}\end{aligned}$$

$$\begin{aligned}\mathbf{F} &= \frac{1}{\alpha/P + (1-\alpha)/R} \\ &= \frac{(\beta^2+1)PR}{\beta^2 P + R}\end{aligned}$$

Metrics for ranked results

(Non-interpolated) average precision

Which is better?

1. Non-relevant
2. Non-relevant
3. Non-relevant
4. Relevant
5. Relevant

1. Relevant
2. Relevant
3. Non-relevant
4. Non-relevant
5. Non-relevant

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

$$AvgP = \frac{1}{5}(1 + \frac{2}{3} + \frac{3}{6})$$

(5 relevant docs. in all)

Metrics for ranked results

(Non-interpolated) average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

$$AvgP = \frac{1}{5}(1 + \frac{2}{3} + \frac{3}{6})$$

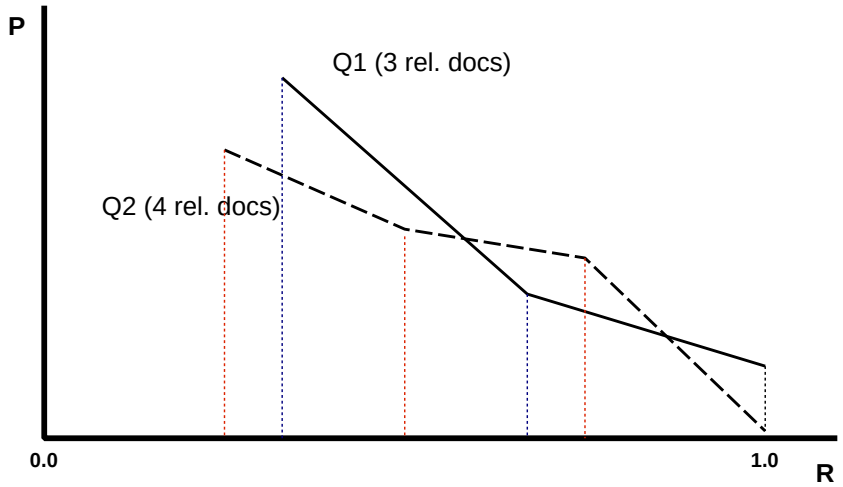
(5 relevant docs. in all)

$$\boxed{AvgP = \frac{1}{N_{Rel}} \sum_{d_i \in Rel} \frac{i}{Rank(d_i)}}$$

Metrics for ranked results

Interpolated average precision at a given recall point

- Recall points correspond to $\frac{1}{N_{Rel}}$
- N_{Rel} different for different queries



- Interpolation required to compute averages across queries

Metrics for ranked results

Interpolated average precision

$$P_{int}(r) = \max_{r' \geq r} P(r')$$

Metrics for ranked results

Interpolated average precision

$$P_{int}(r) = \max_{r' \geq r} P(r')$$

11-pt interpolated average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

Metrics for ranked results

Interpolated average precision

$$P_{int}(r) = \max_{r' \geq r} P(r')$$

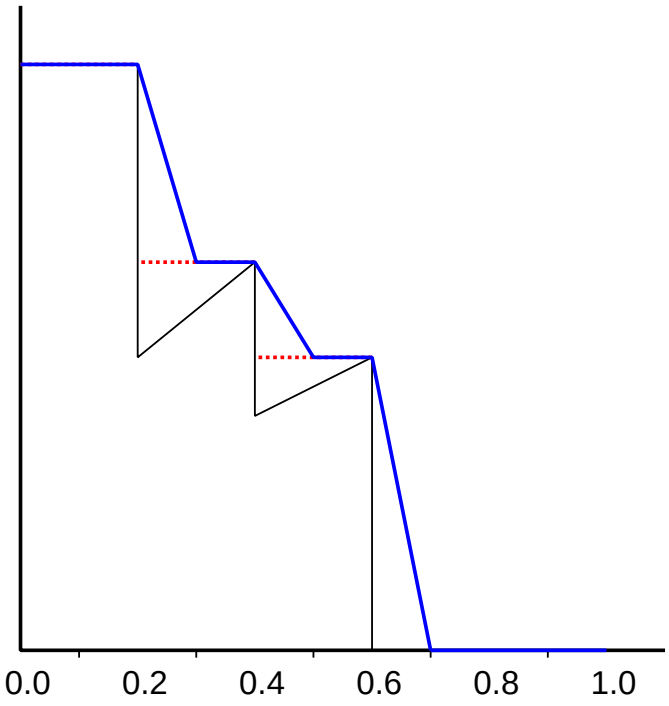
11-pt interpolated average precision

Rank	Type	Recall	Precision
1	Relevant	0.2	1.00
2	Non-relevant		
3	Relevant	0.4	0.67
4	Non-relevant		
5	Non-relevant		
6	Relevant	0.6	0.50
∞	Relevant	0.8	0.00
∞	Relevant	1.0	0.00

R	Interp. P
0.0	1.00
0.1	1.00
0.2	1.00
0.3	0.67
0.4	0.67
0.5	0.50
0.6	0.50
0.7	0.00
0.8	0.00
0.9	0.00
1.0	0.00

Metrics for ranked results

11-pt interpolated average precision



Metrics for sub-document retrieval

Let p_r - document part retrieved at rank r
$rsize(p_r)$ - amount of relevant text contained by p_r
$size(p_r)$ - total number of characters contained by p_r
T_{rel} - total amount of relevant text for a given topic

$$P[r] = \frac{\sum_{i=1}^{r} rsize(p_i)}{\sum_{i=1}^{r} size(p_i)}$$

$$R[r] = \frac{1}{T_{rel}} \sum_{i=1}^{r} rsize(p_i)$$

Metrics for ranked results

- **Precision at k (P@k)** - precision after **k** documents have been retrieved
  - easy to interpret
  - not very stable / discriminatory
  - does not average well
- **R precision** - precision after N_{Rel} documents have been retrieved

Cumulated Gain

Idea:

- Highly relevant documents are more valuable than marginally relevant documents
- Documents ranked low are less valuable

Cumulated Gain

Idea:

- Highly relevant documents are more valuable than marginally relevant documents
- Documents ranked low are less valuable

$$Gain \in \{0, 1, 2, 3\}$$

$$G = \langle 3, 2, 3, 0, 0, 1, 2, 2, 3, 0, \ldots \rangle$$

$$CG[i] = \sum_{j=1}^{i} G[i]$$

(n)DCG

$$DCG[i] = \begin{cases} CG[i] & \text{if } i < b \\ DCG[i-1] + G[i]/\log_b i & \text{if } i \geq b \end{cases}$$

(n)DCG

$$DCG[i] = \begin{cases} CG[i] & \text{if } i < b \\ DCG[i-1] + G[i]/\log_b i & \text{if } i \geq b \end{cases}$$

$$\textbf{Ideal } G = \langle 3, 3, \ldots, 3, 2, \ldots, 2, 1, \ldots, 1, 0, \ldots \rangle$$

$$nDCG[i] = \frac{DCG[i]}{\textbf{Ideal } DCG[i]}$$

Mean Reciprocal Rank

- Useful for *known-item* searches with a single target
- Let r_i — rank at which the "answer" for query i is retrieved. Then reciprocal rank = $1/r_i$

  Mean reciprocal rank (MRR) = $\sum_{i=1}^{n} \frac{1}{r_i}$

Assumptions

- All relevant documents contribute equally to the performance measures.
- Relevance of a document to a query is objectively discernible.
- Relevance of a document is independent of the relevance of other documents.

Assumptions

- All relevant documents contribute equally to the performance measures.
- Relevance of a document to a query is objectively discernible.
- Relevance of a document is independent of the relevance of other documents.
- All relevant documents in the collection are known.

Assessor agreement

- Judges / assessors may not agree about relevance.

Example (MANNING ET AL.)

	Yes_1	No_1	Total_2
Yes_2	300	20	320
No_2	10	70	80
Total_1	310	90	400

$$P(A) = (300 + 70)/400 = 370/400 = 0.925$$
$$P(\text{nrel}) = (80 + 90)/(400 + 400) = 0.2125$$
$$P(\text{rel}) = (320 + 310)/(400 + 400) = 0.7878$$
$$P(E) = P(\text{non-rel})^2 + P(\text{rel})^2 = 0.665$$
$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} = \frac{0.925 - 0.665}{1 - 0.665} = 0.776$$

- Rules of thumb:
  $\kappa > 0.8$ — good agreement
  $0.67 \leq \kappa \leq 0.8$ — fair agreement
  $\kappa < 0.67$ — poor agreement

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*
- A wide variety of models, retrieval algorithms is important.
- **Manual interactive retrieval** is a must.

Pooling

- Exhaustive relevance judgments may be infeasible.
- Pool top results obtained by various systems and assess the pool.
- *Unjudged documents are assumed to be non-relevant.*
- A wide variety of models, retrieval algorithms is important.
- **Manual interactive retrieval** is a must.

Can unbiased, incomplete relevance judgments be used to reliably compare the relative effectiveness of different retrieval strategies?

Bpref

- Based on number of times judged nonrelevant documents are retrieved before relevant documents

  Let R - set of relevant documents for a topic
  N - set of first $|R|$ judged non-rel docs retrieved

$$bpref = \frac{1}{|R|} \sum_{r \in R} (1 - \frac{|n \text{ ranked higher than } r|}{|R|})$$

$$bpref10 = \frac{1}{|R|} \sum_{r \in R} (1 - \frac{|n \text{ ranked higher than } r|}{10 + |R|})$$

Bpref

- With complete judgments:
  system rankings generated based on MAP and bpref10 are highly correlated
- When judgments are incomplete:
  system rankings generated based on bpref10 are more stable

Outline

1 Preliminaries

2 Metrics

3 Forums

4 Tasks
- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

TREC

`http://trec.nist.gov`

- Organized by NIST every year since 1992
- Typical tasks
  - adhoc
    - user enters a search topic for a one-time information need
    - document collection is static
  - routing/filtering
    - user's information need is persistent
    - document collection is a stream of incoming documents
  - question answering

TREC data

- Documents
  - Genres:
    - news (AP, LA Times, WSJ, SJMN, Financial Times, FBIS)
    - govt. documents (Federal Register, Congressional Records)
    - technical articles (Ziff Davis, DOE abstracts)
  - Size: 0.8 million documents – 1.7 million web pages
    (cf. Google indexes several billion pages)
- Topics
  - title
  - description
  - narrative

Enterprise track

- Goal: work with enterprise data (intranet pages, email archives, document repositories)
- Corpus: crawled from W3C
  - lists.w3.org: $\sim$ 200,000 docs, $\sim$ 2 GB
  - documents are html-ised archives of mailing lists (email header information recoverable)

Enterprise track tasks

Known item search

- Scenario: user searches for a particular message that is known to exist
- Test data: topic + corresponding (unique) message
- Measures: mean reciprocal rank (MRR), success at 10 docs.
- Results: best groups obtained MRR $\approx$ 0.6, S@10 $\approx$ 0.8

Enterprise track tasks

Discussion search

- Scenario: user searches for an argument / discussion on an issue
- Test data: topic / issue + relevant messag
  endframe
  - irrelevant
  - partially relevant (does not take a stand)
  - relevant (takes a pro/con stand)
- Measures: MAP, R-precision, etc.
- Results: “strict” and “lenient” evaluations were strongly correlated

Enterprise track tasks

Expert search

- Scenario: user searches for names of persons who are experts on a specified topic
- Test data: working groups of the W3C, members of the working groups
- Measures: MAP, R-precision, etc.

Enterprise track

Hiccups

- Discussion search: no dry runs
- Some topics more amenable than others to a pro/con discussion
- Relevance judgments do not include orientation information
- Assessor agreement: ranking done on the basis of *primary* and *secondary* assessors correlated, but not “essentially identical”

CLEF

`http://www.clef-campaign.org/`

- CLIR track at TREC-6 (1997), CLEF started in 2000
- Objectives:
  - to provide an infrastructure for the testing and evaluation of information retrieval systems operating on European languages in both monolingual and cross-language contexts
  - to construct test-suites of reusable data that can be employed by system developers for benchmarking purposes
  - to create an R&D community in the cross-language information retrieval (CLIR) sector

CLEF tasks

- Monolingual retrieval
- Bilingual retrieval
  - queries in language X
  - document collection in language Y
- Multi-lingual retrieval
  - queries in language X
  - multilingual collection of documents (e.g. English, French, German, Italian)
  - results include documents from various collections and languages in a single list
- Other tasks: spoken document retrieval, image retrieval

NTCIR

`http://research.nii.ac.jp/ntcir`

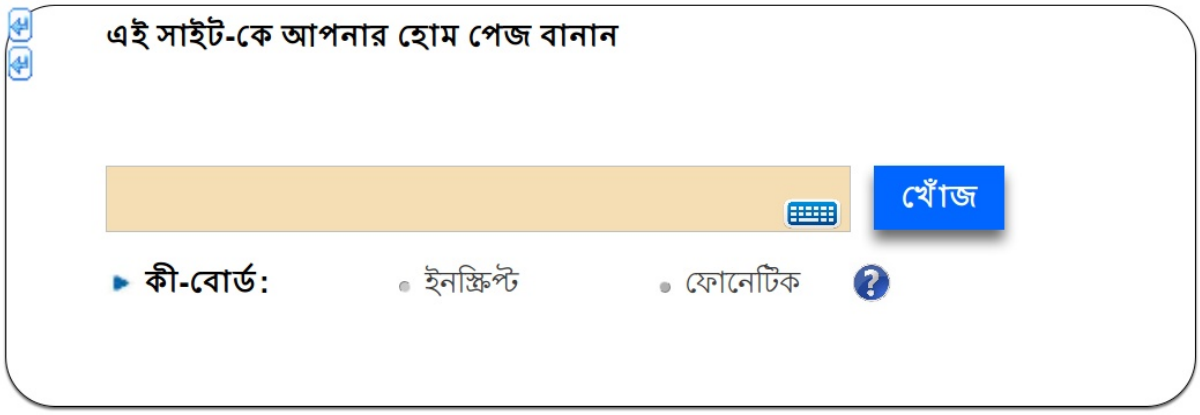
- Started in late 1997
- Held every 1.5 years at NII, Japan
- Focus on East Asian languages (Chinese, Japanese, Korean)
- Tasks
  - cross-lingual retrieval
  - patent retrieval
  - geographic IR
  - opinion analysis

FIRE

- Forum for Information Retrieval Evaluation
  `http://www.isical.ac.in/~fire`
- Evaluation component of a DIT-sponsored, consortium mode project
- Assigned task: create a portal where
  1. a user will be able to give a query in one Indian language;
  2. s/he will be able to access documents available in the language of the query, Hindi (if the query language is not Hindi), and English,
  3. all presented to the user in the language of the query.
- Languages: Bangla, Hindi, Marathi, Punjabi, Tamil, Telugu

Sandhan: a search engine

`http://tdil-dc.in/sandhan`



हिन्दी | বাংলা | मराठी | తెలుగు | தமிழ்

FIRE: goals

- To encourage research in South Asian language Information Access technologies by providing reusable large-scale test collections for ILIR experiments
- To provide a common evaluation infrastructure for comparing the performance of different IR systems
- To explore new Information Retrieval / Access tasks that arise as our information needs evolve, and new needs emerge
- To investigate evaluation methods for Information Access techniques and methods for constructing a reusable large-scale data set for ILIR experiments.
- To build language resources for IR and related language processing tasks

FIRE: tasks

- Ad-hoc monolingual retrieval
  - Bengali, Hindi Marathi and English
- Ad-hoc cross-lingual document retrieval
  - documents in Bengali, Hindi, Marathi, and English
  - queries in Bengali, Hindi, Marathi, Tamil, Telugu , Gujarati and English
  - Roman transliterations of Bengali and Hindi topics
- CL!NSS: Cross-Language !ndian News Story Search
- MET: Morpheme Extraction Task (MET)
- RISOT: Retrieval from Indic Script OCR'd Text
- SMS-based FAQ Retrieval
- Older tracks:
  - Retrieval and classification from mailing lists and forums

FIRE: datasets

Documents

- Bengali: Anandabazar Patrika (123,047 docs)
- Hindi: Dainik Jagran (95,215 docs) +
  Amar Ujala (54,266 docs)
- Marathi: Maharashtra Times, Sakal (99,275 docs)
- English: Telegraph (125,586 docs)

- All from the Sep 2004 - Sep 2007 period
- All content converted to UTF-8
- Minimal markup

FIRE: datasets

Topics

- 225 topics
- Queries formulated parallely in Bengali, Hindi by browsing the corpus
- Refined based on initial retrieval results
  - ensure minimum number of relevant documents per query
  - balance easy, medium and hard queries
- Translated into Marathi, Tamil, Telugu , Gujarati and English
- TREC format (title + desc + narr)

FIRE: topics

Example:

```
<title> Nobel theft

<desc>
Rabindranath Tagore's Nobel Prize medal was stolen from Santiniketan. The
document should contain information about this theft.

<narr>
A relevant document should contain information regarding the
missing Nobel Prize Medal that was stolen along with some other
artefacts and paintings on 25th March, 2004. Documents containing
reports related to investigations by government agencies like CBI
/ CID are also relevant, as are articles that describe public
reaction and expressions of outrage by various political parties.
```


Participants

AU-KBC	Dublin City U.
IIT Bombay	U. of Maryland
Jadavpur U.	U. Neuchatel, Switzerland
IBM	U. North Texas
Microsoft Research	U. Tampere

Outline

1 Preliminaries

2 Metrics

3 Forums

4 Tasks
- Task 1: Morpheme extraction
- Task 2: RISOT
- Task 3: SMS-based FAQ retrieval
- Task 4: Microblog retrieval

Task overview

Morpheme: smallest meaningful units of language

Task overview

Morpheme: smallest meaningful units of language

- Objective: discover morphemes in (morphologically rich) Indian languages
- Started in 2012
- Offered in Bengali, Gujarati, Hindi, Marathi, Odia, Tamil

Details

- Participants need to submit a working program (morpheme extraction system)
- Specifications:
  Input: large lexicon (provided as test data to the participants)
  Output: two column file containing list of words + morphemes, separated by tab
- Evaluation: use system output for *stemming* within an IR system
  - System: TERRIER (http://www.terrier.org)
  - Task: FIRE 2011 adhoc task
  - Metric: MAP

Results

Bengali

Team	MAP
Baseline	0.2740
CVPR-Team1	0.3159
DCU	0.3300
IIT-KGP	0.3225
ISM	0.3013
JU	**0.3307**

Task overview

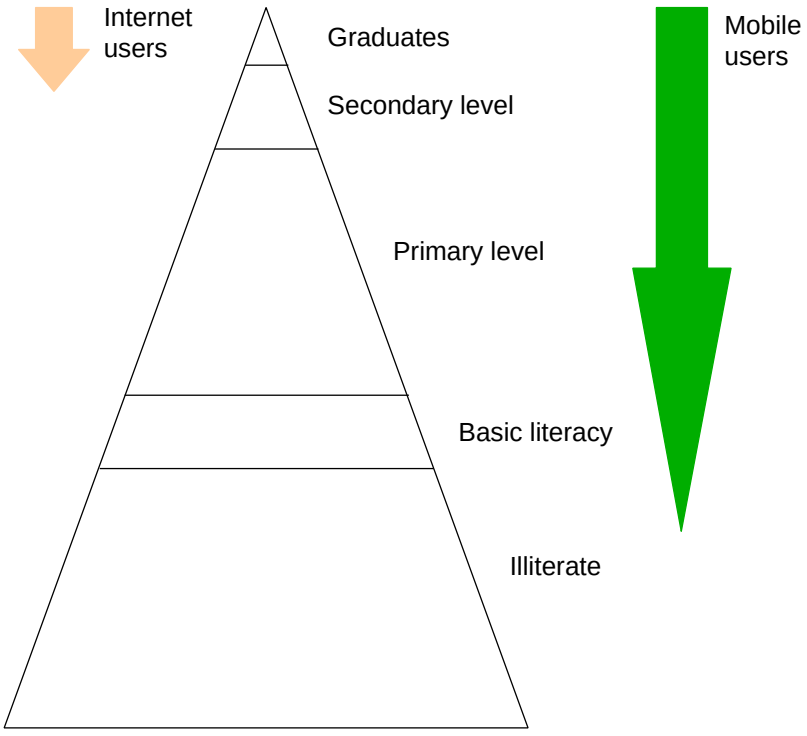
RISOT: retrieval of Indic script OCR'd text

- Data
  - 62,875 Bengali newspaper articles from FIRE 2008/2010 collection + 66 topics
  - 94,432 Hindi newspaper articles + 28 topics
- Documents automatically rendered and OCR'd

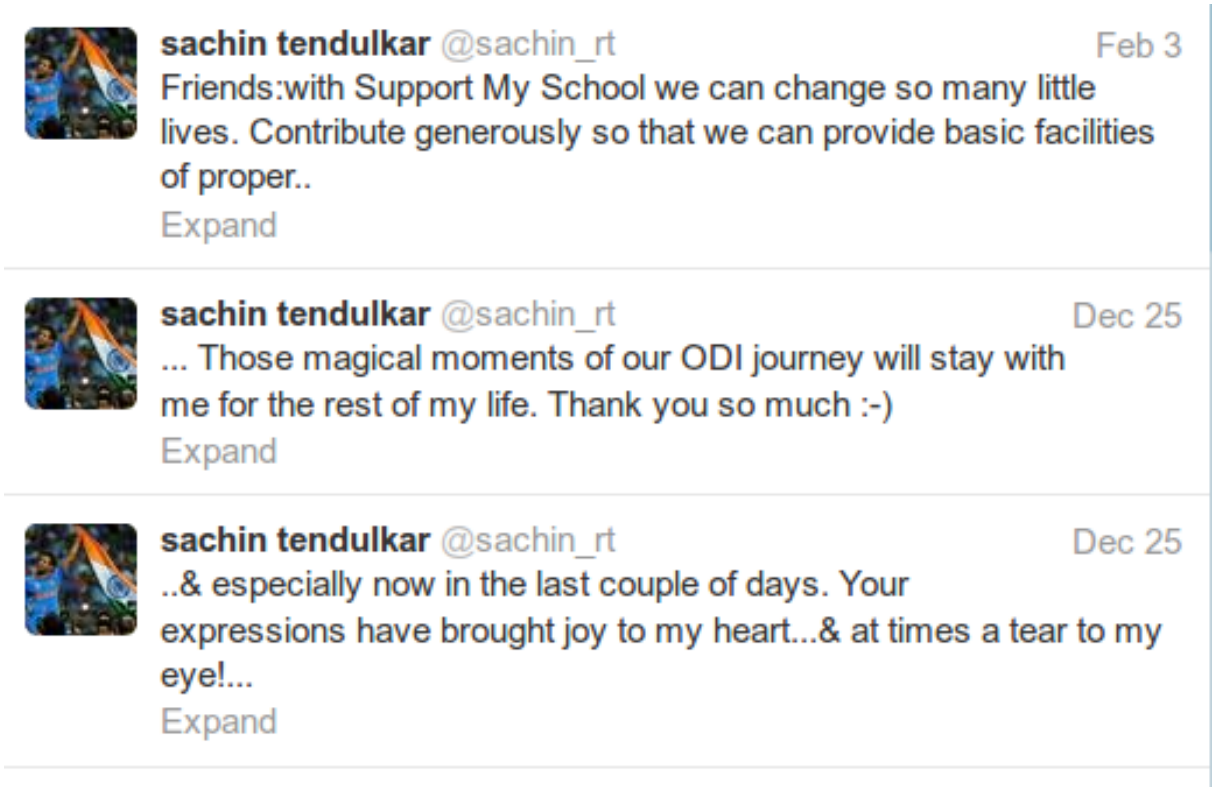
Results

Source	MAP
Clean text	0.2567
OCR'd text	0.1791
OCR'd text + processing	0.1974

Motivation



Background



Challenges

Vocabulary mismatch

- Tweets are short (maximum of 140 characters each)
- Not always written maintaining formal grammar and proper spelling

Challenges

Vocabulary mismatch

- Tweets are short (maximum of 140 characters each)
- Not always written maintaining formal grammar and proper spelling

$\Rightarrow$ Query keywords may not match a relevant tweet

Query expansion

Definition. Add related words to query.

Example:

harmful effects of tobacco

Query expansion

Definition. Add related words to query.

Example:

harmful effects of tobacco

$\Downarrow$

+ health, cancer, cigarette, smoking, gutkha

Relevance feedback

- Original query is used to retrieve some number of documents.
- User examines some of the retrieved documents and provides feedback about which documents are relevant and which are non-relevant.
- System uses the feedback to “learn” a better query:
  - select/emphasize words that occur more frequently in relevant documents than non-relevant documents;
  - eliminate/de-emphasize words that occur more frequently in non-relevant than in relevant documents.
- Resulting query should bring in more relevant documents and fewer non-relevant documents.

Relevance feedback

- Original query is used to retrieve some number of documents.
- User examines some of the retrieved documents and provides feedback about which documents are relevant and which are non-relevant.
- System uses the feedback to "learn" a better query:
  - select/emphasize words that occur more frequently in relevant documents than non-relevant documents;
  - eliminate/de-emphasize words that occur more frequently in non-relevant than in relevant documents.
- Resulting query should bring in more relevant documents and fewer non-relevant documents.

Blind/adhoc/pseudo relevance feedback: In the absence of feedback, *assume* top-ranked documents are relevant.

Query expansion for tweet retrieval

Bandyopadhyay et al., IJWS 2013

Query reformulation

- Initial query is used to retrieve d tweets
- *All distinct words* occurring in these tweets are used as new query

Query expansion for tweet retrieval

Bandyopadhyay et al., IJWS 2013

Query reformulation

- Initial query is used to retrieve d tweets
- *All distinct words* occurring in these tweets are used as new query

Collection enrichment: Use external source, e.g. Google

TREC 2011 microblog task

HTTP status	No. of tweets
Total crawled	**16,087,002**
301 (moved permanently)	987,866
302 (found but retweet)	1,054,459
403 (forbidden)	404,549
404 (not found)	458,388
Unknown[3]	3
200 (OK)	13,181,737
After filtering	**9,363,521**

Results

Run Name	P@30	MAP
B	0.3231	0.1938
PRF	0.3578 (+10.74%)	0.2283 (+17.80%)
QR	0.3891 (+20.43%)	0.2515 (+29.77%)
QR+PRF	0.4150 (+28.44%)	0.2754 (+42.11%)
TGQR	0.4218 (+30.55%)	0.2824 (+45.72%)
TGQE	0.4238 (+31.17%)	0.2819 (+45.46%)

References

- *Introduction to Modern Information Retrieval.* Salton, McGill. McGraw Hill, 1983.
- *An Introduction to Information Retrieval.* Manning, Raghavan, Schutze.

  `http://www-csli.stanford.edu/~schuetze/information-retrieval-book.html`
- *Retrieval Evaluation with Incomplete Information.* Buckley, Voorhees. SIGIR 2004.
- `http://trec.nist.gov`
- *Cross-Language Evaluation Forum: Objectives, Results, Achievements.* Braschler, Peters. Information Retrieval, 7:12, 2004.
- `http://research.nii.ac.jp/ntcir`