Cloud Based Data Engineering

Approximation and Randomization for Streaming Data

Approximation and Randomization

Many things are hard to compute exactly over a stream

- Is the count of all items the same in two different streams?

Approximation: find an answer correct within some factor

”Find an answer that is within 10% of correct result”

Randomization: allow a small probability of failure

“Answer is correct, except with probability 1 in 10,000”

Approximation and Randomization

More generally,

Approximation: find an answer correct within a $(1\pm \varepsilon)$ factor approximation

Randomization: allow a small probability of failure / success probability $(1-\delta)$

Here we want to develop Approximation **and** Randomization algorithms,

called **(ε, δ)-approximations**.

Approximation

$$True\ answer \leq output \leq \alpha \cdot True\ answer$$

Goal: minimize α as much as we can

- Best is $\alpha = 1 + \epsilon$, for all $\epsilon > 0$

- We will require the result to hold with probability of $1 - \delta$.

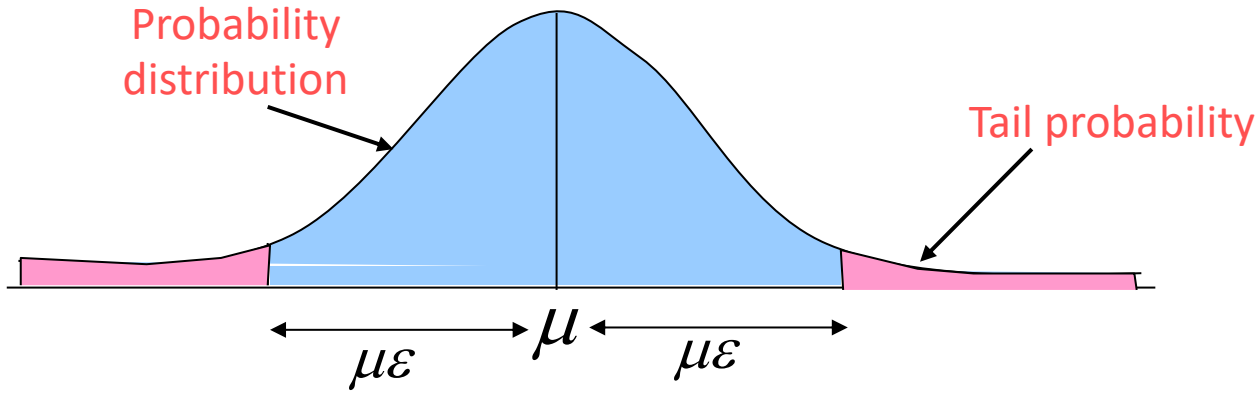
(if δ is very small then the probability is close to 1).

For example, if $\epsilon = \frac{1}{3}$ and $\delta = 0.01$ then we will have an error of more than $\frac{1}{3}$ only 1% of the time(in 99% we are good!!)

Basic Tools: Tail Inequalities

Let X be a random variable with expectation $E[X] = \mu$ and variance $Var[X] = \sigma^2$.

We will use two tail inequalities: Markov and Chebyshev.



Basic Tools: Tail Inequalities

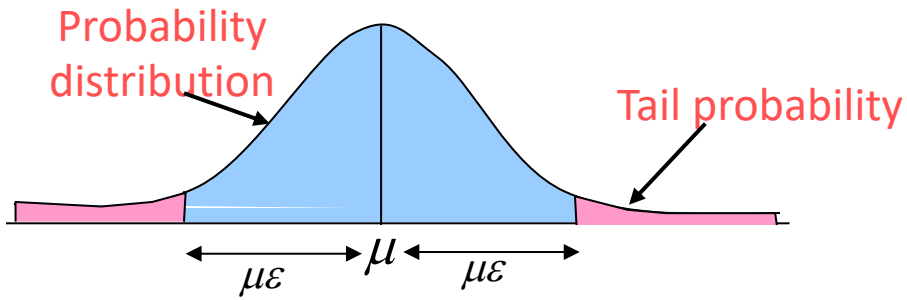
Markov Property

for any $\varepsilon > 0$, it holds $P[X > (1+\epsilon)\mu] \leq \frac{1}{1+\epsilon}$

For example,

In case of $\varepsilon = 0.1$, $\mu = 3.5$ then

$$P[X > 1.1 \cdot 3.5] \leq \frac{1}{1.1} \Rightarrow P[X > 3.85] = 0.90$$



Basic Tools: Tail Inequalities

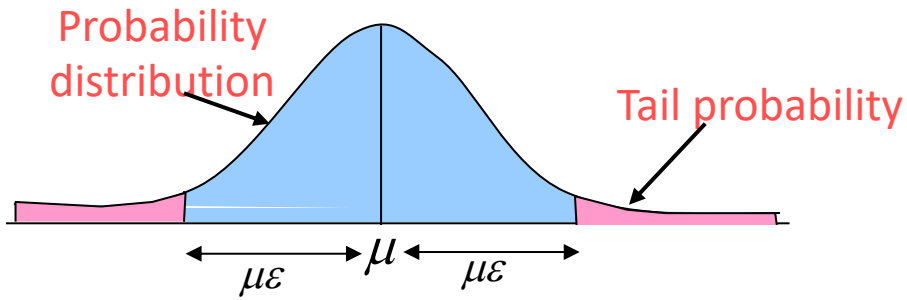
Chebyshev Property

for any $\varepsilon > 0$, it holds $P[|X - \mu| \geq \mu\epsilon] \leq \frac{\sigma^2}{\mu^2\epsilon^2}$

For example,

In case of $\varepsilon = 0.3, \mu = 3.5, \sigma^2 = 1$ then

$$P[|X - 3.5| \geq 0.3 \cdot 3.5] \leq \frac{1}{1.1025} = 0.907$$



Basic Stream Data Model

A stream is a sequence of data elements that comes in one by one.

Streaming - processing data elements as they arrive.

We are interested in cases where data stream is a massive sequence of data

– Network traffic

– Database transactions

Basic Stream Data Model

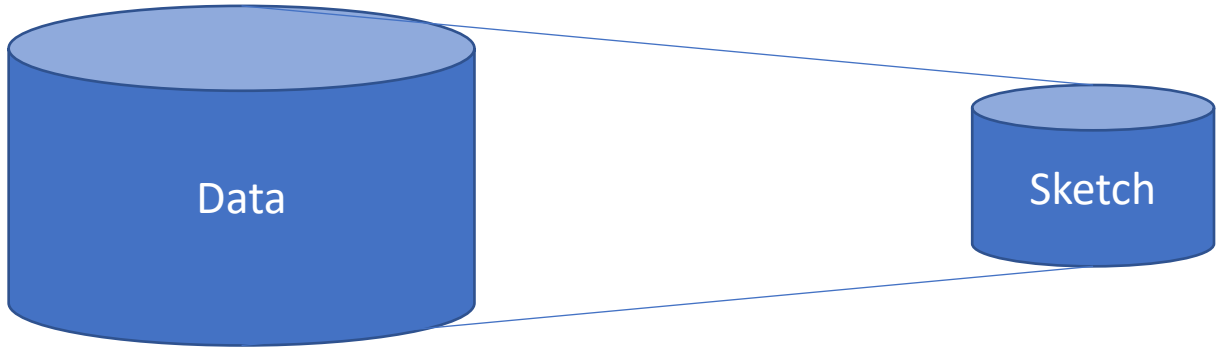
Given sequence of elements from some domain:

$$< x_1, x_2, x_3, \dots >$$

- Data is read sequentially in **one** pass
- Bounded storage - limited in the size of working memory
- Fast processing time per stream element
- We want to create and maintain a sketch which allows us to obtain good estimates of properties

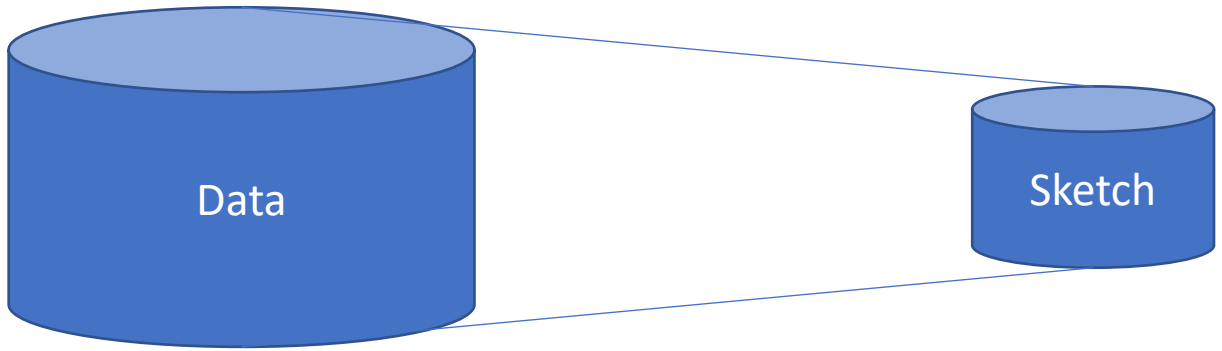
Sketch

A small summary of a large data set that (approximately) captures some statistics/properties we are interested in.

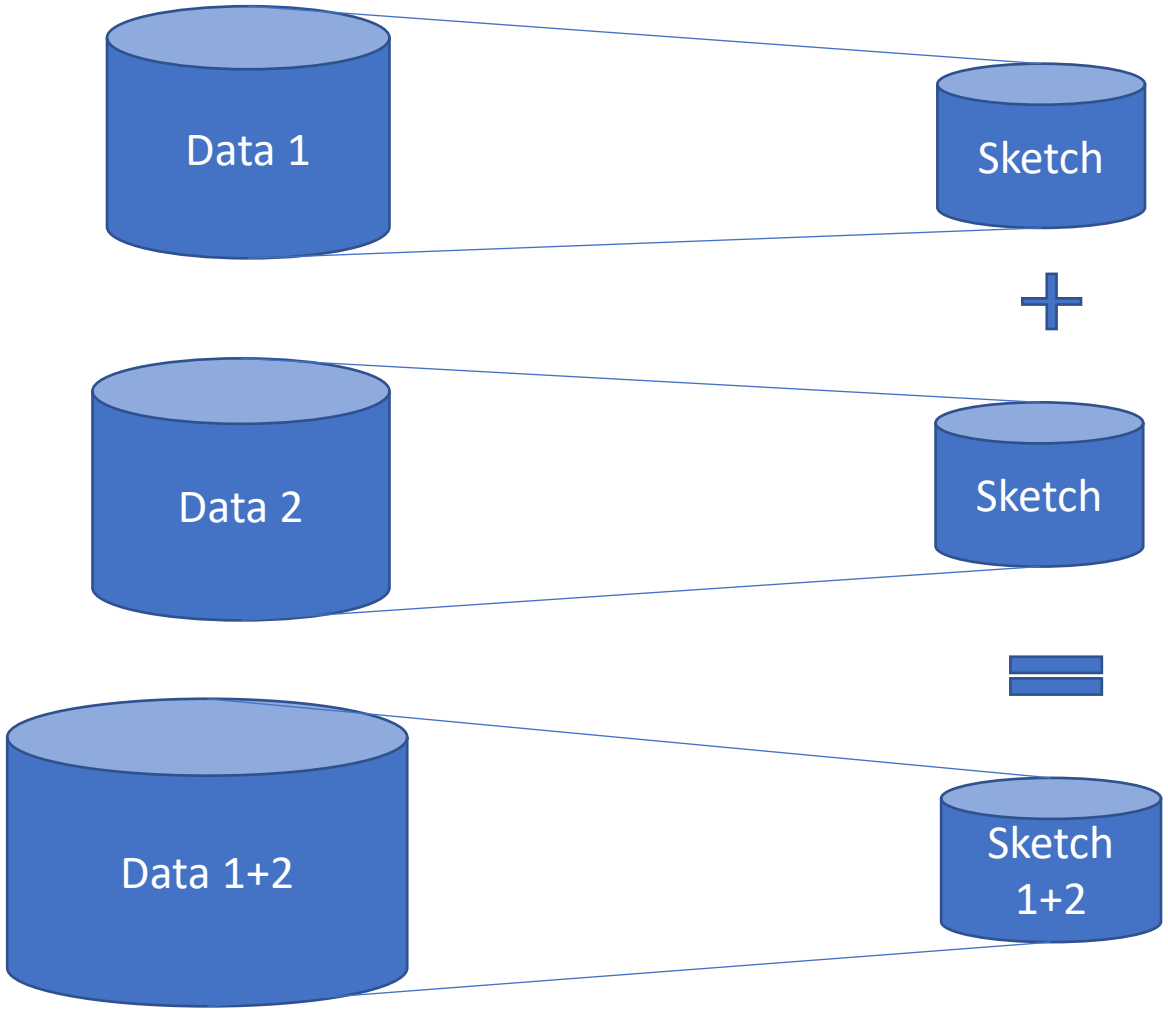


Sketch

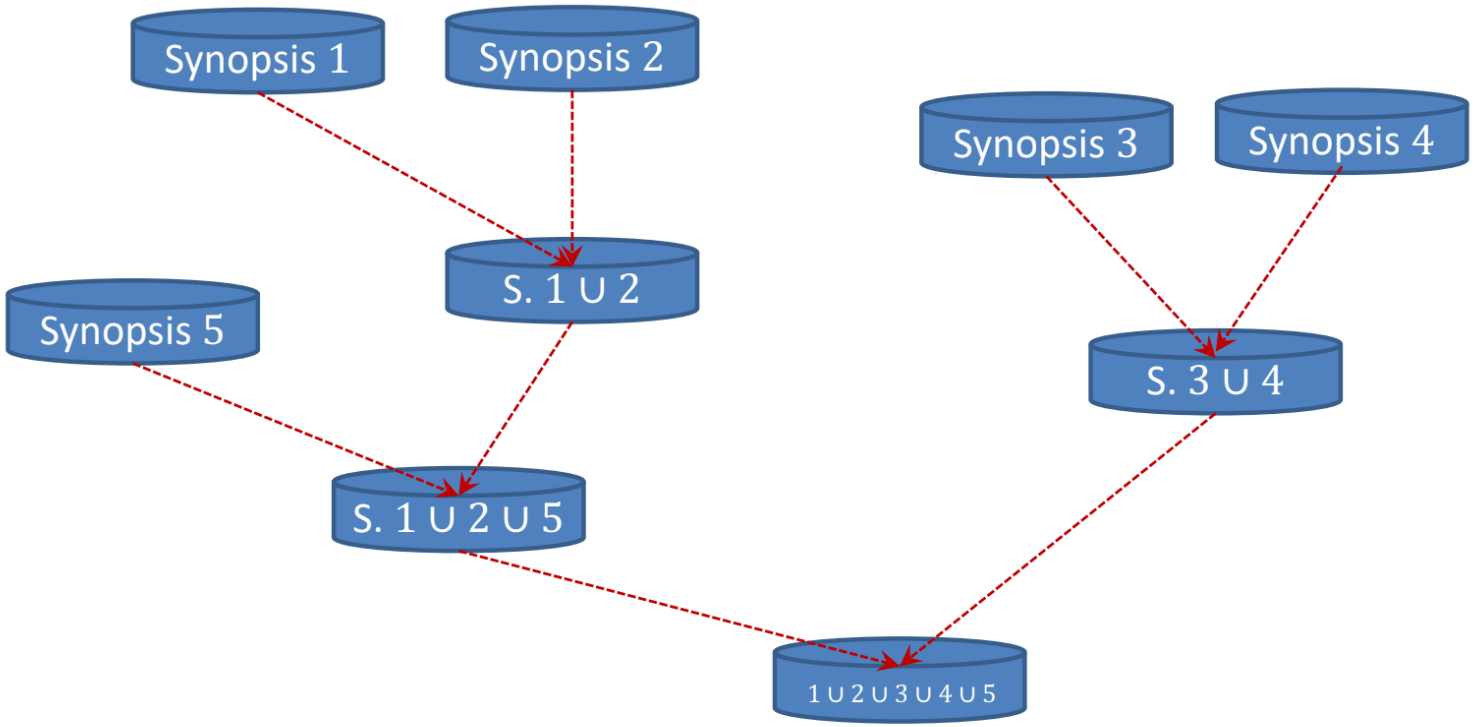
- Easy to add an element
- Mergeable : can create summary of union from summaries of data sets
- Deletions/“undo” support
- Flexible: supports multiple types of queries



Sketch Mergeability



Why Mergeability?



Count Elements in Stream

Simple Counting

Consider the following simple counting application:

counter = 0
for each element arrives: counter += 1

How much space does it take to maintain the count of the events?

We need to store *counter* of size n, which we can do in $O(\log n)$ bits.

Suppose n is very large, so $O(\log n)$ bits **cannot fit in the storage**.

Morris's Algorithm

Given very large stream values of $a_1, a_2, \ldots, a_n$ arriving one by one, how many values are in the stream?

Naïve: using a counter, count the exact number of values

- Perfect solution, 100% accurate
- Needs $O(\log n)$ storage (number of bits needed to store the value of 'n'). It can be also huge for very large n.

Morris's Algorithm

Idea: Track logn intead of n, use log(logn) bits instead of logn bits.

Initialize X = 0

When an item arrives, increase X by 1 with probability $\frac{1}{2^X}$

When the stream is over, output our estimate $2^X - 1$

Claim: $E[2^X] = n + 1$

Proof: Induction (not here)

Claim: $V[2^X] = n^2$

Proof: not here

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1							
$\boldsymbol{X}$	0							
$\boldsymbol{p = 2^{-X}}$	1							
Estimation $\boldsymbol{2^X - 1}$	0							

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2						
$\boldsymbol{X}$	0	1						
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$						
Estimation $\boldsymbol{2^X - 1}$	0	1						

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3					
$\boldsymbol{X}$	0	1	1					
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$					
Estimation $\boldsymbol{2^X - 1}$	0	1	1					

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3	4				
$\boldsymbol{X}$	0	1	1	2				
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$				
Estimation $\boldsymbol{2^X - 1}$	0	1	1	3				

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3	4	5			
$\boldsymbol{X}$	0	1	1	2	2			
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$			
Estimation $\boldsymbol{2^X - 1}$	0	1	1	3	3			

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3	4	5	6		
$\boldsymbol{X}$	0	1	1	2	2	2		
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$		
Estimation $\boldsymbol{2^X - 1}$	0	1	1	3	3	3		

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3	4	5	6	7	
$\boldsymbol{X}$	0	1	1	2	2	2	2	
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	
Estimation $\boldsymbol{2^X - 1}$	0	1	1	3	3	3	3	

Morris's Algorithm

Stream	1	1	1	1	1	1	1	1
Real Count	1	2	3	4	5	6	7	8
$\boldsymbol{X}$	0	1	1	2	2	2	2	3
$\boldsymbol{p = 2^{-X}}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$
Estimation $\boldsymbol{2^X - 1}$	0	1	1	3	3	3	3	7

How to Ensure $O(loglogn)$ Memory?

By the claim, $E[2^X] = n + 1$, taking log form both sides:

$$\log(2^x) = \log(n + 1)$$

$$X \approx \log(n)$$

To store $X \approx \log(n)$ we need to use $O(\log\log(n))$ bits.

Is Morris's Algorithm Good?

Chebyshev

$$P[|X - \mu| \geq \mu\epsilon] \leq \frac{\sigma^2}{\mu^2\epsilon^2}$$

$$P[|X - \mu| \geq \epsilon n] \leq \frac{\sigma^2}{\mu^2\epsilon^2} = \frac{n^2}{\epsilon^2 n^2} = \frac{1}{\epsilon^2}$$

By approximation rules:

$$\frac{1}{\epsilon^2} < \delta$$

Choosing $\epsilon = 0.1$ raise $\delta > 100$

Morris's Algorithm – Naïve Implementation

Attached practice.ipynb

Morris's Algorithm – Beta Version

Idea: instead of 1 counter, maintain s counters and average their results.

Initialize each $X_i = 0$

When an item arrives, increase each X_i by 1 independently with probability $\frac{1}{2^{X_i}}$

When the stream is over, output our estimate $\frac{1}{s} \cdot \sum(2^{X_i} - 1)$

Is Morris's Beta-Version Good?

By the linearity of variance (independent variables), we know that

$$V[2^X] = n^2$$

$$Var\left[\frac{1}{s} \cdot \sum (2^{X_i} - 1)\right] = \frac{1}{s^2} \cdot sn^2 = \frac{n^2}{s}$$

Using Chebyshev:

$$P[|X - E[X]| \geq c] \leq \frac{Var[X]}{c^2}$$

where $c = \epsilon n$ (error for each data point)

Is Morris's Beta-Version Good?

Using Chebyshev:

$$P[|X-\mu| \geq \epsilon n] \leq \frac{\sigma^2}{\mu^2\epsilon^2} = \frac{\frac{n^2}{s}}{\epsilon^2 n^2} = \frac{1}{s\epsilon^2}$$

By approximation rules:

$$\frac{1}{s\epsilon^2} < \delta$$

So choosing $s = \frac{1}{\delta\epsilon^2}$ counters we will get any estimation accuracy we want ☺

Is Morris's Beta-Version Good?

We want to have an error of $\epsilon = \frac{1}{4}$ only 1% $(\delta = 0.01)$ of the time (99% we are good!), so:

$$s = \frac{1}{\delta \epsilon^2} = \frac{1}{0.01 \cdot 0.25^2} = 1600$$

We need 1600 estimators to get the required accuracy ☺

By keeping s estimators we can control the error rate!

Count Distinct Elements

Data Streams

Stream of m elements from $\{1,2,\dots,n\}$. Example:

$$< x_1, x_2, \dots, x_m > = < 5,8,1,1,1, \dots, 10 >$$

Let f_a be the frequency of element a in the stream (number of occurrences).

Formally,

$$f_a = |\{a_i = a\}|$$

By the example, $f_1 = 3$

Frequency Moments – Special Cases

1. $F_0 = number\ of\ distinct\ elements$
2. $F_1 = \sum f_a$ (counting elements like Morris)
3. $F_2 = \sum {f_a}^2$ (gini-index, not here)

Counting Distinct Elements

Sometimes, we need to know how many **UNIQUE** rows exist in a table.

In a Relational Database World there is a simple but **high-costly** action for DB engine level (for example **DISTINCT** or **GROUP BY** with subquery).

In the modern era of Big and Streaming Data, we often face the speed of our request to the database and just approximate number is enough for our needs.

Counting Distinct Elements

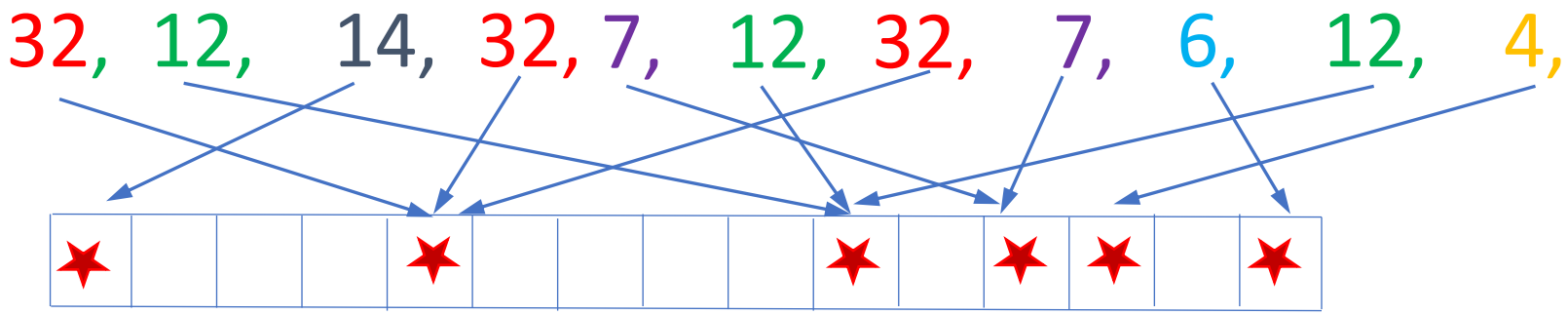
32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

- Elements occur multiple times, we want to count the number of *distinct* elements.
- Number of distinct element is $\boldsymbol{n}$ (=6 in example)
- Total number of elements is 11 in this example

Exact counting of n distinct element requires a structure of size $\Omega(n)!$

We are happy with an *approximate* count that uses a *small-size* working memory.

Distinct Elements: Exact Solution



Exact solution:

- Maintain an array/associative array/ hash table
- Hash/place each element to the table
- Query: count number of entries in the table

Problem: For n distinct elements, size of table is $\Omega(n)$

Distinct Elements: Approximate Counting

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

We want to be able to compute and maintain a small sketch $s(N)$ of the set N of distinct items seen so far $N = \{32, 12, 14, 7, 6, 4\}$

Distinct Elements: Approximate Counting

32, 12, 14, 32, 7, 12, 32, 7, 6, 12, 4,

Size-estimation/Minimum value technique:[Flajolet-Martin 1985]

Let $h(x)$ be a hash function

For each element a in the stream

Find h(a) in binary form

Count the number of trailing zeros

Return $2^{Max-Count}$

Distinct Elements: Approximate Counting

Input: $X = \{1,3,5,7,5,2,7\}$, $h(x_i) = (3x_i + 1) mod 5$

Step 1: Calculate hash for each element

x_i	$h(x_i)$
1	$(3 \cdot 1 + 1) mod 5 = 4$
3	$(3 \cdot 3 + 1) mod 5 = 0$
5	$(3 \cdot 5 + 1) mod 5 = 1$
7	$(3 \cdot 7 + 1) mod 5 = 2$
5	$(3 \cdot 5 + 1) mod 5 = 1$
2	$(3 \cdot 2 + 1) mod 5 = 2$
7	$(3 \cdot 7 + 1) mod 5 = 2$

Distinct Elements: Approximate Counting

Input: $X = \{1,3,5,7,5,2,7\}$, $h(x_i) = (3x_i + 1) mod 5$

Step 2: Find the binary form of each hash value

x_i	$h(x_i)$	Binary $h(x_i)$
1	$(3 \cdot 1 + 1) mod 5 = 4$	100
3	$(3 \cdot 3 + 1) mod 5 = 0$	000
5	$(3 \cdot 5 + 1) mod 5 = 1$	001
7	$(3 \cdot 7 + 1) mod 5 = 2$	010
5	$(3 \cdot 5 + 1) mod 5 = 1$	001
2	$(3 \cdot 2 + 1) mod 5 = 2$	010
7	$(3 \cdot 7 + 1) mod 5 = 2$	010

Distinct Elements: Approximate Counting

Input: $X = \{1,3,5,7,5,2,7\}$, $h(x_i) = (3x_i + 1) mod 5$

Step 3: Find r_i - the count of trailing zeros

x_i	$h(x_i)$	Binary $h(x_i)$	r_i
1	$(3 \cdot 1 + 1) mod 5 = 4$	100	2
3	$(3 \cdot 3 + 1) mod 5 = 0$	000	0
5	$(3 \cdot 5 + 1) mod 5 = 1$	001	0
7	$(3 \cdot 7 + 1) mod 5 = 2$	010	1
5	$(3 \cdot 5 + 1) mod 5 = 1$	001	0
2	$(3 \cdot 2 + 1) mod 5 = 2$	010	1
7	$(3 \cdot 7 + 1) mod 5 = 2$	010	1

Distinct Elements: Approximate Counting

Input: $X = \{1,3,5,7,5,2,7\}$, $h(x_i) = (3x_i + 1) mod 5$

Step 4: Estimate number of unique elements by

$$R = 2^{\max(r_i)} = 2^2 = 4$$

By fact, distinct values is 5. So, almost equal ☺

Claim: FM algorithm raise $E[X] = \frac{1}{n+1}$, $V[X] = \frac{n}{(n+1)^2(n+2)}$

F_0 Estimation – Flajolet-Martin

Claim: $E[X] = \frac{1}{n+1},\ V[X] = \frac{n}{(n+1)^2(n+2)}$

Does it good enough?

$$P\left[|ALG - OPT| > \epsilon \cdot E[X]\right] < \frac{Var[X]}{(\epsilon \cdot E[X])^2} =$$

$$= \frac{\frac{n}{(n+1)^2(n+2)}}{\epsilon^2 \cdot \frac{1}{(n+1)^2}} = \frac{n}{(n+1)^2(n+2)} \cdot \frac{(n+1)^2}{\epsilon^2} \approx \frac{1}{\epsilon^2}$$

To bound the error, we need $\frac{1}{\epsilon^2} < \delta$

F_0 Estimation – Flajolet-Martin Final Version

Use random hash function $h: stream \rightarrow [0,1]$

Run FM q times in parallel to obtain $X_1, \ldots, X_q$

Calculate $Z = \frac{1}{q} \sum X_i$

Output $\frac{1}{Z} - 1$

F_0 Estimation – Flajolet-Martin Final Version

Expectation of final version:

$$E[Z] = E\left[\frac{1}{q}\sum_{i=1}^{q} X_i\right] = \frac{1}{q}\sum_{i=1}^{q} E[X_i] = \frac{1}{\mathrm{q}} \cdot q\left(\frac{1}{n+1}\right) = \frac{1}{n+1} = E[X]$$

Variance of final version:

$$Var[Z] = Var\left[\frac{1}{q}\sum_{i=1}^{q} X_i\right] = \frac{1}{q^2}\sum_{i=1}^{q} Var[X_i] =$$

$$\frac{1}{q^2} \cdot q\left(\frac{n}{(n+1)^2(n+2)}\right) = \frac{1}{q} \cdot \frac{n}{(n+1)^2(n+2)}$$

F_0 Estimation – Flajolet-Martin Final Version

Does it good enough?

$$P\left[|ALG - OPT| > \epsilon \cdot E[X]\right] < \frac{Var[X]}{(\epsilon \cdot E[X])^2} =$$

$$= \frac{\frac{1}{q} \cdot \frac{n}{(n+1)^2(n+2)}}{\epsilon^2 \cdot \frac{1}{(n+1)^2}} = \frac{1}{q} \cdot \frac{n}{(n+1)^2(n+2)} \cdot \frac{(n+1)^2}{\epsilon^2} \approx \frac{1}{q\epsilon^2}$$

To bound the error, we need $\frac{1}{q\epsilon^2} < \delta$ ☺

F_0 Estimation – Flajolet-Martin Final Version

In case we want error of $\epsilon = 0.01$ and probability error of 0.1 we need to choose:

$$\frac{1}{q \cdot 0.01^2} < 0.1$$

$$\frac{1}{0.01^2} < 0.1q$$

$$q > 100{,}000$$