

Big Data (CS-3032)

**Kalinga Institute of Industrial Technology
Deemed to be University
Bhubaneswar-751024**

School of Computer Engineering



Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

3 Credit

Lecture Note

Course Contents



Sr #	Major and Detailed Coverage Area	Hrs
3	Big Data Streaming Introduction to Streams Concepts – Stream data model and architecture – Stream Computing, Sampling data in a stream – Filtering streams, Counting distinct elements in a stream.	8

Data Stream



Data Stream – Large data volume, likely unstructured and structured arriving at a very high rate, which requires real time/near real time analysis for effective decision making.

- ❑ It is basically continuously generated data and arrives in a stream (sequence of data elements made available over time). It is generally time-stamped and geo-tagged (in the form of latitude and longitude)
- ❑ Stream is composed of synchronized sequence of elements or events.
- ❑ If it is not processed immediately, then it is lost forever (No Real Time Significance).
- ❑ In general, such data is generated as part of application logs, events, or collected from a large pool of devices continuously generating events such as ATM or PoS

Example:

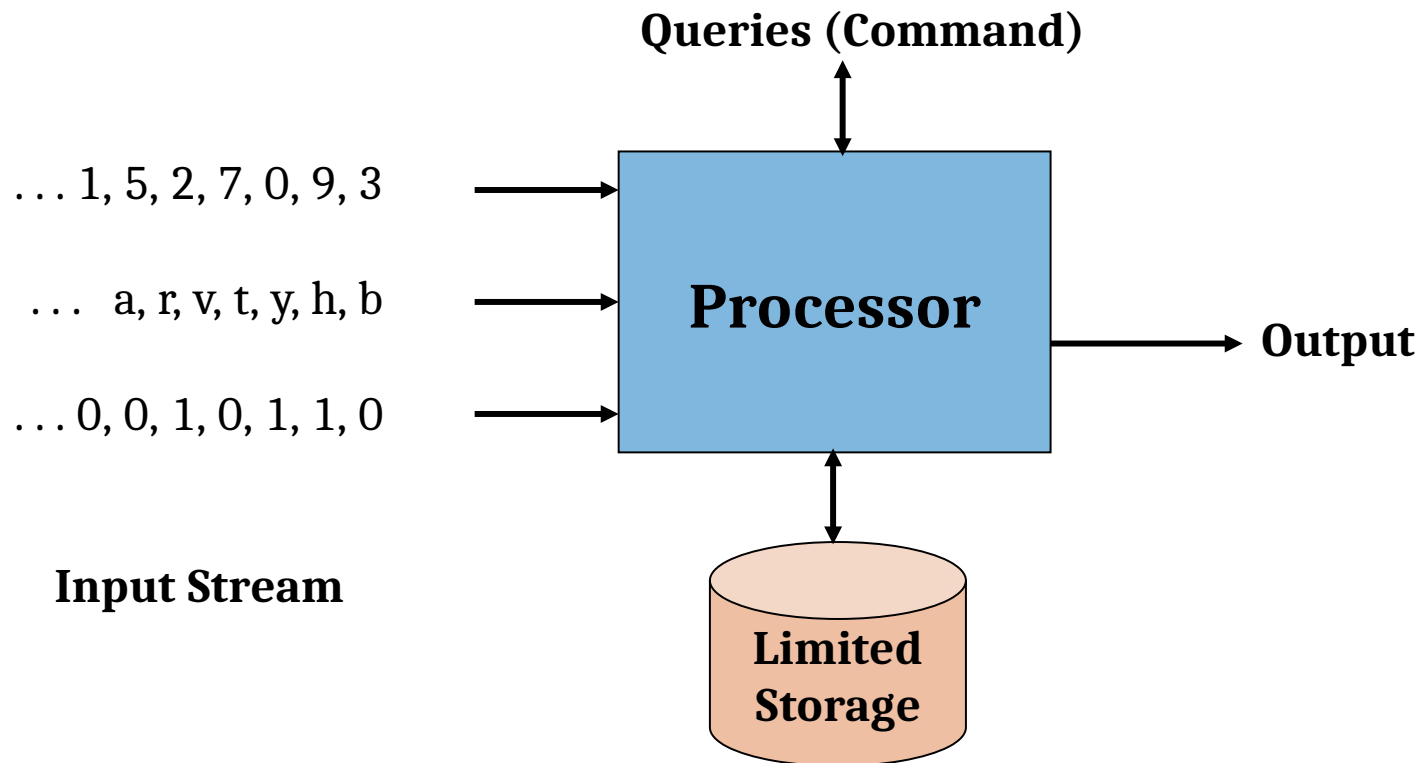
Data Center: Large network deployment of a data center with hundreds of servers, switches, routers and other devices in the network. The event logs from all these devices at real time create a stream of data. This data can be used to prevent failures in the data center and automate triggers so that the complete data center is fault tolerant

Stock Market: The data generated here is a stream of data where a lot of events are happening in real-time. The price of stock are continuously varying. These are large continuous data streams which needs analysis in real-time for better decisions on trading

Basic Model of Stream data



- ❑ Input data come rapidly and streams needn't have the same data rates or data types
- ❑ The system cannot store the data entirely
- ❑ Queries tends to ask information about recent data



Big Data Streaming



Big data streaming is a process in which big data is quickly processed in order to extract real-time insights from it. The data on which processing is done is the data in motion. Big data streaming is ideally a speed-focused approach wherein a continuous stream of data is processed.

For real-time big data stream processing, three key attributes are required:

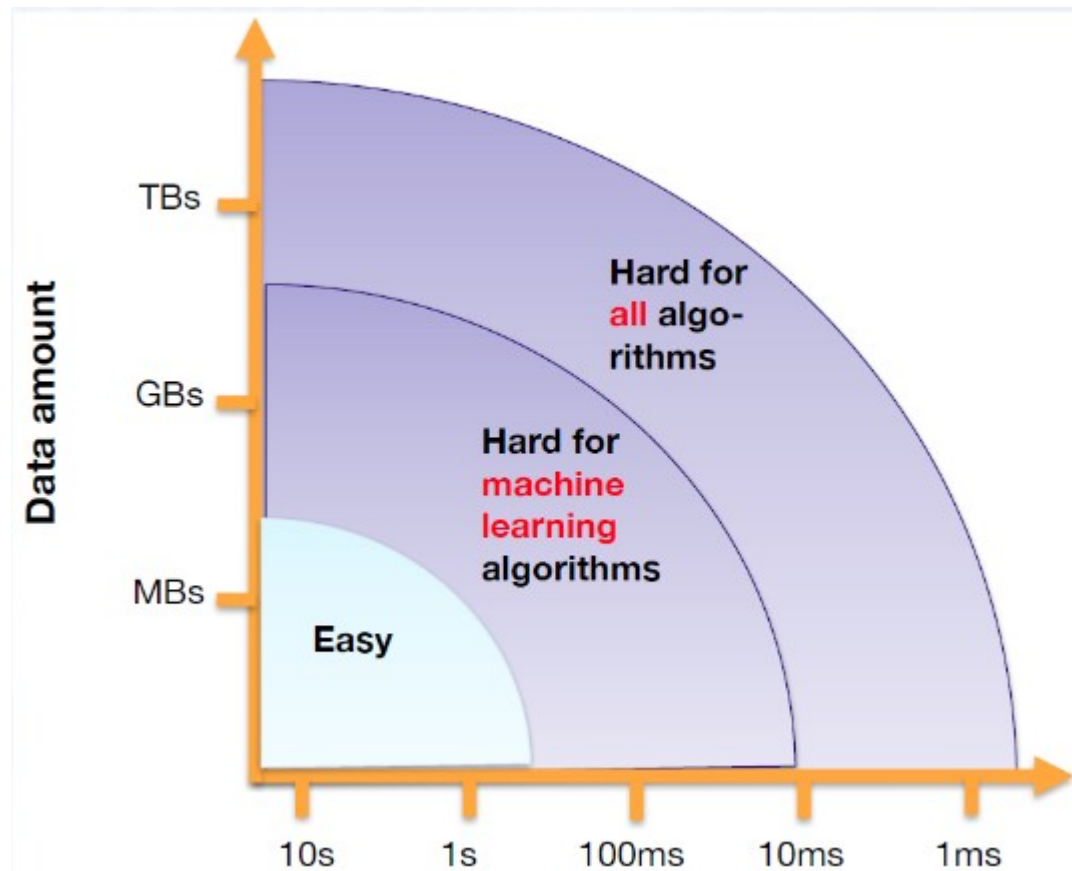
- ❑ System to collect the big data generated in real time
- ❑ System to handle the massive parallel processing of this data
- ❑ Event correlation and event processing engine for generating analytics

All the above mentioned attributes / components need to be fault tolerant, scalable and distributed, with low latency for each system.

Example of Big Data Streaming Tools: Apache Flink

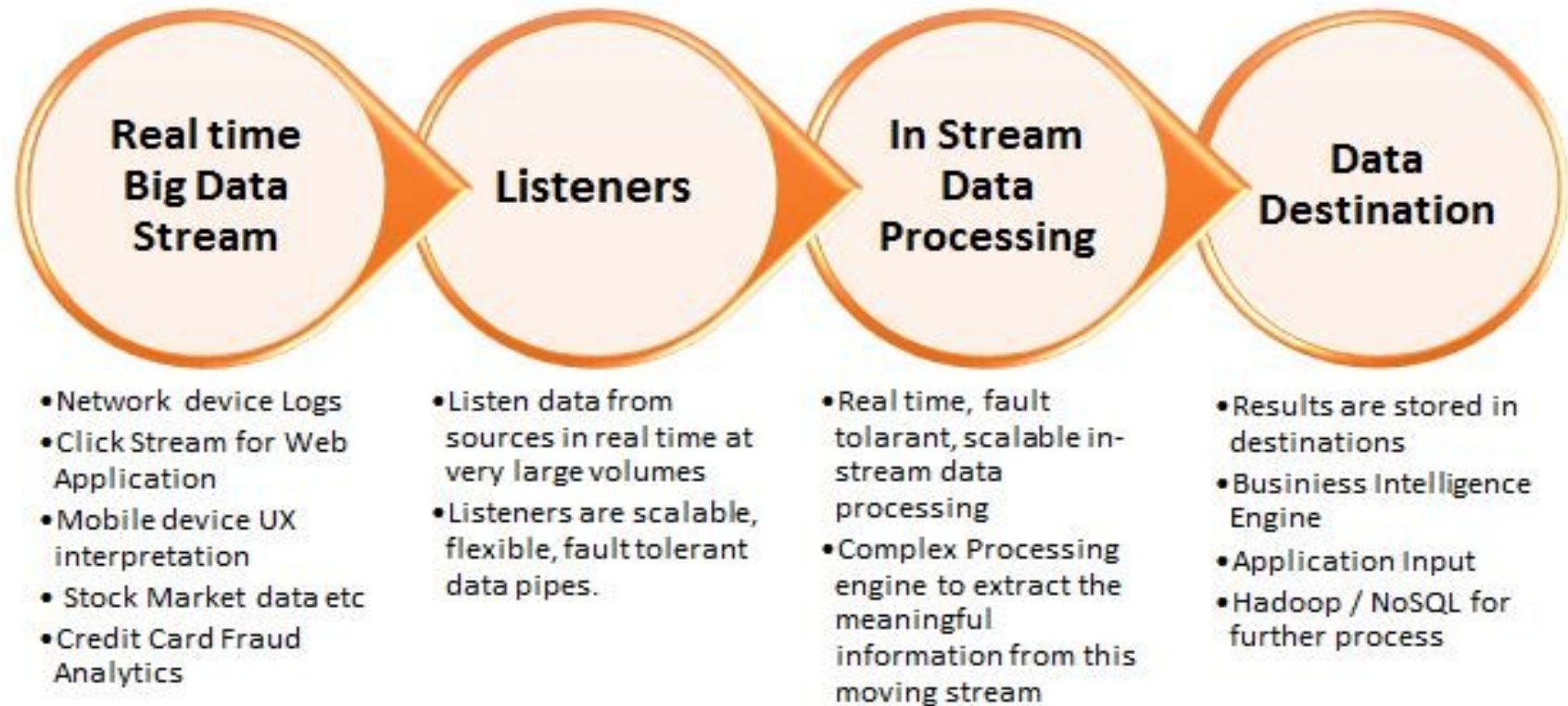
It captures streaming data on Hadoop. Flink can process millions of events in milliseconds. It also employs machine learning and graph processing techniques to handle complex event processing. Can be connected to YARN, Kubernetes etc.

Why Big Data Streaming is hard?



Source: Tilmann Rabl, Big Data Stream Processing

Big Data Streaming Process

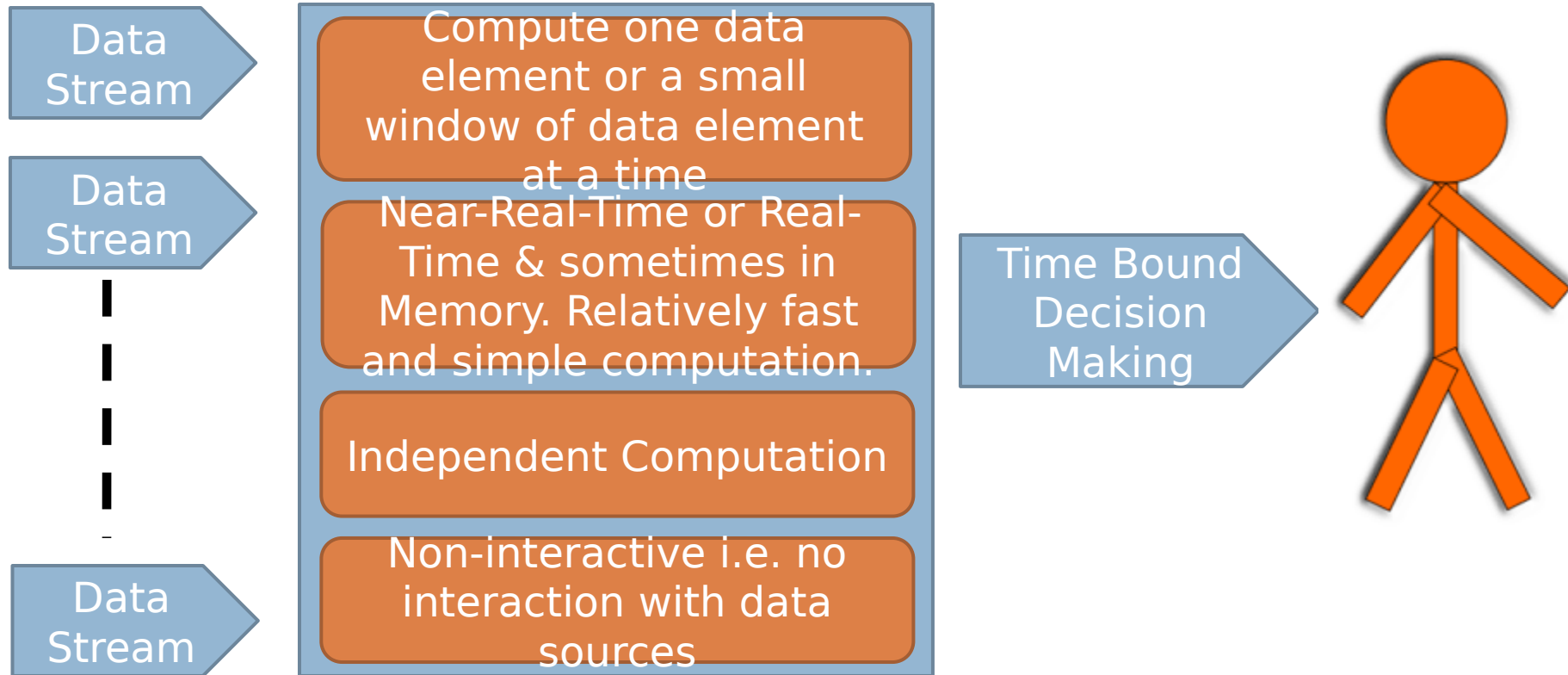


Source: <https://blog.blazeclan.com/>

Streaming Data System



Streaming Data System



Data-at-Rest vs. Data-in-Motion

- ❑ Data-at-rest - This refers to data that has been collected from various sources and is then analyzed after the event occurs. The point where the data is analyzed and the point where action is taken on it occur at two separate times. For example, a retailer analyzes a previous month's sales data and uses it to make strategic decisions about the present month's business activities. The action takes place after the data-creating event has occurred. For data at rest, a batch processing method would be most likely.
- ❑ Data-in-motion - The collection process for data in motion is similar to that of data at rest; however, the difference lies in the analytics. In this case, the analytics occur in real-time as the event happens. For example – sensor data from self-driving vehicles. For data in motion, you'd want to utilize a real-time processing method.

Data-at-Rest vs. Data-in-Motion Infrastructure Option



Data-at-rest



Public Cloud

Public cloud can be an ideal infrastructure choice in such scenario from a cost standpoint, since virtual machines can easily be spun up as needed to analyze the data and spun down when finished.

Data-in-motion



Bare-Metal Cloud

Bare-Metal cloud can be an preferable infrastructure choice. It involves the use of dedicated servers that offers cloud-like features without the use of virtualization.

Streaming Data Changes over Time



Change can be periodic or sporadic

Periodic: evening,
weekends etc

People post Facebook messages more in the evening in comparison to during day, working hours.

Sporadic: major
events

BREAKING NEWS



In summary, streaming data:

- ❑ **Size is unbounded** i.e. is continually generated and can't process all at once
- ❑ **Size and Frequency is unpredictable due to human behavior**
- ❑ **Processing must be relatively fast and simple**

Stream Computing



A high-performance computer system that analyzes multiple data streams from many sources. The word stream in stream computing is used to mean pulling in streams of data, processing the data and streaming it back out as a single flow. Stream computing uses software algorithms that analyzes the data in real time as it streams in to increase speed and accuracy when dealing with data handling and analysis.

Traditional Computing	Stream Computing
Historical fact finding	Current fact finding
Find and analyze information stored on disk	Analyze data in motion – before it is stored
Batch paradigm, pull model	Low latency paradigm, push model
Query-driven – queries data to static data	Data-driven – bring data to the analytics
	

Query Types on Data Stream



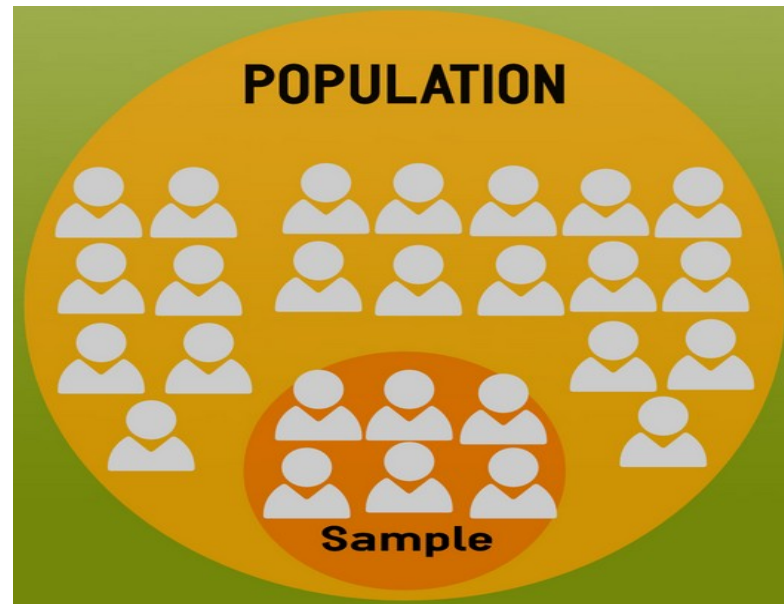
Types of queries one wants on answer on a data stream:

- ❑ **Sampling data from a stream** - construct a random sample
- ❑ **Queries over sliding windows** - number of items of type x in the last k elements of the stream
- ❑ **Filtering a data stream** - select elements with property x from the stream
- ❑ **Counting distinct elements** - number of distinct elements in the last k elements of the stream or in the entire stream
- ❑ **Estimating moments** - estimate avg./std. dev. of last k elements
- ❑ **Finding frequent elements**

Data Sampling



Data sampling is a technique used to select, manipulate and analyze a representative subset of data points to identify patterns and trends in the larger data set being examined. It enables data scientists, predictive modelers and other data analysts to work with a small, manageable amount of data to build and run analytical models more quickly, while still producing accurate findings.



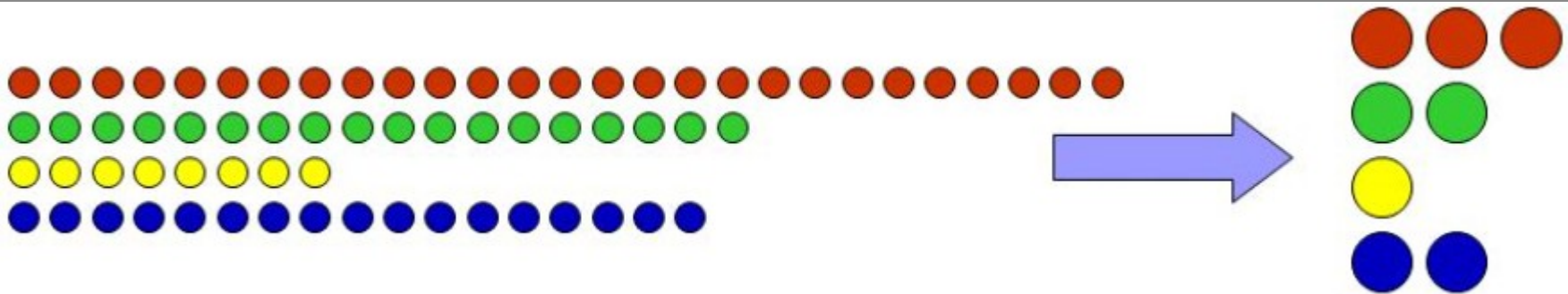
Source: Google Analytics Data Sampling

Why Sampling is Important?



- ❑ **Resource Constraints:** If the target population is not small enough, or if the resources at the disposal don't give the bandwidth to cover the entire population, it is important to identify a subset of the population to work with – a carefully identified group that is representative of the population. This process is called survey sampling. Whatever the sample size, there are fixed costs associated with any survey. Once the survey has begun, the marginal costs associated with gathering more information, from more people, are proportional to the size of the sample.
- ❑ **Drawing Inferences About the Population:** Researchers are not interested in the sample itself, but in the understanding that they can potentially infer from the sample and then apply across the entire population. Working within a given resource constraint, sampling may make it possible to study the population of a larger geographical area or to find out more about the same population by examining an area in greater depth through a smaller sample.

Sampling data in Stream



Need & how of sampling - System cannot store the entire stream conveniently, so

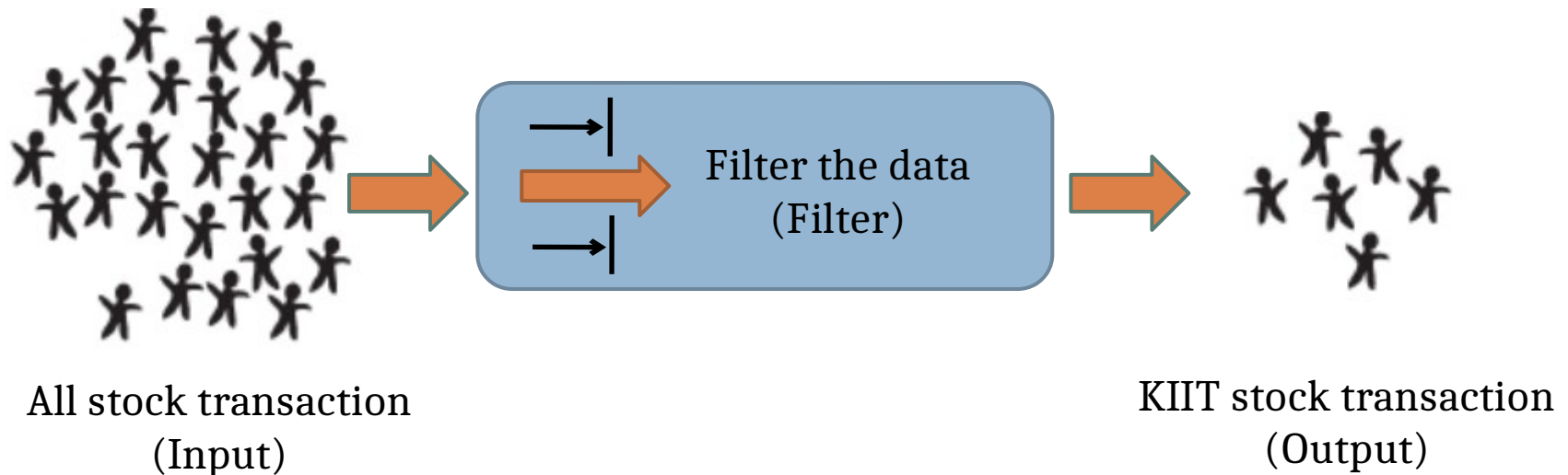
- ☐ how to make critical calculations about the stream using a limited amount of (primary or secondary) memory?
- ☐ Don't know how long the stream is, so when and how often to sample?

Whenever a list or set is used, and space is consideration, a **Bloom filter** should be considered.

Filter



A filter is a program or section of code that is designed to examine each input or output stream for certain qualifying criteria and then process or forward it accordingly by producing another stream.



In this example, the streams processing application needs to filter the stock transaction data for KIIT transaction records.

Bloom Filter



Bloom filter is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. **False positive** matches are possible, but **False Negatives are not** – in other words, a query returns either "**possibly in set**" or "**definitely not in set**"

False Positive = "**possibly in set**" or "**definitely not in set**"

False Negative = "**possibly not in set**" or "**definitely in set**"

Overview

x : An element

S : A set of elements

Input: x, S

Output:

-TRUE if x in S

-FALSE if x not in S

Bloom Filter cont'd



Suppose you are creating an account on Facebook, you want to enter a cool username, you entered it and got a message, “Username is already taken”. You added your birth date along username, still no luck. Now you have added your university roll number also, still got “Username is already taken”. It’s really frustrating, isn’t it?

But have you ever thought how quickly Facebook check availability of username by searching millions of username registered with it. There are many ways to do this job –

Linear search : Bad idea!

Binary Search : There must be something better!!

Bloom Filter is a data structure that can do this job.

Bloom Filter cont'd



For understanding bloom filters, one must know hashing. A hash function takes input and outputs a unique identifier of fixed length which is used for identification of input.

Properties of Bloom Filter

- ❑ Unlike a standard hash table, a Bloom filter of a fixed size can represent a set with an arbitrarily large number of elements.
- ❑ Adding an element never fails. However, the false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
- ❑ Bloom filters never generate false negative result, i.e., telling you that a username doesn't exist when it actually exists.
- ❑ Deleting elements from filter is not possible because, if we delete a single element by clearing bits at indices generated by k hash functions, it might cause deletion of few other elements.

Working of Bloom Filter



An empty bloom filter is a bit array of n bits, all set to zero, like below:

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

We need k number of hash functions to calculate the hashes for a given input. When we want to add an item in the filter, the bits at k indices $h_1(x)$, $h_2(x)$, ... $h_k(x)$ are set, where indices are calculated using hash functions.

Example – Suppose we want to enter “geeks” in the filter, we are using 3 hash functions and a bit array of length 10, all set to 0 initially. First we’ll calculate the hashes as following :

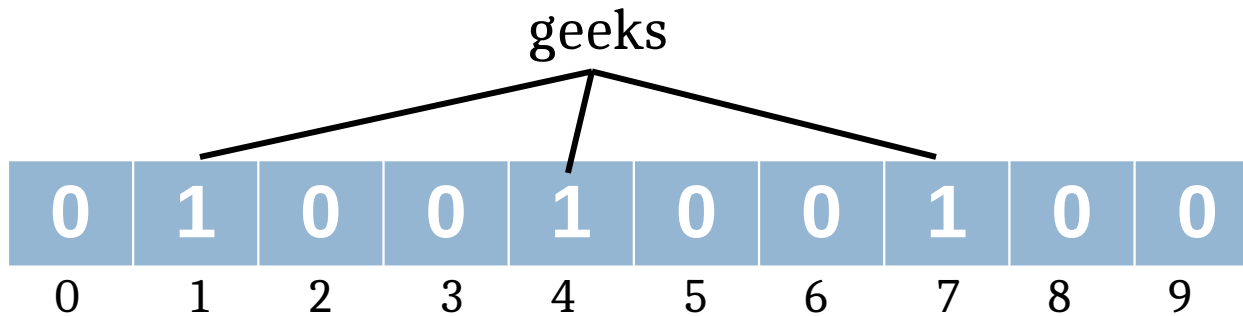
$h_1(\text{“geeks”}) \% 10 = 1$, $h_2(\text{“geeks”}) \% 10 = 4$, and $h_3(\text{“geeks”}) \% 10 = 7$

Note: *These outputs are random for explanation only.*

Working of Bloom Filter cont'd



Now we will set the bits at indices 1, 4 and 7 to 1



Again we want to enter “nerd”, similarly we’ll calculate hashes

$$h_1(\text{“nerd”}) \% 10 = 3$$

$$h_2(\text{“nerd”}) \% 10 = 5$$

$$h_3(\text{“nerd”}) \% 10 = 4$$

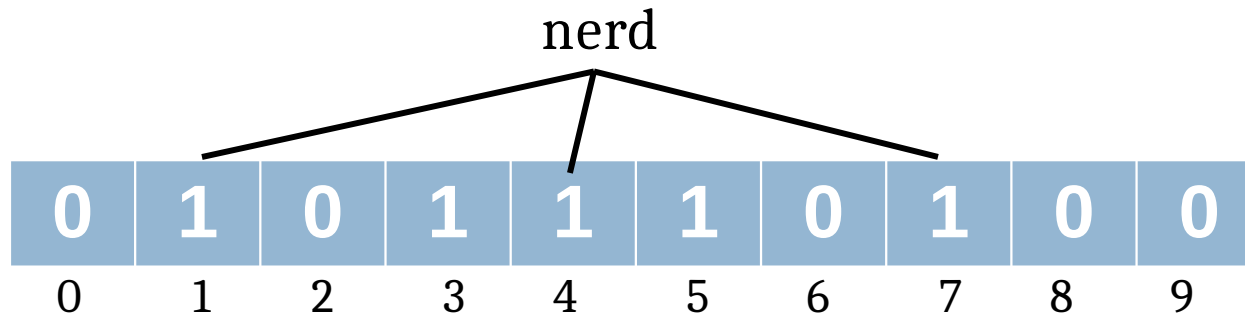
Note: *These outputs are random for explanation only.*

Set the bits at indices 3, 5 and 4 to 1

Working of Bloom Filter cont'd



Now we will set the bits at indices 3, 5 and 4 to 1



Now if we want to check “geeks” is present in filter or not. We’ll do the same process but this time in reverse order. We calculate respective hashes using h_1 , h_2 and h_3 and check if all these indices are set to 1 in the bit array. If all the bits are set then we can say that “geeks” is probably present. If any of the bit at these indices are 0 then “geeks” is definitely not present.

False Positive in Bloom Filters



The question is why we said “probably present”, why this uncertainty. Let’s understand this with an example. Suppose we want to check whether “cat” is present or not. We’ll calculate hashes using h_1 , h_2 and h_3

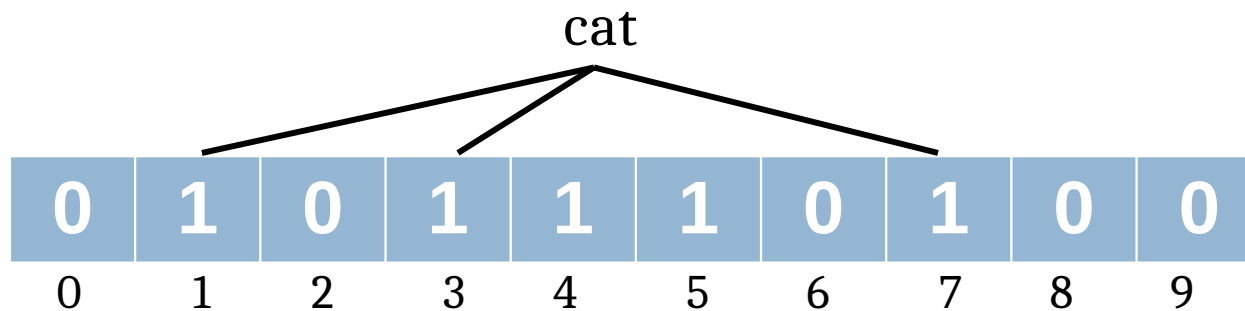
$$h_1(\text{“cat”}) \% 10 = 1$$

$$h_2(\text{“cat”}) \% 10 = 3$$

$$h_3(\text{“cat”}) \% 10 = 7$$

If we check the bit array, bits at these indices are set to 1 but we know that “cat” was never added to the filter. Bit at index 1 and 7 was set when we added “geeks” and bit 3 was set we added “nerd”.

False Positive in Bloom Filters cont'd



So, because bits at calculated indices are already set by some other item, bloom filter erroneously claim that “cat” is present and generating a false positive result. Depending on the application, it could be huge downside or relatively okay.

We can control the probability of getting a false positive by controlling the size of the Bloom filter. More space means fewer false positives. If we want to decrease probability of false positive result, we have to use more number of hash functions and larger bit array. This would add latency in addition of item and checking membership.



Bloom Filter Algorithm

Insertion

Data: e is the element to insert into the Bloom filter.

`insert(e)`

`begin`

`/* Loop all hash functions k */`

`for $j : 1 \dots k$ do`

`$m \leftarrow h_j(e)$ //apply the hash function on e`

`$B_m \leftarrow bf[m]$ //retrive val at m th pos from Bloom filter bf`

`if $B_m == 0$ then`

`/* Bloom filter had zero bit at index m */`

`$B_m \leftarrow 1$;`

`end if`

`end for`

`end`

Lookup

Data: x is the element for which membership is tested.

`bool isMember(x) /* returns true or false to the membership test */`

`begin`

`$t \leftarrow 1$`

`$j \leftarrow 1$`

`while $t == 1$ and $j \leq k$ do`

`$m \leftarrow h_j(x)$`

`$B_m \leftarrow bf[m]$`

`if $B_m == 0$ then`

`$t \leftarrow 0$`

`end`

`$j \leftarrow j + 1$;`

`end while`

`return (bool) t`

`end`

Bloom Filter Performance



A Bloom filter requires space $O(n)$ and can answer membership queries in $O(1)$ time where n is number item inserted in the filter. Although the asymptotic space complexity of a Bloom filter is the same as a hash map, $O(n)$, a Bloom filter is more space efficient.

Class



Exercise

□ A empty bloom filter is of size 11 with 4 hash functions namely

□ $h_1(x) = (3x + 3) \bmod 6$

□ $h_2(x) = (2x + 9) \bmod 2$

□ $h_3(x) = (3x + 7) \bmod 8$

□ $h_4(x) = (2x + 3) \bmod 5$

Illustrate bloom filter insertion with 7 and then 8.

Perform bloom filter lookup/membership test with 10 and 48

False Positive in Bloom Filters cont'd



0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12

$K=4$ (# of Hash Function)

INSERT (x_1), $x_1=7$

INSERT (x_2), $x_2 = 8$

Note: *These outputs are random for explanation only.*

$$h_1(x_1) = 0$$

$$h_1(x_2) = 3$$

$$h_2(x_1) = 1$$

$$h_2(x_2) = 1$$

$$h_3(x_1) = 4$$

$$h_3(x_2) = 7$$

$$h_4(x_1) = 2$$

$$h_4(x_2) = 4$$

State of Hashtable post to the insertion of x_1 and x_2



1	1	1	1	1	0	0	1	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11

False Positive in Bloom Filters cont'd



1	1	1	1	1	0	0	1	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11

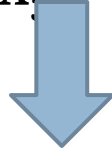
LOOKUP(x_3), $x_3 = 10$

$$h_1(x_3) = 3$$

$$h_2(x_3) = 1$$

$$h_3(x_3) = 5$$

$$h_4(x_3) = 3$$



**x_3 doesn't
exist**

LOOKUP(x_4), $x_4 = 48$

$$h_1(x_4) = 3$$

$$h_2(x_4) = 1$$

$$h_3(x_4) = 7$$

$$h_4(x_4) = 4$$



**x_4 - Case of
FALSE POSITIVE**

Optimum number of hash functions



The number of hash functions k must be a positive integer. If n is size of bit array and m is number of elements to be inserted, then k can be calculated as :

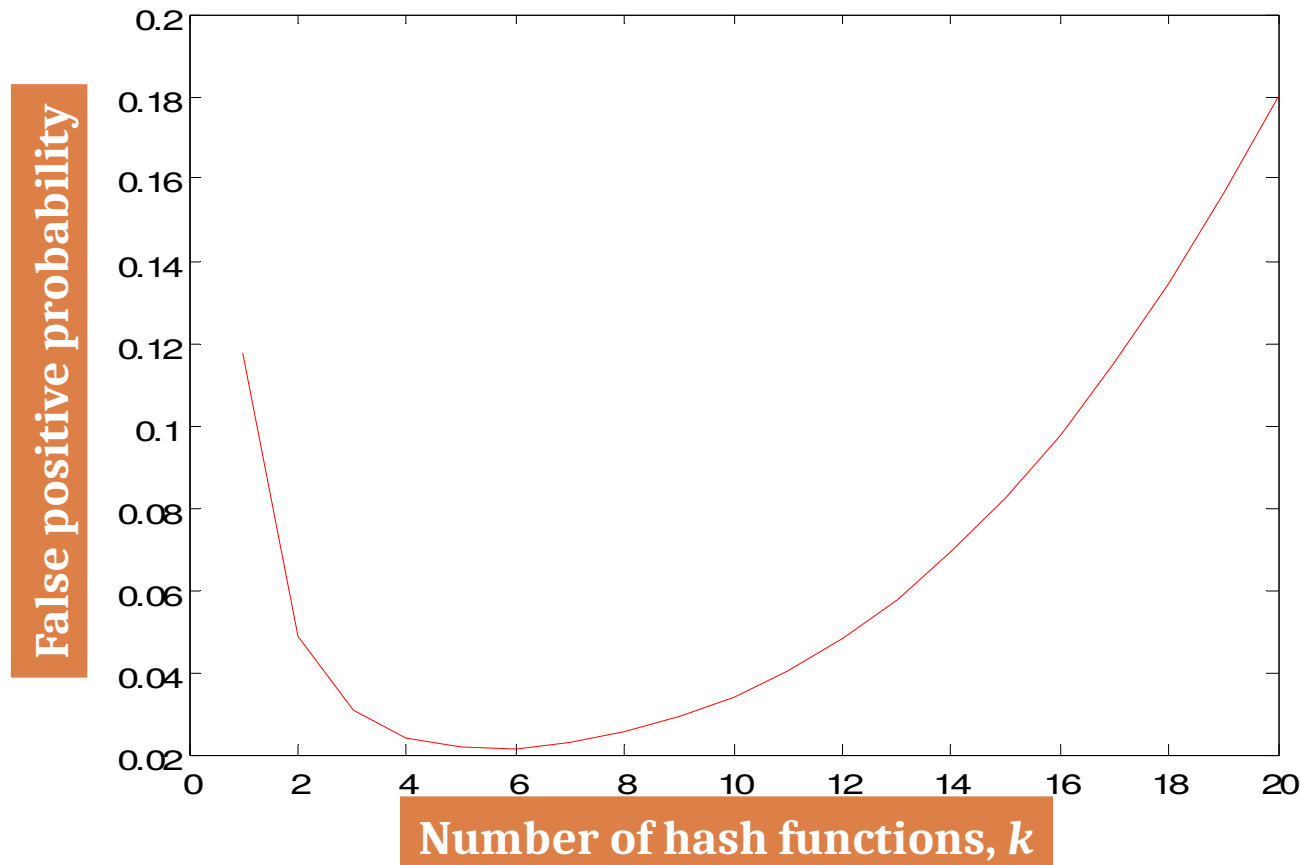
$$k = \frac{n}{m} \ln(2)$$



*Class
Exercise*

Find the optimal number of hash functions for 10 bit length bloom filter having 3 numbers of input elements.

What happens to increasing k ?



Calculating probability of False Positives



- ❑ Probability that a slot is hashed = $1/n$
- ❑ Probability that a slot is not hashed = $1 - (1/n)$
- ❑ Probability that a slot is not hashed after insertion of an element for all the k hash function is:

$$\left(1 - \frac{1}{n}\right)^k$$

- ❑ Probability that a slot is not set to 1 after insertion of m element is:

$$\left(1 - \frac{1}{n}\right)^{km}$$

- ❑ Probability that a slot is set to 1 after insertion of m element is:

$$1 - \left(1 - \frac{1}{n}\right)^{km}$$

Calculating probability of False Positives cont'd



Let n be the size of bit array, k be the number of hash functions and m be the number of expected elements to be inserted in the filter, then the probability of false positive p can be calculated as:

$$\left(1 - \left(\frac{1}{e}\right)^{\frac{km}{n}}\right)^k$$

Class Exercise

Calculate the probability of False Positives with table size 10 and no. of items to be inserted are 3.

Counting distinct elements in a stream – Approach 1



(1, 2, 2, 1, 3, 1, 5, 1, 3, 3, 3, 2, 2)
Number of distinct elements = 4

How to calculate?

1. Initialize the hashtable (large binary array) of size n with all zeros.
2. Choose the hash function $h_i : i \in \{1, \dots, k\}$
3. For each flow label $f \in \{1, \dots, m\}$, compute $h(f)$ and mark that position in the hashtable with 1
4. Count the number of positions in the hashtable with 1 and call it c .
5. The number of distinct items is $m * \ln (m / (m-c))$

Class Exercise



Counting distinct elements in a data stream of elements {1, 2, 2, 1, 3, 1, 5, 1, 3, 3, 3, 2, 2} with the hash function $h(x) = (5x+1) \bmod 6$ of size 11.

Counting distinct elements in a stream using Flajolet-Martin algorithm – Approach 2



Count the distinct elements in a data stream of elements {6,8,4,6,3,4} with hash function $h(x) = (5x+1) \bmod 6$ of size 11.

How to calculate?

Step 1:

Apply Hash function(s) to the data stream and compute the slots.

$h(6)=1$, $h(8)=5$, $h(4)=3$, $h(6)=1$, $h(3)=4$, $h(4)=3$.

The slot numbers obtained are: {1,5,3,1,4,3}

Step 2: Convert the numbers to binary

$h(6)=1=001$, $h(8)=5=101$, $h(4)=3=001$, $h(6)=1=001$, $h(3)=4=100$,
 $h(4)=3=011$

Step 3: Calculate the maximum trailing zeros

$TZ = \{0, 0, 0, 0, 2, 0\}$ /* TZ stands for Trailing Zeros */

$R = \text{MAX}(TZ) = \text{MAX}(0, 0, 0, 0, 2, 0) = 2$

Step 4: Estimate the distinct elements with the formula 2^R

Number of distinct elements $= 2^R = 2^2 = 4$

Bloom Filter Use Cases



- ❑ Bitcoin uses Bloom filters to speed up wallet synchronization and also to improve Bitcoin wallet security
- ❑ Google Chrome uses the Bloom filter to identify malicious URLs - it keeps a local Bloom filter as the first check for Spam URL
- ❑ Google BigTable and Apache Cassandra use Bloom filters to reduce disk lookups for non-existent rows or columns
- ❑ The Squid Web Proxy Cache uses Bloom filters for cache digests - proxies periodically exchange Bloom filters for avoiding ICP messages
- ❑ Genomics community uses Bloom filter for classification of DNA sequences and efficient counting of k-mers in DNA sequences
- ❑ Used for preventing weak password choices using a dictionary of easily guessable passwords as bloom filter
- ❑ Used to implement spell checker using a predefined dictionary with large number of words

Other Types of Bloom Filter



- ❑ **Compressed Bloom Filter** - Using a larger but sparser Bloom Filter can yield the same false positive rate with a smaller number of transmitted bits.
- ❑ **Scalable Bloom Filter** - Scalable Bloom Filters consist of two or more Standard Bloom Filters, allowing arbitrary growth of the set being represented.
- ❑ **Generalized Bloom Filter** - Generalized Bloom Filter uses hash functions that can set as well as reset bits.
- ❑ **Stable Bloom Filter** - This variant of Bloom Filter is particularly useful in data streaming applications.

Comparison of DWH, Hadoop and Stream Computing



Charactristics	Data warehouse	Hadoop	Stream computing
Type of Data Stored	Structured	Structured & Unstructured	No storage
Storage purpose	Reporting & dash board	Long running computations	Real time analytics
Age of data	Old	Past	Current/new data
Size of data	Terra/Peta bytes	Giga Bytes	Kilo Bytes
Speed of processing	Peta bytes /day	Kbps	Mbps
Implementation cost	High	Medium	Low
Volume	High	High	Low
Velocity	Nil	Nil	High
Variety	Nil	High	High

**THANK
YOU!**

Practice Questions



1. Develop Flajolet-Martin algorithm and using it, count the distinct elements in a data stream of elements {6, 8, 4, 6, 3, 4, 7, 6, 9} with the hash function $h(x) = (5x+1) \bmod 6$ of size 11.
2. A empty bloom filter is of size 11 with 4 hash functions namely:
 $h_1(x) = (3x+ 3) \bmod 6$
 $h_2(x) = (2x+ 9) \bmod 2$
 $h_3(x) = (3x+ 7) \bmod 8$
 $h_4(x) = (2x+ 3) \bmod 5$
Illustrate bloom filter insertion with 17, 81 and 37.
Perform bloom filter lookup/membership test with 10 and 81.

Practice Questions



3. A empty bloom filter is of size 11 with 2 hash functions namely:
 $h_1(x) = (3x + 3) \bmod 18$
 $h_2(x) = (2x + 9) \bmod 22$
Illustrate bloom filter insertion with 7, 8 and 77.
Perform bloom filter lookup/membership test with 7, 10 and 88.
4. Develop an algorithm to i) insert an item, and to ii) test the membership (or lookup) in Bloom Filter. Draw a step-by-step process in the insertion of element 25, and then 40 into the Bloom Filter of size 10. Then, draw a step-by-step process for lookup/membership test with the elements 10 and 48. The hash functions are: $h_1(x) = (3x + 41) \bmod 6$, and $h_2(x) = (7x + 5)$. Identify whether any lookup element (i.e. either 10 or 48) is resulting into the case of FALSE POSITIVE?

Practice Questions



5. Let, Facebook wants to count “How many unique users visited the Facebook this month?” What will be the stream elements in this case?

GOOD LUCK
FOR YOUR
EXAM^{AND}
DO THE BEST