Sampling from a Data Stream.....

Data Sampling?

Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points in order to identify patterns and trends in the larger data set being examined.

Stream Queries

- There are two ways that queries get asked about streams.
- **Ad-hoc Queries:** Normal queries asked one time about streams.
  - ***Example: What is the maximum value seen so far in stream S?***
- **Standing Queries:** These queries are, in a sense, permanently executing, and produce outputs at appropriate times. Queries that are in principle, asked about the stream at all time.
  - **Example: Report each maximum value ever seen in stream S.**

Problems on Data Streams

- Types of queries one wants on answer on a 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
  - **Estimating moments**
    - Estimate avg./std. dev. of last *k* elements
  - **Finding frequent elements**

Reasons for Sampling

Reasons for Sampling

- Sampling can save money.
- Sampling can save time.
- For given resources, sampling can broaden the scope of the data set.
- Because the research process is sometimes destructive, the sample can save product.
- If accessing the population is impossible; sampling is the only option.

AT A GLANCE

Sampling Techniques in Big Data Stream

Following are the techniques:

1. Fixed Proportion Sampling
2. Fixed Size Sampling
3. Biased Reservoir Sampling
4. Concise Sampling

Yash Paddalwar

Sampling Techniques in Big Data Stream

1. Fixed Proportion Sampling

- Samples fixed proportion of data
- Used when you are aware of the length of data
- Ensures representative sample
- Useful for large volumes
- Less biased than fixed sized sampling
- May lead to under/over representation

Sampling Techniques in Big Data Stream

1. Fixed Proportion Sampling

Example

A social media platform wants to analyze the sentiments of its users towards a topic. They receive millions of tweets per day and use fixed proportion sampling to select a representative sample. They randomly select 1% of the tweets received each hour, ensuring a representative sample for statistical analysis of user sentiments towards the topic.

Sampling Techniques in Big Data Stream

2. Fixed Size Sampling

- Samples fixed number of data points.
- Does not guarantee representative sample.
- Useful for reducing data volume.
- Can be biased if data is not randomly distributed

Sampling Techniques in Big Data Stream

2. Fixed Size Sampling

Example

Suppose we have a data stream of customer orders for an online store, with 10,000 orders coming in every hour. Using fixed size sampling, we randomly select 1,000 orders from each hour's data stream for analysis, thus reducing the total number of data points to process from 10,000 to 1,000 per hour.

Sampling Techniques in Big Data Stream

3. Biased Reservoir Sampling

- Used in streams to select a subset of the data in a way that is not uniformly random.
- Can lead to a biased sample that may not be representative of the full dataset.
- The selection of elements is based on a predetermined probability distribution that may be weighted towards certain elements or groups of elements.
- The probability distribution used for biased reservoir sampling may be based on various factors, such as the frequency of occurrence of certain types of data or the importance of certain data points.
- Used when there are constraints on the resources available for sampling, such as limited memory or computational power.
- It is important to carefully consider the potential biases introduced by this sampling technique and adjust the analysis accordingly.

Sampling Techniques in Big Data Stream

3. Biased Reservoir Sampling

Example

Suppose we have a data stream of product ratings, and we want to select a sample of ratings to estimate the average rating of a product. However, we know that some users tend to give higher ratings than others. Using biased reservoir sampling, we can assign a higher probability of selection to ratings from users who tend to give more accurate ratings. This way, our sample is more likely to represent the true average rating of the product.

Sampling Techniques in Big Data Stream

4. Concise Sampling

- Goal is to maintain a small reservoir of a fixed size while still achieving representative sampling of the data stream
- Number of samples that can be stored in memory at a given time is limited, which can be a challenge when dealing with large data streams.
- Size of the sample may need to be adjusted based on the amount of memory available to store the data.
- Instead of selecting samples randomly, the sampling algorithm may prioritize choosing samples with unique or representative values of a particular attribute in the data stream

Sampling Techniques in Big Data Stream

4. Concise Sampling

Example

- A bank wants to analyze customer spending habits from a stream of transactions.
- They use concise sampling to choose distinct customer IDs as their attribute.
- The size of the reservoir is limited to 1000 customers.
- They adjust the sample size based on available memory.
- This allows for efficient analysis while maintaining accuracy.

Sampling from a Data Stream

- Since we can not store the entire stream, one obvious approach is to store a **sample**
- **Two different problems:**
  - **(1)** Sample a **fixed proportion** of elements in the stream (say 1 in 10)
  - **(2)** Maintain a **random sample of fixed size** over a potentially infinite stream
    - At any "time" ***k*** we would like a random sample of ***s*** elements
      - **What is the property of the sample we want to maintain?**

For all time steps ***k***, each of ***k*** elements seen so far has equal prob. of being sampled.

Sampling a Fixed Proportion

- **Problem 1: Sampling fixed proportion**
- **Scenario:** Search engine query stream
  - **Stream of tuples:** (user, query, time)
  - **Answer questions such as:** How often did a user run the same query on two different days?
- **Naïve solution:**
  - Generate a random integer in [0..9] for each query
  - Store the query if the integer is 0, otherwise discard
  - If we do so, each user has, on average, 1/10th of their queries stored.

Problem with Naive Approach

- Simple question: **What fraction of queries by an average user are duplicates?**
- Suppose each user issues x queries once and d queries twice (total of $x+2d$ queries)

Correct answer: $d/(x+d)$

  - If we have a 1/10th sample (10% of the queries), of queries, we shall see in the sample for that user an expected $x/10$ of the search queries issued once and $2d/10$ of the duplicate queries at least once.
  - But only $d/100$ pairs of duplicates
    - $d/100 = 1/10 * 1/10 * d$
  - Of d "duplicates" $18d/100$ appear once
    - $18d/100 = ((1/10*9/10)+(9/10*1/10))*d$
  - **So the sample-based answer is: $d/(10x + 19d)$**

$$\frac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}} = \frac{d}{10x+19d}$$

Sampling a fixed-size sample

- **Problem 2: Fixed-size sample**
- **Suppose we need to maintain a random sample *S* of size exactly *s* tuples**
  - E.g., main memory size constraint
- **Why?** Don't know length of stream in advance
- **Suppose at time *n* we have seen *n* items**
  - **Each item is in the sample *S* with equal prob. *s/n***

How to think about the problem: say s = 2 **Stream:** a x c y z k c d e g…
At **n= 5,** each of the first 5 tuples is included in the sample **S** with equal prob. At **n= 7,** each of the first 7 tuples is included in the sample **S** with equal prob.

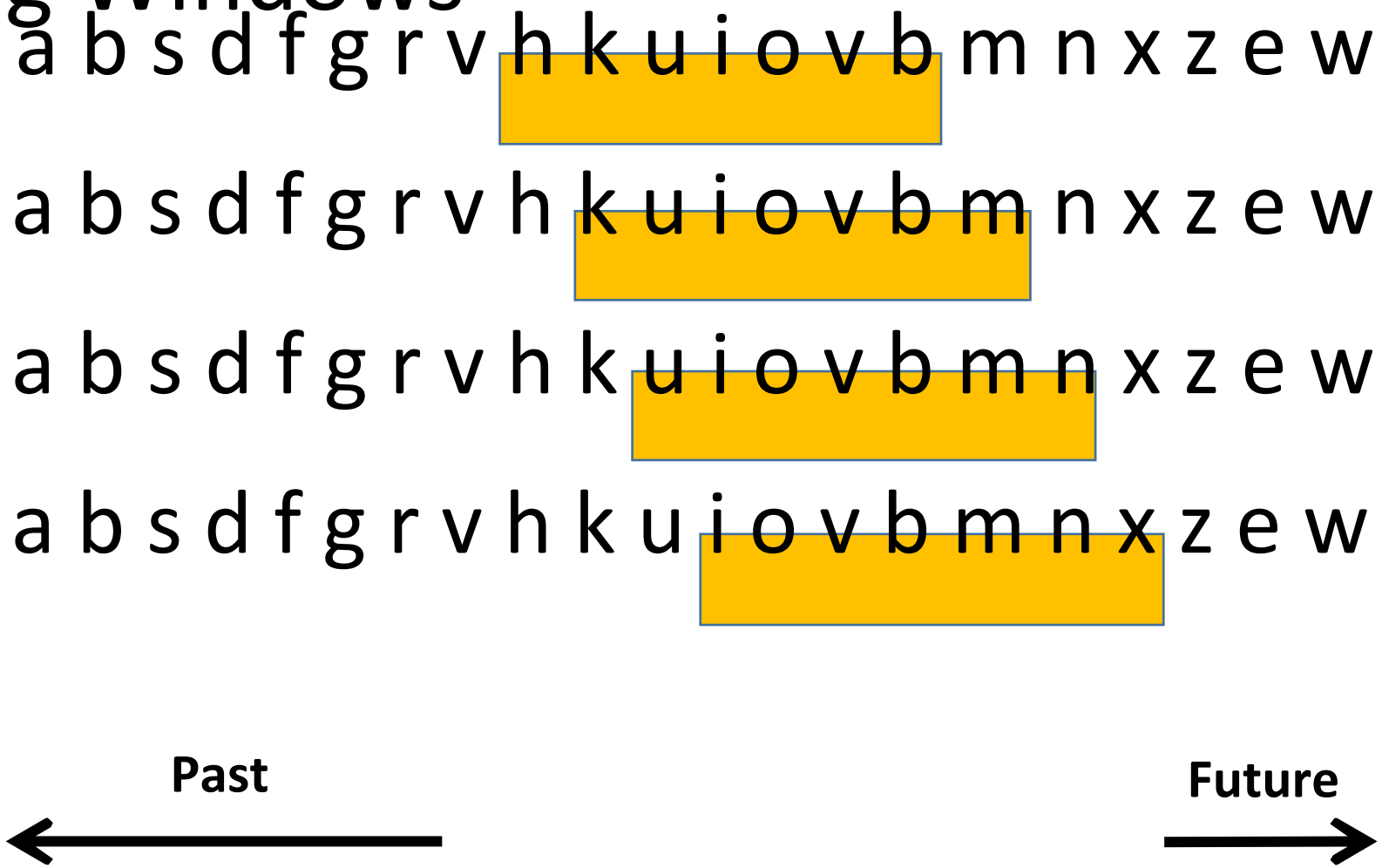
Impractical solution would be to store all the *n* tuples seen so far and out of them pick *s* at random

Sliding Windows

- A useful model of stream processing is that queries ar about a windowof length N - the N most recent elements received.
  - *Elements received within a time interval T.*

- ***Interesting case:*** N is so large it cannot be stored in main memory.
  - *Ther are so many streams that windows for all do not fit in main memory.*

Sliding Windows



Types of Sliding Window

- **Sequence-based**

  The window of size k moving over the k mist recently arrived data. Example being chain-sample algorithm

- **Time-stamp based**

  The window of duration t consist of elements whose arrival timestamp is within a time interval t of the current time. Example being Priority Sample for Sliding Window

Solution: Fixed Size Sample

- **Algorithm (Reservoir Sampling)**
  - Assumes that ***n***, the stream length, is a constant. In practice, ***n*** grows with time.
  - Store all the first ***s*** elements of the stream to ***S***
  - Suppose we have seen ***n-1*** elements, and now the n^{th} element arrives (***n*** > ***s***)
    - With probability ***s/n***, keep the n^{th} element, else discard it
    - If we picked the n^{th} element, then it replaces one of the ***s*** elements in the sample ***S***, picked uniformly at random
- **Claim:** This algorithm maintains a sample ***S*** with the desired property:
  - After ***n*** elements, the sample contains each element seen so far with probability ***s/n***

Proof: By Induction

- **We prove this by induction:**
  - Assume that after ***n*** elements, the sample contains each element seen so far with probability ***s/n***
  - We need to show that after seeing element ***n+1*** the sample maintains the property
    - When the ***(n+1)***st element arrives, pick that position with probability ***s/(n+1).***
    - ***If not picked, then the s variables keep their same positions.***
- **Base case:**
  - After we see **n=s** elements the sample **S** has the desired property
    - Each out of **n=s** elements is in the sample with probability ***s/s = 1***

Reservoir Sampling

Stream of ***n*** items (where ***n*** not known in advance)

$$i.e\ a_1, a_2, a_3, \ldots\ldots\ldots\ldots\ldots a_n$$

$$\forall_i \text{ The } \mathbf{Pr}(a_i \text{ is choosen}) = \mathbf{1/n}$$

Let, ***s*** be the memory location, and when i^{th} item arrives,

with the **Pr (1/i)** ,

$$s \leftarrow a_i \quad (\forall\ i^{th}\ items)$$

So, the $\mathbf{Pr}(s = a_i) = \mathbf{1/n}$

Reservoir Sampling (cont..)

The **Pr($s = a_i$) = 1/n,**

$$= \frac{1}{i}\left(1-\frac{1}{i+1}\right)\left(1-\frac{1}{i+2}\right)\cdots\left(1-\frac{1}{n}\right)$$

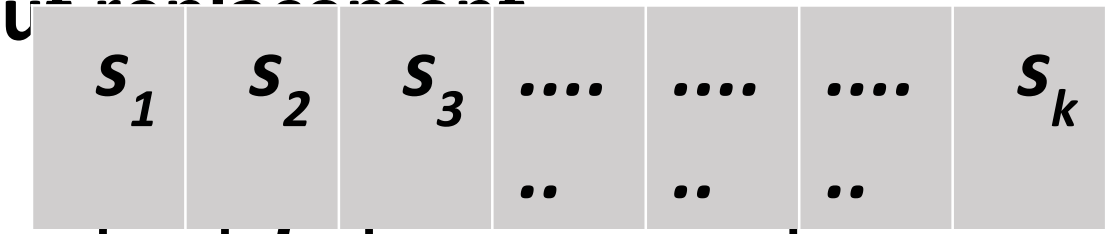
$$= \frac{1}{i}\left(\frac{i+1-1}{i+1}\right)\left(\frac{i+2-1}{i+2}\right)\cdots\left(\frac{n}{n}\right)$$

$$= \frac{1}{i}\left(\frac{i}{i+1}\right)\left(\frac{i+1}{i+2}\right)\cdots\left(\frac{n-1}{n}\right)$$

$$= \frac{1}{i}\left(\frac{i}{i+1}\right)\left(\frac{i+1}{i+2}\right)\cdots\left(\frac{n-1}{n}\right) = 1/n \quad (\forall i)$$

Reservoir Sampling (cont..)

- How to sample ***k*** items from the stream ?
  - **With replacement** (*Repeat reservoir sampling k times in parallel*)
    - Straight forward
  - **Without replacement**



- For the first ***k*** items, simply assign $s_i \leftarrow a_i$ **(1 ≤ *i* ≤ *k*)**

Reservoir Sampling (cont..)

- For the first k items, simply assign $s_i \leftarrow a_i$ **($1 \le i \le k$)**
  - For $i > k$
    - j = random number from $1 \ldots i$ **(Uniform and Random)**
  - if $j \le k$
    - $s_j = a_i$

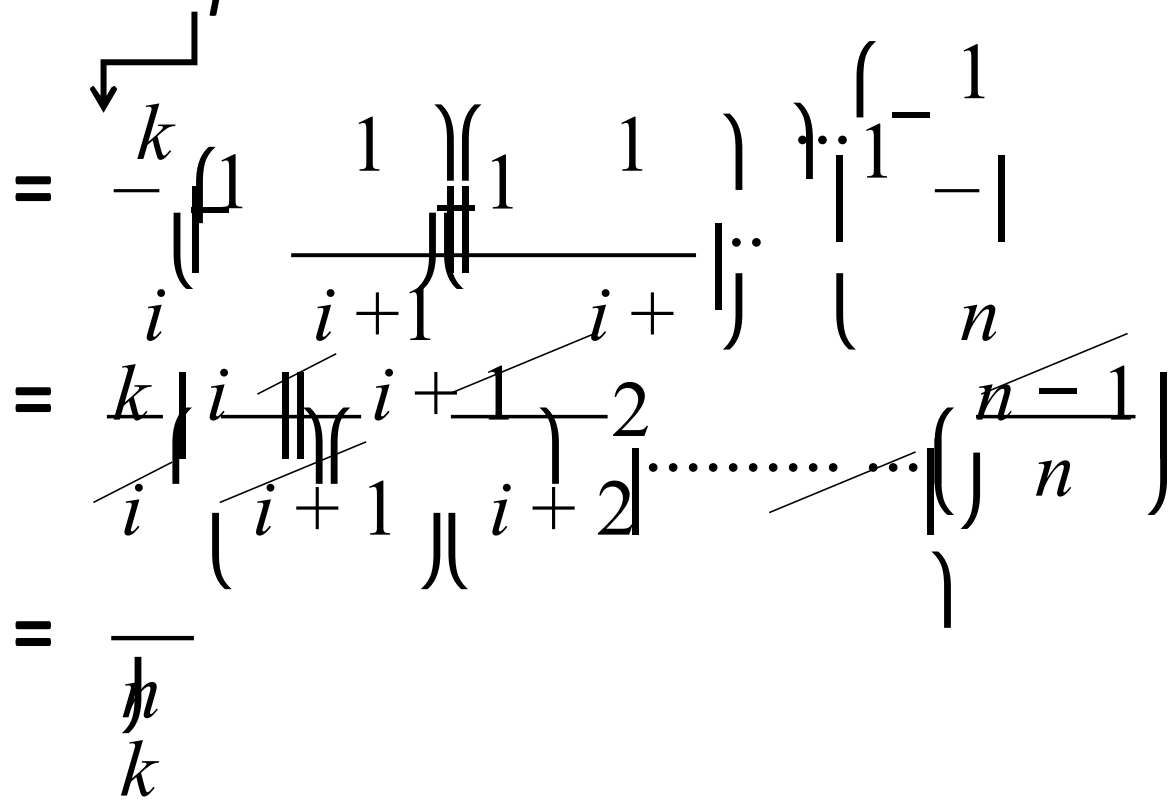
s_1	s_2	s_3	…	…sj.	….	s_k
			..	..	..	

Let, **S = {s_1, s_2, …….s_k} at the end of the stream**

Claim: Pr($a_i \in$ S) = k/n

Reservoir Sampling (cont..)

- **Claim: Pr($a_i \in S$) = k/n**



Example

Choose 3 numbers from [111, 222, 333, 444]. Make sure each number is selected with a probability of 3/4

First, choose [111, 222, 333] as the initial reservior Then choose 444 with a probability of 3/4

For 111, it stays with a probability of

P(444 is not selected) + P(444 is selected but it replaces 222 or 333)

= 1/4 + 3/4*2/3

= 3/4

The same case with 222 and 333

Now all the numbers have the probability of 3/4 to be picked

BloomsFilter Algorithm

What is Bloom Filter?

- A Bloom filter is a **space-efficient probabilistic** data structure that is used to test whether an element is a member of a set.
- For example, checking availability of username is set membership problem, where the set is the list of all registered username.
- The price we pay for efficiency is that it is probabilistic in nature that means, there might be some False Positive results.
- **False positive means**, it might tell that given username is already taken but actually it’s not.

Properties of Bloom Filters

- 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.

Bloom Filters

Whenever a list or set is used, and space is consideration, a Bloom filter should be considered. When using a Bloom filter, consider the potential effects of **false positives**."

- It is a randomized data structure that is used to represent a set.
- It answers membership queries
- It can give **FALSE POSITIVE** while answering membership queries (very less %).
- But **can't return FALSE NEGATIVE**
  - **POSSIBLY IN SET**
  - **DEFINITELY NOT IN SET**
- Space efficient

Bloom Filters

- Bloom filters are a natural variant of hashing proposed by **Burton Bloom in 1970** as a mechanism for supporting membership queries in sets.
- Applications:
- **Example: Email spam filtering**
  - We know 1 billion “good” email addresses
  - If an email comes from one of these, it is **NOT** spam

Filtering Stream Content

- To motivate the Bloom-filter idea, consider a web crawler.
- It keeps, centrally, a list of all the URL's it has found so far.
- It assigns these URL's to any of a number of parallel tasks; these tasks stream back the URL's they find in the links they discover on a page.
- It needs to filter out those URL's it has seen before.

Role of the Bloom Filter

- A Bloom filter placed on the stream of URL's will declare that certain URL's have been seen before.
- Others will be declared new, and will be added to the list of URL's that need to be crawled.
- Unfortunately, the Bloom filter can have false positives.
  - It can declare a URL has been seen before when it hasn't.
  - But if it says “never seen”, then it is truly new.

Operations that a Bloom Filter supports

- insert(x) : To insert an element in the Bloom Filter.

- lookup(x) : to check whether an element is already present in Bloom Filter with a positive false probability.

NOTE : We cannot delete an element in Bloom Filter.

How a Bloom Filter Works?

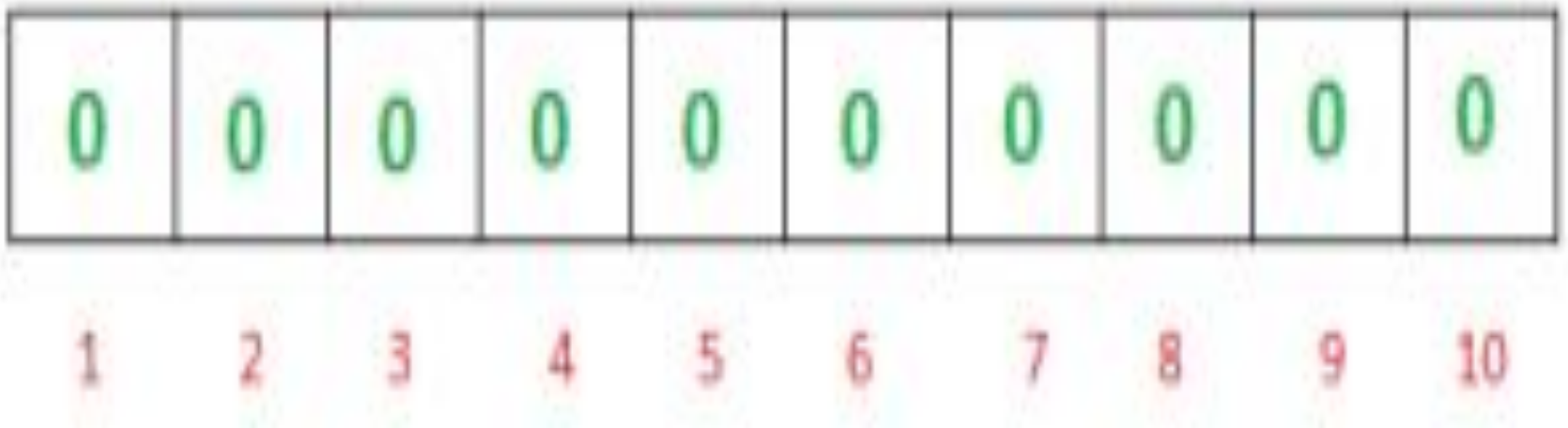
- A ***Bloom filter*** is an array of bits, together with a number of hash functions.
- The argument of each hash function is a stream element. and it returns a position in the array.
- Initially, all bits are 0.
- When input x arrives, we set to 1 the bits h(x). for each hash function h.

The Set memebrship task

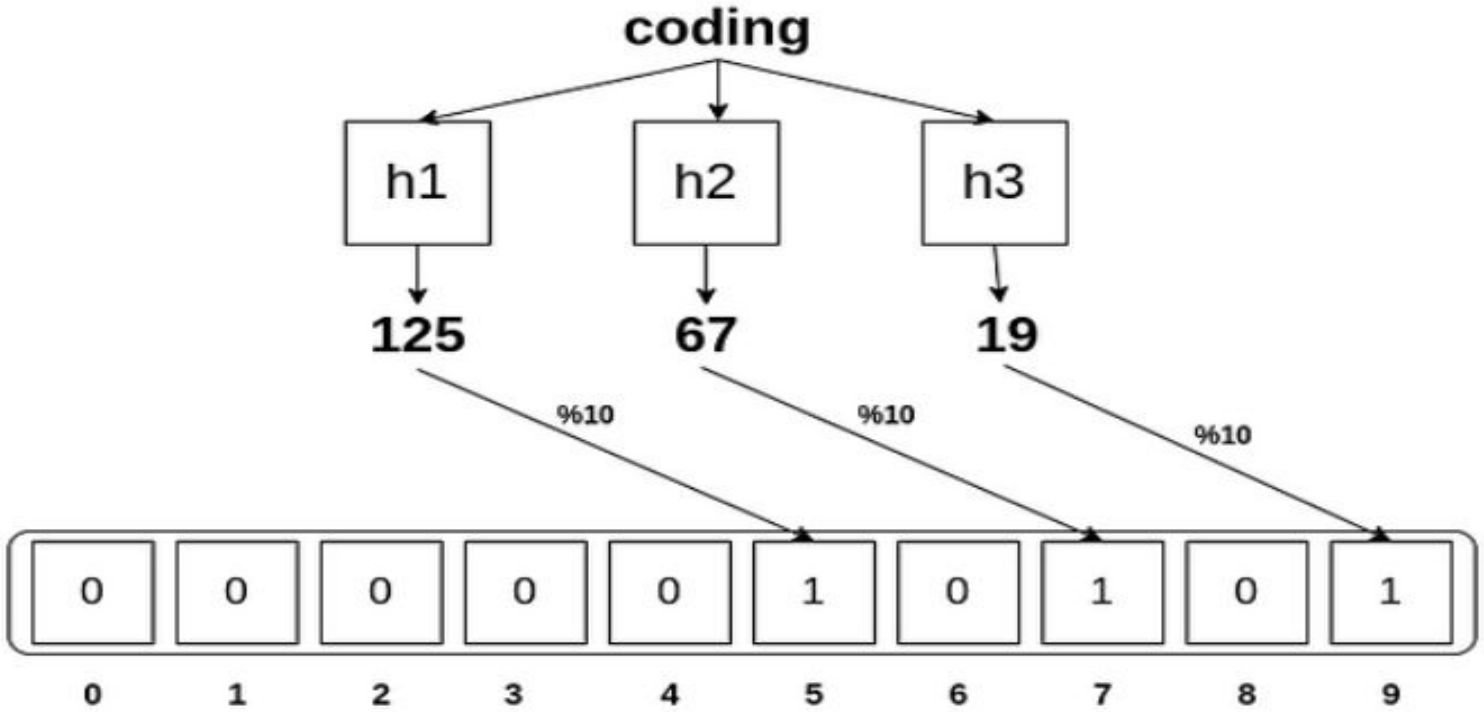
- **x :** An element
- **S:** A set of elements
- **Input:** x, S
- **Output**:
  - -TRUE if x in S
  - -FALSE if x not in S

Working of Bloom Filter

- A empty bloom filter is a **bit array** of **m** bits, all set to zero, like this –

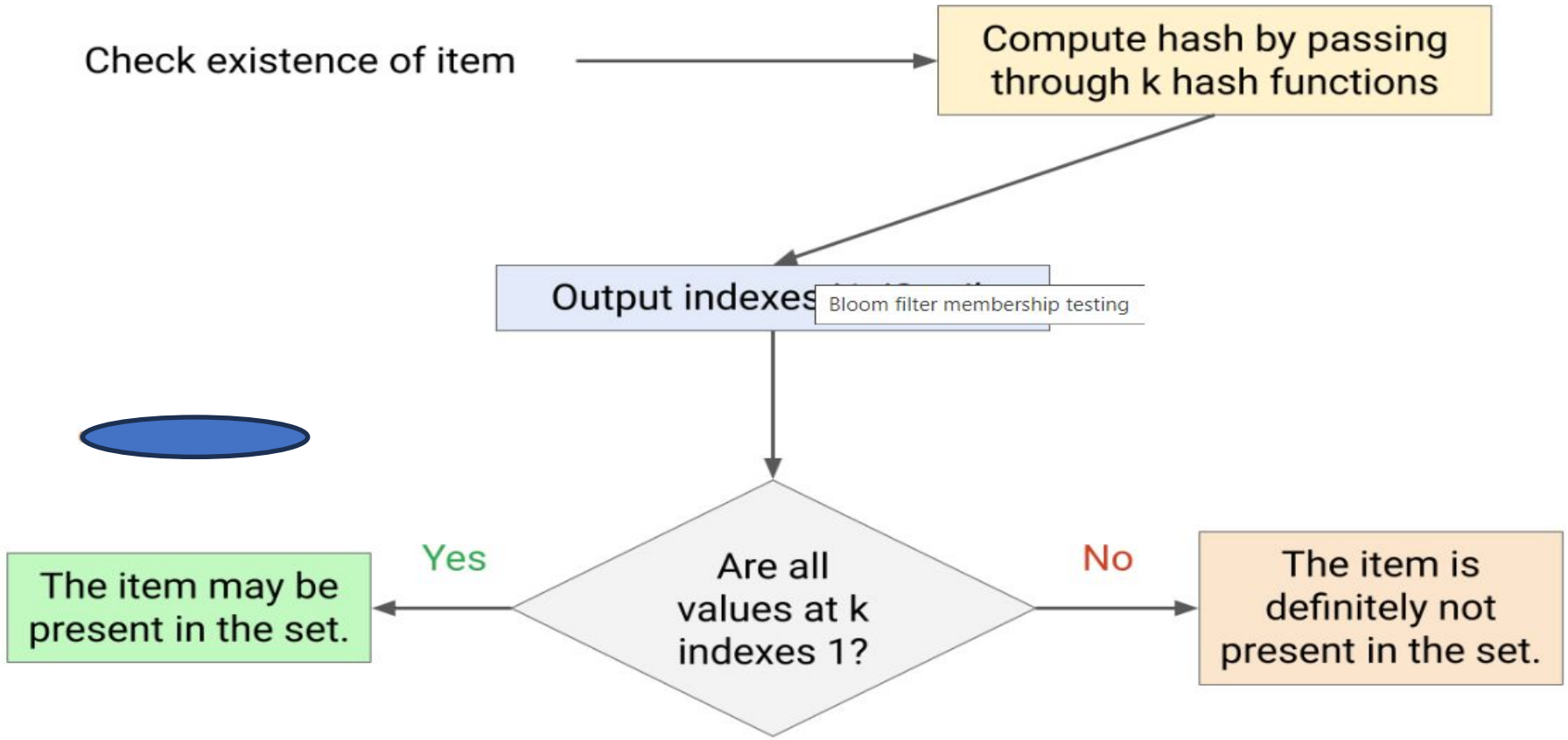


Inserting Items into Bloomfilter



Inserting in bloom filter

Testing membership of an item in Blooms Filter



Insert (x)

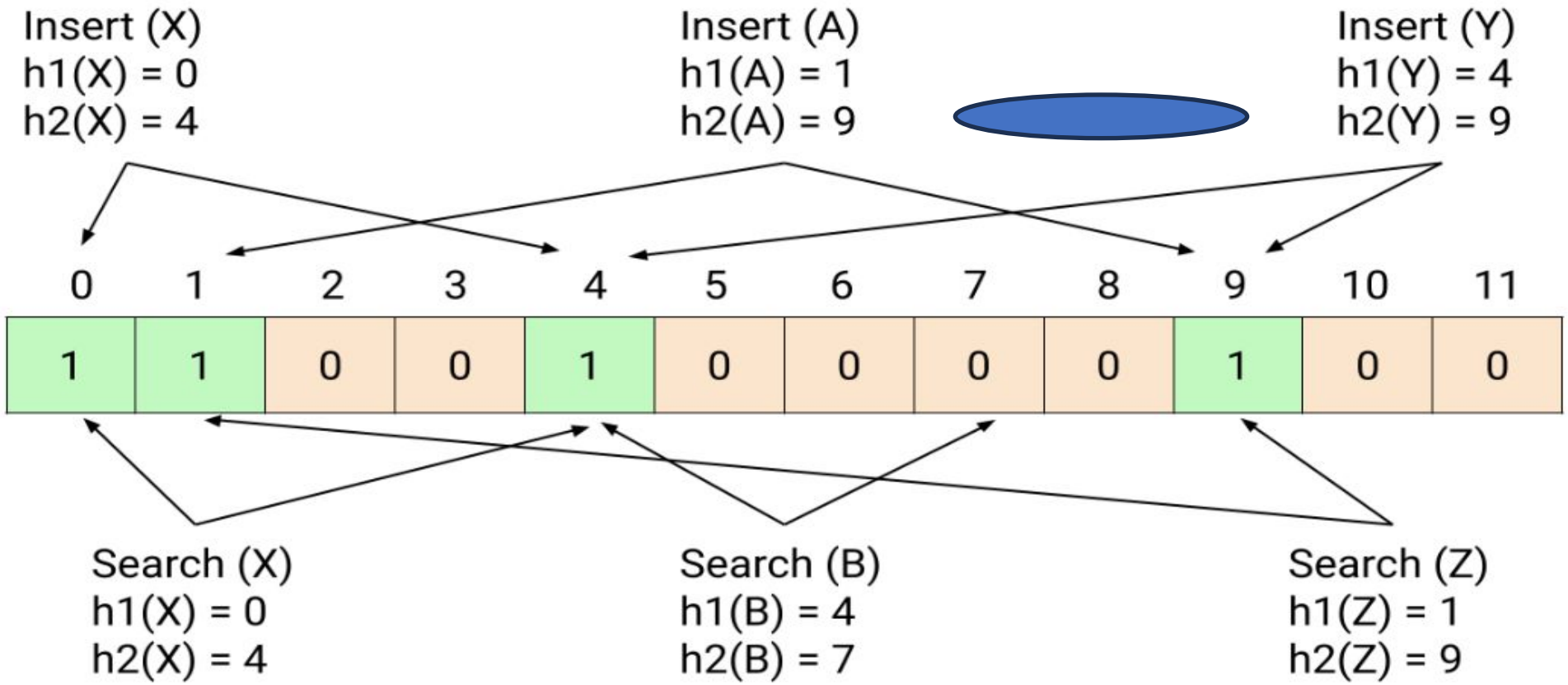
Find $h_1(x), h_2(x),.....,h_k(x),$ set all these bits in Bloom Filter to **1**

```
QUERY-Bloom Filter (y)
Find h1(y), h2(y),....hk(y)
IF (All h1(y), h2(y),....hk(y) ==1)
     RETURN(1)
ELSE
     RETURN(0);
```

We assume hash functions maps an element in bits 0,1,2,...(n-1)

m = number of elements inserted or present in the set

Bloom Filter Visualization With Example



True Positive
Value X passes through bloom filter and present in memory

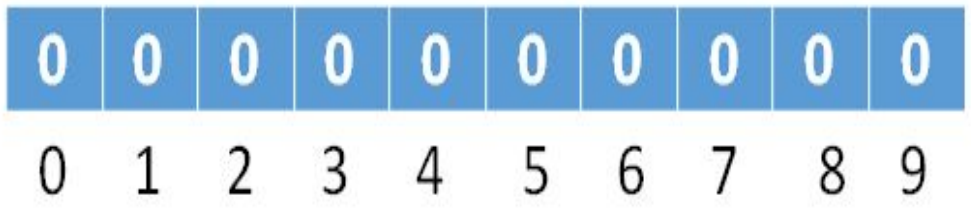
True Negative
Value B not passes through bloom filter and not present in memory

False Positive
Value Z passes through bloom filter but not present in memory

Probability of False positivity:

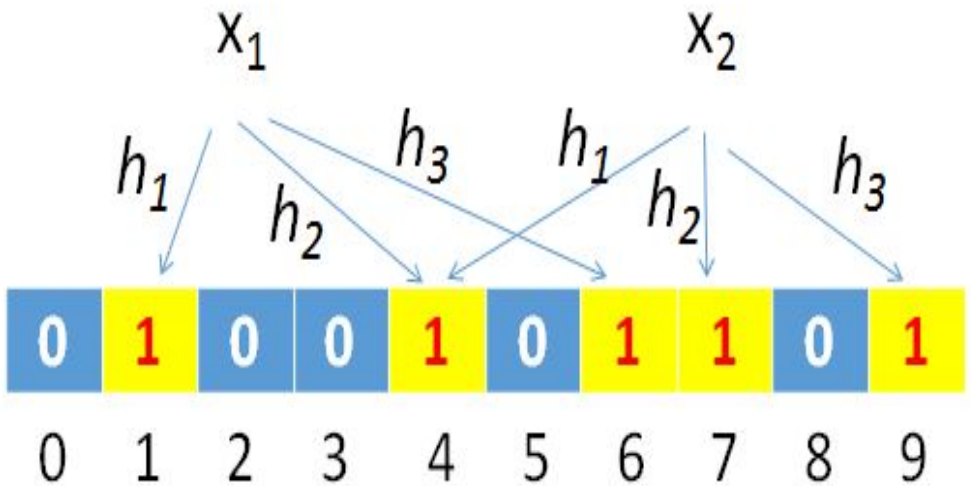
Let **m** be the size of bit array, k be the number of hash functions and **n** be the number of expected elements to be inserted in the filter, then the probability of false positive **p** can be calculated as:

- A Bloom filter consists of vectors of ***n*** boolean values, initially all set false, as well as ***k*** independent hash functions, $h_1, h_2, \ldots, h_k$, each with range **{0,1,..., n-1}**



Initial setup ***n***=10

- For each element **x** in **S**, the boolean values with positions $h_1(x)$, $h_2(x),\ldots,h_k(x)$, are set true.

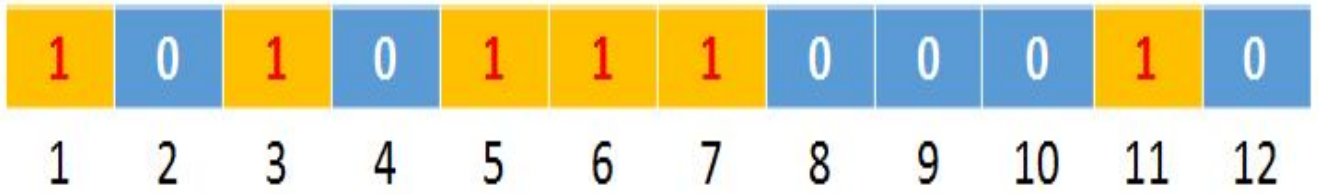


Installing two elements x1, x2

Error types

- **False Negative**: answering “not there” on an element that is in the set.
  - Never happens for Bloom Filters
- **False Positive**: answering “is there” on an element that is not in the set
  - We design the filter so that the probability of a false positive is very small.

Calculating the Probability of False Positives



12 bit Bloom filter

k=3 (hash function)

INSERT (x_1)
$h_1(x_1) = 3$
$h_2(x_1) = 5$
$h_3(x_1) = 11$

INSERT (x_2)
$h_1(x_2) = 1$
$h_2(x_2) = 7$
$h_3(x_2) = 6$

QUERY(x_3)
$h_1(x_3) = 3$
$h_2(x_3) = 11$
$h_3(x_3) = 7$

→ CASE of FALSE POSITIVE

QUERY(x_4)
$h_1(x_4) = 4$
$h_2(x_4) = 9$
$h_3(x_4) = 10$

→ x_4 is not PRESENT

Calculating the Probability of False Positives

- Probability that a cell is not hashed after insertion of an element for all the ***k*** hash function is:

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

(Probability for a cell is hashed = 1/n, for all the ***k*** hash function **$(1/n)^k$**)

- Pr (Cell is not set to 1 after insertion of ***m*** element is): $\left(1-\frac{1}{n}\right)^{km}$

- Pr (Cell is set to 1 after insertion of ***m*** element is): $1-\left(1-\frac{1}{n}\right)^{km}$

Calculating the Probability of False Positives

- Now every element are inserted into the bloom filter.
- So, **what is the probability of FALSE POSITIVE ?**
- **i.e the probability of false positive is whatever the k cells we have that elemnts should have to hash 1.**
- Thus, Pr (All k cells are set to 1) =

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

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

As per approximation, $\left(1-\frac{1}{n}\right)^{n} = \frac{1}{e} = e^{-1}, e = 2.71....$

Calculating the Probability of False Positives

- The False Positive probability is minimised by choosing

$$k=$$

Applications

- **Applications of Bloom filters**
- Medium uses bloom filters for recommending post to users by filtering post which have been seen by user.
- Quora implemented a shared bloom filter in the feed backend to filter out stories that people have seen before.
- The Google Chrome web browser used to use a Bloom filter to identify malicious URLs
- Google BigTable, Apache HBase and Apache Cassandra, and Postgresql use Bloom filters to reduce the disk lookups for non-existent rows or columns