

Counting Distinct Elements

Definition

- Data stream consists of a universe of elements chosen from a set of N
- Maintain a count of number of distinct items seen so far.

Let us consider a stream :

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 n (=6 in this example) Number of elements in this example is 11

Why do we count distinct elements?

- Number of distinct queries issued
- Unique IP addresses passing packages through a router
- Number of unique users accessing a website per month
- Number of different people passing through a traffic hub (airport, etc.)
- How many unique products we sold tonight?
- How many unique requests on a website came in today?
- How many different words did we find on a website?
 - Unusually large number of words could be indicative of spam

Question: how would you do it?

Now, let's constrain ourselves with limited storage...

- How to estimate (in an unbiased manner) the number of distinct elements seen?

- **Flajolet-Martin (FM) Approach**

- FM algorithms approximate the number of unique objects in a stream or a database in one pass.
- If the stream contains n elements with m of them unique, this algorithm runs in $O(n)$ time and needs $O(\log(m))$ memory.

FM-sketch

(Flajolet-Martin)

Task: given a data stream, estimate the number of distinct elements occurring in it.

- **Approach:** hash data stream elements uniformly to N bit values, i.e.:
$$h : a_i \rightarrow \{0, 1\}^N$$
- **Assumption:** the larger the number of distinct elements in the stream, the more distinct the occurring hash values, and the more likely one with an “unusual” property appears

FM-sketch (Flajolet-Martin)

- One possibility of interpreting “unusual” is the **hash tail**: the number of 0’s a binary hash value ends in

100110101110 100110101100 100110000000

Algorithm:

*Maximum hash
tail seen so far*



for all $a_i \in S$,
 $h(a_i) \rightarrow \{0,1\}^N$
 $R = \max_{a_i \in S} h(a_i)$
return $|S| = 2^R$

Important:

N must be long enough:
there must be more
possible results of the
hash function than
elements in the
universal set.

FM-sketch (Flajolet-Martin)

Pick a hash function that maps each of the N elements to at least $\log_2 N$ bits

For each stream element a , let $r(a)$ be the number of trailing 0s in $h(a)$

$r(a)$ = position of first 1 counting from the right

E.g., say $h(a) = 12$, then 12 is 1100 in binary, so $r(a) = 2$

Record R = the maximum $r(a)$ seen

$R = \max_a r(a)$, over all the items a seen so far

Estimated number of distinct elements = 2^R

Median of means for a stable result

FM-sketch (Flajolet-Martin)

- Intuitive justification:

$$P(h(a) \text{ has tail length of at least } r) = \frac{1}{2 \times 2 \dots \times 2} = \frac{1}{2^r}$$

r 0's occur

- When there are m distinct elements in the stream

$$P(\text{none has tail length} \geq r) = \left(1 - \frac{1}{2^r}\right)^m$$

if $m \gg 2^r$: the prob. of finding a tail $\geq r$ reaches 1

if $m \ll 2^r$: the prob. of finding a tail $\geq r$ reaches 0

Thus: the proposed estimate is neither too low nor too high.

Counting Distinct element in a stream

Naïve Solution/Algorithm

```
SET COUNTER = 0
SET UNIQUE_SET = []
WHILE COUNTER NOT EQUALS LAST ELEMENT INDEX:
    IF CURRENT ELEMENT NOT PRESENT IN UNIQUE_SET:
        ADD CURRENT ELEMENT IN UNIQUE_SET
    INCREMENT THE COUNTER
DISPLAY COUNT OF DISTINCT ELEMENTS : LENGTH(UNIQUE_SET)
```


Flajolet-Martin Algorithm

Overview

- To find the **approximate** number of distinct elements in a stream
- In a **single pass**
- Uses very **less** memory space while executing
- Hence, **efficient** and robust
- Note: This algorithm is meant to be used when the stream of elements as well as the expected distinct element count is very very large

Flajolet-Martin Algorithm

Pseudocode / Algorithm

1. Select a hash function $h(x)$ so each element in the set is mapped to a value to at least $\log_2 n$ bits
 2. Convert this $h(x)$ output to binary_value
 3. For each binary_value , find $r(\text{binary_value}) = \text{length of the trailing zeroes in binary_value}$
 4. Find $R = \max(r(\text{binary_value}))$
 5. Finally, Approximate count of distinct elements will be 2^R
- 

Flajolet-Martin Algorithm

Example

Consider stream, $x = [1, 5, 10, 5, 15, 1]$ and hash function $h(x) = x \bmod 11$
Find the count of distinct/unique elements using the FM Algorithm

Flajolet-Martin Algorithm

Consider stream, $x = [1, 5, 10, 5, 15, 1]$ and hash function $h(x) = x \bmod 11$
Find the count of distinct/unique elements using the FM Algorithm

x	h(x)	Binary	Count trailing 0	R = Max of count of trailing 0	Distinct elements count = 2^R

Flajolet-Martin Algorithm

Consider stream, $x = [1, 5, 10, 5, 15, 1]$ and hash function $h(x) = x \bmod 11$

Find the count of distinct/unique elements using the FM Algorithm

x	$h(x)$	Binary	Count trailing 0	$R = \text{Max of count of trailing 0}$	Distinct elements count = 2^R
1	$1 \bmod 11$ 1	1	0	2	2^2 4
5	$5 \bmod 11$ 5	101	0		
10	$10 \bmod 11$ 10	1010	1		
5	$5 \bmod 11$ 5	101	0		
15	$15 \bmod 11$ 4	100	2		
1	$1 \bmod 11$ 1	1	0		

Example

Determine the distinct element in the stream using FM.

➤ Input stream of integers $x = 1, 3, 2, 1, 2, 3, 4, 3, 1, 2, 3, 1$.

➤ Hash Function, $h(x) = 6x + 1 \bmod 5$

$$h(1) = 6(1) + 1 \bmod 5$$

$$= 6 + 1 \bmod 5$$

$$= 7 \bmod 5$$

$$= 2$$

Therefore $h(1) = 2$

Similarly, calculate hash function for the complete stream.

Calculate hash

functions $h(v)$

➤ For the given input stream 1,3,2,1,2,3,4,3,1,2,3,1.

$$h(x) = 6x + 1 \mod 5$$

$$h(1) = 2$$

$$h(4) = 0$$

$$h(3) = 4$$

$$h(3) = 4$$

$$h(2) = 3$$

$$h(1) = 2$$

$$h(1) = 2$$

$$h(2) = 3$$

$$h(2) = 3$$

$$h(3) = 3$$

$$h(3) = 4$$

$$h(1) = 2$$

The numbers obtained are:

$\{2, 4, 3, 2, 3, 4, 0, 4, 2, 3, 3, 2\}$

Binnary

- For the given input stream 1,3,2,1,2,3,4,3,1,2,3,1.
- For every hash function calculated, write the binary equivalent for the same.

$h(1) = 2 = 010$	$h(4) = 0 = 000$
$h(3) = 4 = 100$	$h(3) = 4 = 100$
$h(2) = 3 = 011$	$h(1) = 2 = 010$
$h(1) = 2 = 010$	$h(2) = 3 = 011$
$h(2) = 3 = 011$	$h(3) = 4 = 100$
$h(3) = 4 = 100$	$h(1) = 2 = 010$

Convesion to binanary:

{010,100,011,010,011,100,000,100,010,011,100,010}

Trailing

- For the given input stream 1,3,2,1,2,3,4,3,1,2,3,1.
- For every hash function calculated and the binary equivalent for the same is written.
- Now write the count of trailing zeros in each hash function bit.

$$h(1) = 2 = 010 = 1$$

$$h(3) = 4 = 100 = 2$$

$$h(2) = 3 = 011 = 0$$

$$h(1) = 2 = 010 = 1$$

$$h(2) = 3 = 011 = 0$$

$$h(3) = 4 = 100 = 2$$

$$h(4) = \underline{0} = 000 = 0$$

$$h(3) = 4 = 100 = 2$$

$$h(1) = 2 = 010 = 1$$

$$h(2) = 3 = 011 = 0$$

$$h(3) = 4 = 100 = 2$$

$$h(1) = 2 = 010 = 1$$

Computing $r(a)$:

$\{1, 2, 0, 1, 0, 2, 0, 2, 1, 0, 2, 1\}$

Distinct

Elements

➤ From the binary equivalent trailing zero values, write the value of maximum number of trailing zeros.

So, $r(a)$: {1,2,0,1,0,2,0,2,1,0,2,1}

$$R = \max r(a) = 2$$

$$\text{Estimate} = 2^R = 2^2 = 2 * 2 = 4$$

➤ Hence , there are 4 distinct elements as 1,3,2,4

Counting Distinct Elements

Problem:

Data stream consists of a universe of elements chosen from a set of size N
Maintain a count of the number of distinct elements seen so far

Obvious approach:

Maintain the set of elements seen so far

That is, keep a hash table of all the distinct elements seen so far

Applications

How many different words are found among the Web pages being crawled at a site?

Unusually low or high numbers could indicate artificial pages (spam?)

How many different Web pages does each customer request in a week?

How many distinct products have we sold in the last week?

Using Small Storage

Real problem: What if we do not have space to maintain the set of elements seen so far?

Estimate the count in an unbiased way

Accept that the count may have a little error, but limit the probability that the error is large

Flajolet-Martin Approach

Pick a hash function h that maps each of the N elements to at least $\log_2 N$ bits

For each stream element a , let $r(a)$ be the number of trailing 0s in $h(a)$

$r(a)$ = position of first 1 counting from the right

E.g., say $h(a) = 12$, then 12 is 1100 in binary, so $r(a) = 2$

Record R = the maximum $r(a)$ seen

$R = \max_a r(a)$, over all the items a seen so far

Estimated number of distinct elements = 2^R

Why It Works: Intuition

Very very rough and heuristic intuition why Flajolet-Martin works:

$h(a)$ hashes a with equal prob. to any of N values

Then $h(a)$ is a sequence of $\log_2 N$ bits,

where 2^{-r} fraction of all a s have a tail of r zeros

About 50% of a s hash to $***0$

About 25% of a s hash to $**00$

So, if we saw the longest tail of $r=2$ (i.e., item hash ending $*100$) then we have probably seen

about 4 distinct items so far

So, it takes to hash about 2^r items before we see one with zero-suffix of length r

Why It Works: More formally

Now we show why Flajolet-Martin works

Formally, we will show that **probability of finding a tail of r zeros:**

Goes to **1** if $m \gg 2^r$

Goes to **0** if $m \ll 2^r$

where m is the number of distinct elements
seen so far in the stream

Thus, 2^R will almost always be around m !

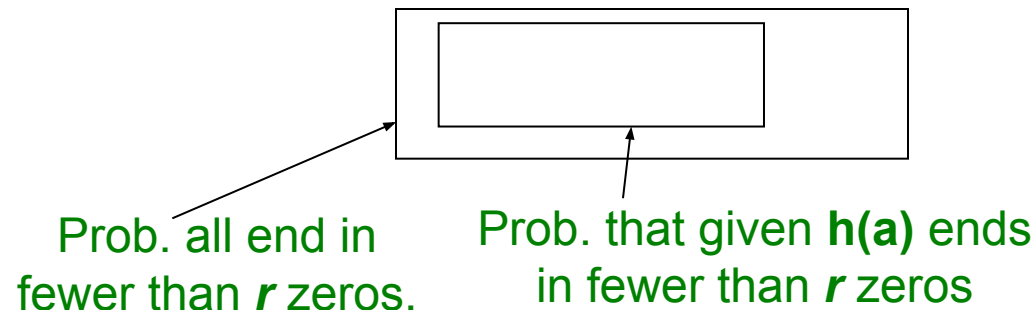
Why It Works: More formally

What is the probability that a given $h(a)$ ends in at least r zeros is 2^{-r}

$h(a)$ hashes elements uniformly at random
Probability that a random number ends in at least r zeros is 2^{-r}

Then, the probability of **NOT** seeing a tail of length r among m elements:

$$(1 - 2^{-r})^m$$



Why It Works: More formally

Note: $(1 - 2^{-r})^m = (1 - 2^{-r})^{2^r (m 2^{-r})} \approx e^{-m 2^{-r}}$

Prob. of NOT finding a tail of length r is:

If $m \ll 2^r$, then prob. tends to **1**
as $m/2^r \rightarrow 0$ $(1 - 2^{-r})^m \approx e^{-m 2^{-r}} = 1$

So, the probability of finding a tail of length r tends to **0**

If $m \gg 2^r$, then prob. tends to **0**
as $m/2^r \rightarrow \infty$ $(1 - 2^{-r})^m \approx e^{-m 2^{-r}} = 0$

So, the probability of finding a tail of length r tends to **1**

Thus, 2^R will almost always be around m !

Why It Doesn't Work

$E[2^R]$ is actually infinite

Probability halves when $R \rightarrow R+1$, but value doubles

Workaround involves using many hash functions h_i and getting many samples of R_i

How are samples R_i combined?

Average? What if one very large value 2^{R_i} ?

Median? All estimates are a power of 2

Solution:

Partition your samples into small groups

Take the median of groups

Then take the average of the medians

(3) Computing Moments

Generalization: Moments

Suppose a stream has elements chosen from a set A of N values

Let m_i be the number of times value i occurs in the stream

The k^{th} *moment* is

$$\sum_{i \in A} (m_i)^k$$

Special Cases

$$\sum_{i \in A} (m_i)^k$$

0th moment = number of distinct elements

The problem just considered

1st moment = count of the numbers of elements

= length of the stream

Easy to compute

2nd moment = *surprise number S* =

a measure of how uneven the distribution is

Example: Surprise Number

Stream of length 100
11 distinct values

Item counts: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 Surprise $S = 910$

Item counts: 90, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 Surprise $S = 8,110$

AMS Method

- Computing moments of any order requires main memory, a count for each element that appears in the stream.
- If not to use that much memory, then we need to estimate the k th moment by keeping a limited number of values in main memory and computing an estimate from these values.
- For the case of distinct elements, each of these values were counts of the longest tail produced by a single hash function.
- To calculate second and higher moments we use

The Alon-Matias-Szegedy Algorithm for Second Moments

AMS Method

AMS method works for all moments

Gives an unbiased estimate

We will just concentrate on the 2nd moment S

We pick and keep track of many variables X :

For each variable X we store $X.el$ and $X.val$

$X.el$ corresponds to the item i

$X.val$ corresponds to the **count** of item i

Note this requires a count in main memory,
so number of X s is limited

Our goal is to compute $S = \sum_i m_i^2$

One Random Variable (X)

How to set $X.val$ and $X.el$?

Assume stream has length n (we relax this later)

Pick some random time t ($t < n$) to start,
so that any time is equally likely

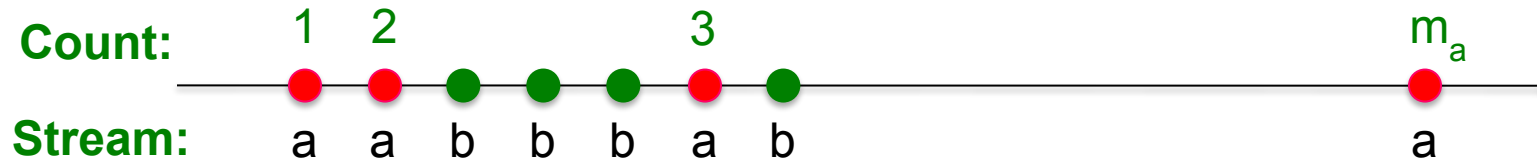
Let at time t the stream have item i . **We set $X.el = i$**

Then we maintain count c (**$X.val = c$**) of the number of i s in the stream starting from the chosen time t

- **Then the estimate of the 2nd moment ($\sum_i m_i^2$) is:**
 $S = f(X) = n(2 \cdot c - 1)$

Note, we will keep track of multiple X s, (X_1, X_2, \dots, X_k)
and our final estimate will be **$S = 1/k \sum_j f(X_j)$**

Expectation Analysis



2nd moment is $S = \sum_i m_i^2$

c_t ... number of times item at time t appears from time t onwards ($c_1=m_a$, $c_2=m_a-1$, $c_3=m_b$)

$$E[f(X)] = \frac{1}{n} \sum_{t=1}^n n(2c_t - 1)$$

m_i ... total count of item i in the stream (we are assuming stream has length n)

$$= \frac{1}{n} \sum_i n (1 + 3 + 5 + \dots + 2m_i - 1)$$

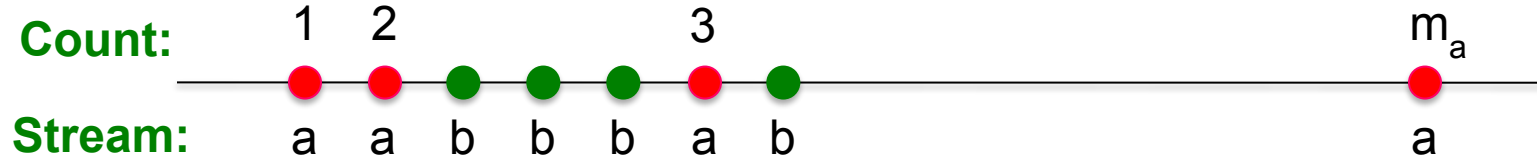
Group times by the value seen

Time t when the last i is seen ($c_t=1$)

Time t when the penultimate i is seen ($c_t=2$)

Time t when the first i is seen ($c_t=m_i$)

Expectation Analysis



$$E[f(X)] = \frac{1}{n} \sum_i n (1 + 3 + 5 + \dots + 2m_i - 1)$$

Little side calculation: $(1 + 3 + 5 + \dots + 2m_i - 1) = \sum_{i=1}^{m_i} (2i - 1) = 2 \frac{m_i(m_i+1)}{2} - m_i = (m_i)^2$

$$\text{Then } E[f(X)] = \frac{1}{n} \sum_i n (m_i)^2$$

$$\text{So, } E[f(X)] = \sum_i (m_i)^2 = S$$

We have the second moment (in expectation)!

Suppose the stream is a, b, c, b, d, a, c, d, a, b, d, c, a, a, b. The length of the stream is $n = 15$.

Since a appears 5 times, b appears 4 times, and c and d appear three times each

The second moment for the stream is $5^2 + 4^2 + 3^2 + 3^2 = 59$.
Suppose we keep three variables, X_1 , X_2 , and X_3 . Also

that at “random” we pick the 3rd, 8th, and 13th positions to define these three variables

$$X1=\{c,3\}$$

$$X2=\{d,2\}$$

$$X3=\{a,2\}$$

$$F(x)=n(2c-1)$$

Higher-Order Moments

For estimating k^{th} moment we essentially use the same algorithm but change the estimate:

For $k=2$ we used $n (2 \cdot c - 1)$

For $k=3$ we use: $n (3 \cdot c^2 - 3c + 1)$ (where $c=X.\text{val}$)

Why?

For $k=2$: Remember we had $(1 + 3 + 5 + \dots + 2m_i - 1)$ and we showed terms $2c-1$ (for $c=1, \dots, m$) sum to m^2

$$\sum_{c=1}^m 2c - 1 = \sum_{c=1}^m c^2 - \sum_{c=1}^m (c - 1)^2 = m^2$$

$$\text{So: } 2c - 1 = c^2 - (c - 1)^2$$

$$\text{For } k=3: c^3 - (c-1)^3 = 3c^2 - 3c + 1$$

Generally: Estimate = $n (c^k - (c - 1)^k)$

Combining Samples

In practice:

Compute $f(X) = n(2c - 1)$ for
as many variables X as you can fit in memory
Average them in groups
Take median of averages

Problem: Streams never end

We assumed there was a number n ,
the number of positions in the stream
But real streams go on forever, so n is
a variable – the number of inputs seen so far

Streams Never End: Fixups

(1) The variables X have n as a factor –
keep n separately; just hold the count in X

(2) Suppose we can only store k counts.

We must throw some X s out as time goes on:

Objective: Each starting time t is selected with probability k/n

Solution: (fixed-size sampling!)

Choose the first k times for k variables

When the n^{th} element arrives ($n > k$), choose it with probability k/n

If you choose it, throw one of the previously stored variables X out, with equal probability

Queries over a (long) Sliding Window

Sliding Windows

A useful model of stream processing is that queries are about a *window* of length N – the N most recent elements received

Interesting case: N is so large that the data cannot be stored in memory, or even on disk

Or, there are so many streams that windows for all cannot be stored

Amazon example:

For every product X we keep 0/1 stream of whether that product was sold in the n -th transaction

We want answer queries, how many times have we sold X in the last k sales

Sliding Window: 1 Stream

Sliding window on a single stream:

N = 6

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

q w e r t y u i o p a s d f g h j k l z x c v b n m

← Past Future →

Counting Bits (1)

Problem:

Given a stream of **0**s and **1**s

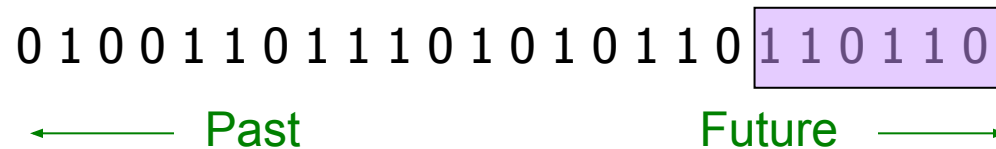
Be prepared to answer queries of the form

How many 1s are in the last k bits? where $k \leq N$

Obvious solution:

Store the most recent N bits

When new bit comes in, discard the $N+1^{\text{st}}$ bit



Suppose $N=6$

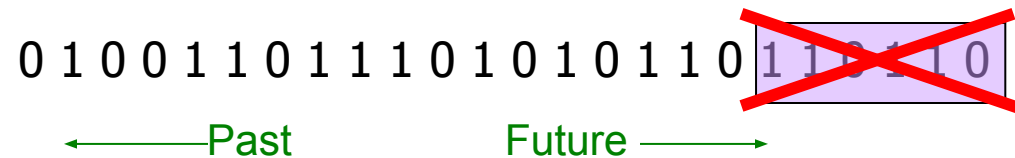
Counting Bits (2)

You can not get an exact answer without storing the entire window

Real Problem:

What if we cannot afford to store N bits?

E.g., we're processing 1 billion streams and
 $N = 1$ billion

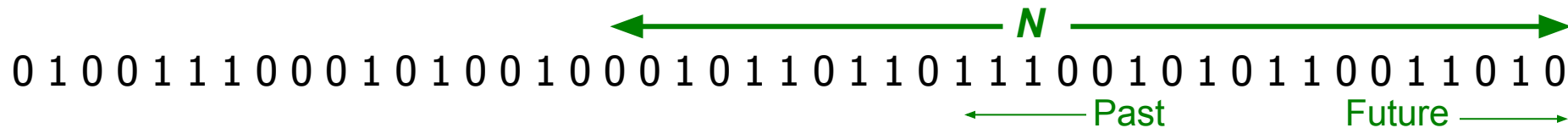


But we are happy with an approximate answer

An attempt: Simple solution

Q: How many 1s are in the last N bits?

A simple solution that does not really solve our problem: **Uniformity assumption**



Maintain 2 counters:

S : number of 1s from the beginning of the stream

Z : number of 0s from the beginning of the stream

How many 1s are in the last N bits? $N \cdot \frac{S}{S+Z}$

But, what if stream is non-uniform?

What if distribution changes over time?

DGIM Method

DGIM solution that does not assume uniformity

We store $O(\log^2 N)$ bits per stream

**Solution gives approximate answer,
never off by more than 50%**

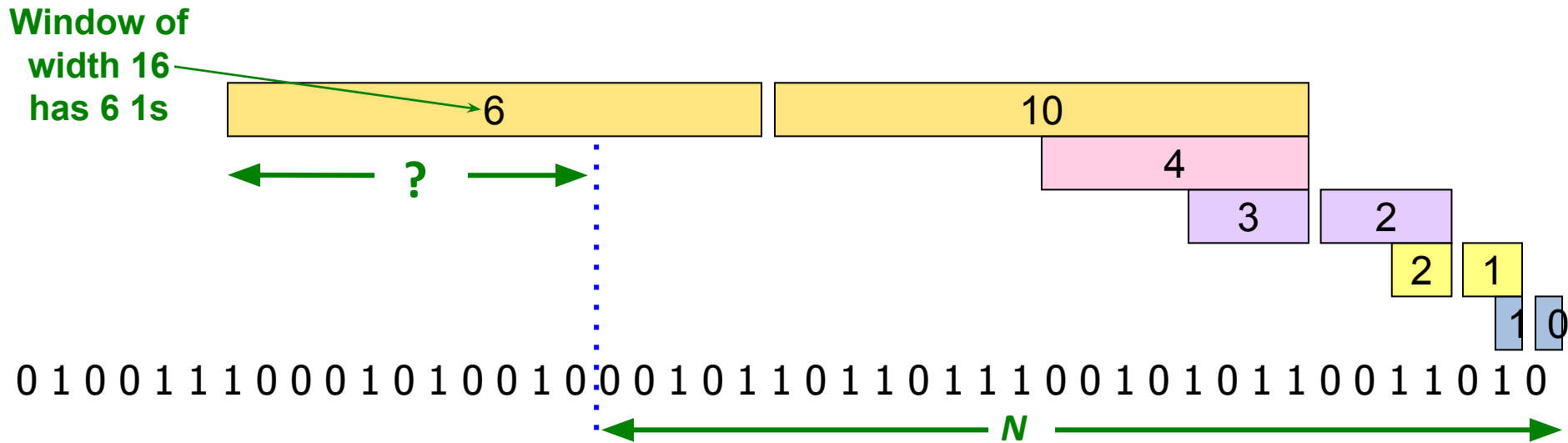
Error factor can be reduced to any fraction > 0 , with more complicated algorithm and proportionally more stored bits

Idea: Exponential Windows

Solution that doesn't (quite) work:

Summarize **exponentially increasing** regions
of the stream, looking backward

Drop small regions if they begin at the same point as a larger region



We can reconstruct the count of the last N bits, except we are not sure how many of the last 6 1s are included in the N

What's Good?

Stores only $O(\log^2 N)$ bits

$O(\log N)$ counts of $\log_2 N$ bits each

Easy update as more bits enter

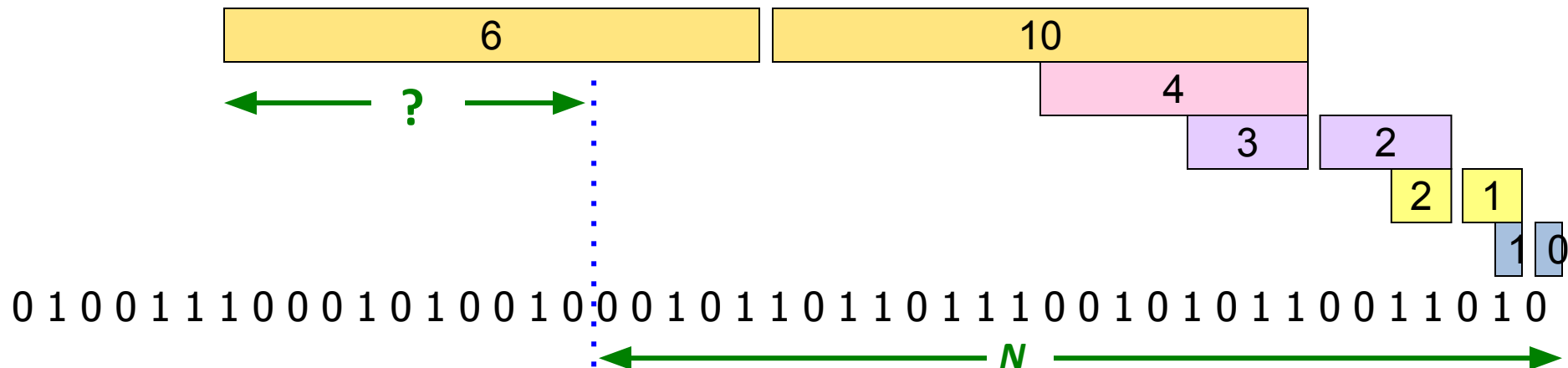
Error in count no greater than the number of **1s** in the “**unknown**” area

What's Not So Good?

As long as the **1s** are fairly evenly distributed, the error due to the unknown region is small – **no more than 50%**

But it could be that all the **1s** are in the unknown area at the end

In that case, **the error is unbounded!**

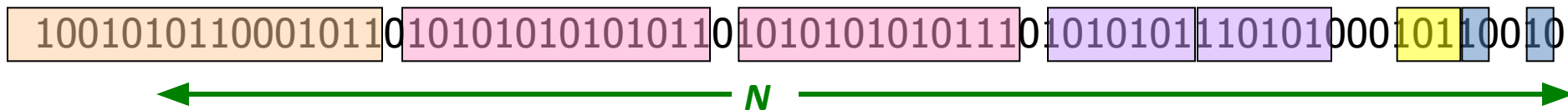


Fixup: DGIM method

Idea: Instead of summarizing fixed-length blocks, summarize blocks with specific number of **1s**:

Let the block *sizes* (number of **1s**) increase exponentially

When there are few 1s in the window, block sizes stay small, so errors are small



DGIM: Timestamps

Each bit in the stream has a *timestamp*, starting 1, 2, ...

Record timestamps modulo N (**the window size**), so we can represent any **relevant** timestamp in $O(\log_2 N)$ bits

DGIM: Buckets

A **bucket** in the DGIM method is a record consisting of:

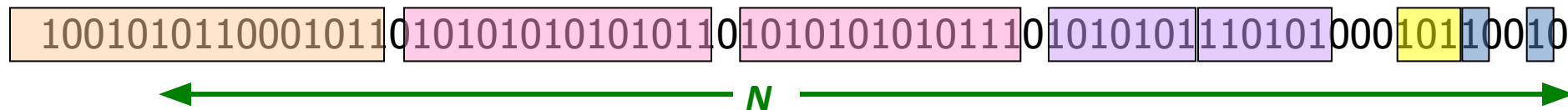
(A) The timestamp of its end [$O(\log N)$ bits]

(B) The number of 1s between its beginning and end [$O(\log \log N)$ bits]

Constraint on buckets:

Number of **1s** must be a power of 2

That explains the $O(\log \log N)$ in (B) above



Representing a Stream by Buckets

Either **one** or **two** buckets with the same **power-of-2 number of 1s**

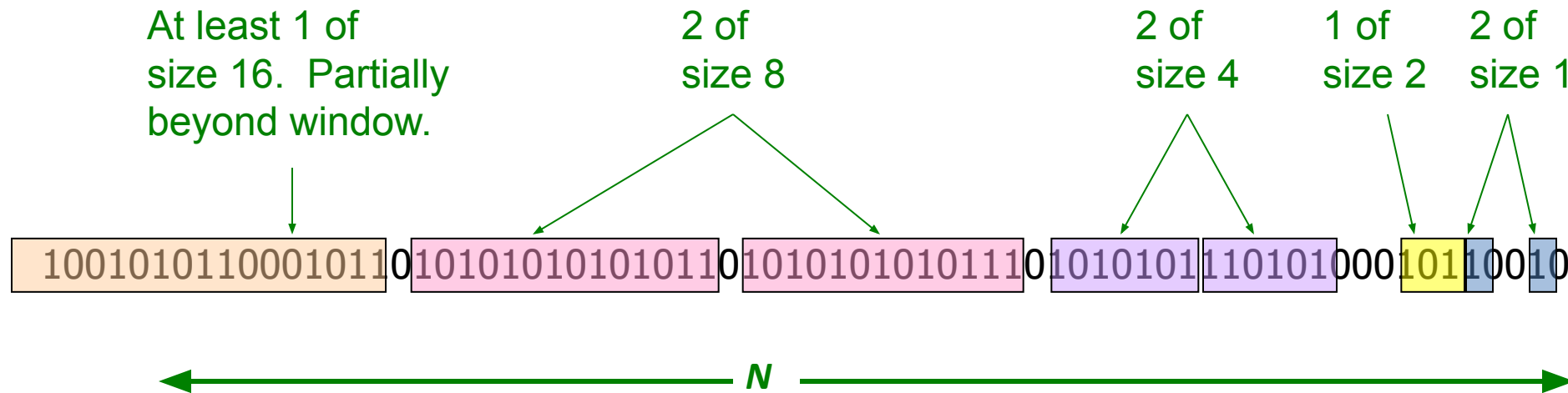
Buckets do not overlap in timestamps

Buckets are sorted by size

Earlier buckets are not smaller than later buckets

Buckets disappear when their
end-time is $> N$ time units in the past

Example: Bucketized Stream



Three properties of buckets that are maintained:

- Either **one** or **two** buckets with the same **power-of-2** number of **1s**
- Buckets do not overlap in timestamps
- Buckets are sorted by size

Updating Buckets (1)

When a new bit comes in, drop the last (oldest) bucket if its end-time is prior to ***N*** time units before the current time

2 cases: Current bit is **0** or **1**

If the current bit is 0:

no other changes are needed

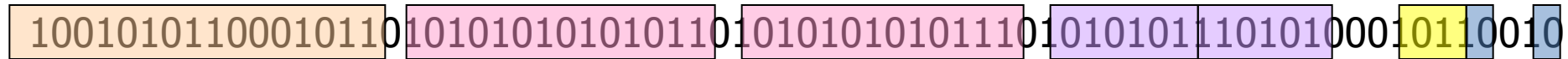
Updating Buckets (2)

If the current bit is 1:

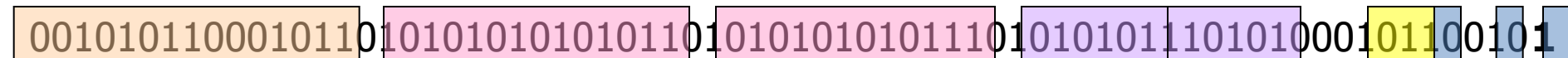
- (1) Create a new bucket of size 1, for just this bit
End timestamp = current time
- (2) If there are now **three buckets of size 1**,
combine the oldest two into a bucket of size 2
- (3) If there are now **three buckets of size 2**,
combine the oldest two into a bucket of size 4
- (4) And so on ...

Example: Updating Buckets

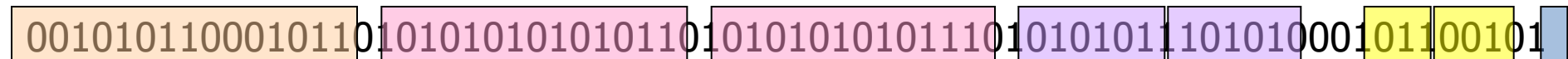
Current state of the stream:



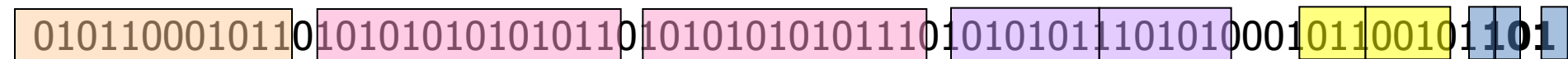
Bit of value 1 arrives



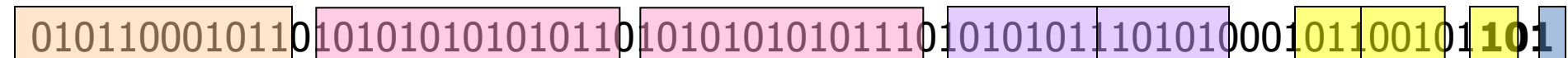
Two orange buckets get merged into a yellow bucket



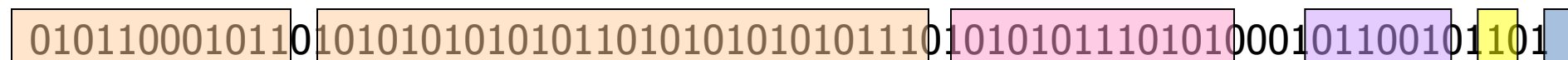
Next bit 1 arrives, new orange bucket is created, then 0 comes, then 1:



Buckets get merged...



State of the buckets after merging



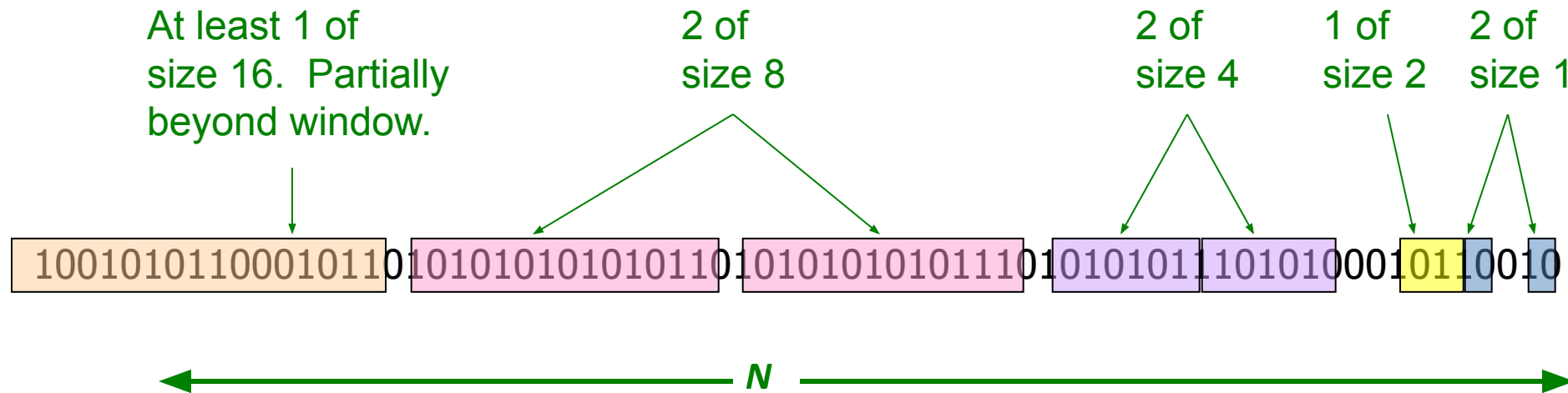
How to Query?

To estimate the number of 1s in the most recent N bits:

1. Sum the sizes of all buckets but the last
(note “size” means the number of 1s in the bucket)
2. Add half the size of the last bucket

Remember: We do not know how many **1s** of the last bucket are still within the wanted window

Example: Bucketized Stream



Error Bound: Proof

Why is error 50%? Let's prove it!

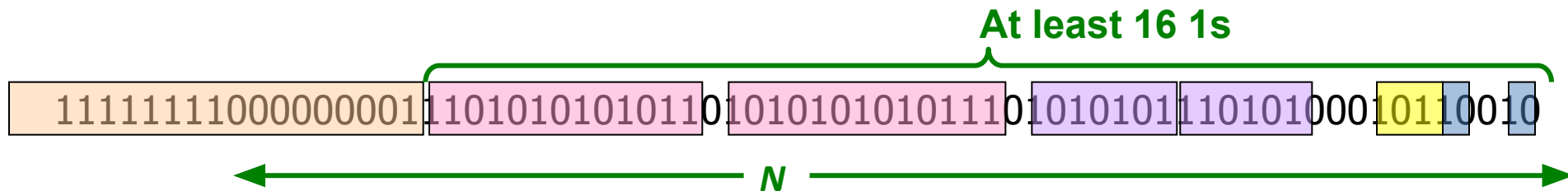
Suppose the last bucket has size 2^r

Then by assuming 2^{r-1} (i.e., half) of its 1s are still within the window, we make an error of at most 2^{r-1}

Since there is at least one bucket of each of the sizes less than 2^r , the true sum is at least

$$1 + 2 + 4 + \dots + 2^{r-1} = 2^r - 1$$

Thus, error at most 50%



Further Reducing the Error

Instead of maintaining **1** or **2** of each size bucket, we allow either **$r-1$** or **r** buckets (**$r > 2$**)

Except for the largest size buckets; we can have any number between **1** and **r** of those

Error is at most $O(1/r)$

By picking **r** appropriately, we can tradeoff between number of bits we store and the error

Extensions

Can we use the same trick to answer queries **How many 1's in the last k ?** where $k < N$?

A: Find earliest bucket **B** that overlaps with k .

Number of 1s is the **sum of sizes of more recent buckets + $\frac{1}{2}$ size of B**

10010101100010110101010101011010101010101110101010111010101011101010100010110010

Can we handle the case where the stream is not bits, but integers, and we want the sum of the last k elements?

Extensions

Stream of positive integers

We want the sum of the last k elements

Amazon: Avg. price of last k sales

Solution:

(1) If you know all have at most m bits

Treat m bits of each integer as a separate stream

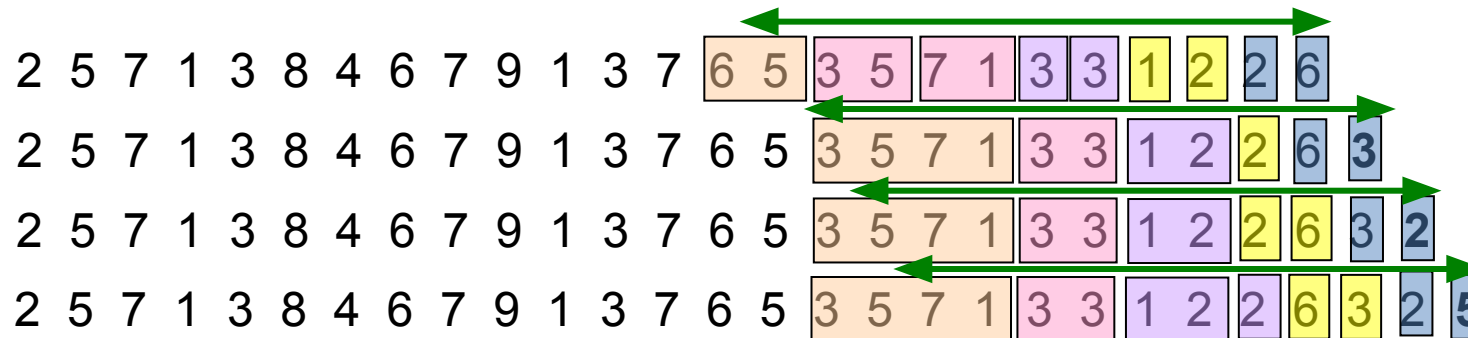
Use DGIM to count 1s in each integer

c_i ...estimated count for i -th bit

The sum is $= \sum_{i=0}^{m-1} c_i 2^i$

(2) Use buckets to keep partial sums

Sum of elements in size b bucket is at most 2^b



Idea: Sum in each bucket is at most 2^b (unless bucket has only 1 integer)

Bucket sizes:



Summary

Sampling a fixed proportion of a stream

Sample size grows as the stream grows

Sampling a fixed-size sample

Reservoir sampling

Counting the number of 1s in the last N elements

Exponentially increasing windows

Extensions:

Number of 1s in any last k ($k < N$) elements

Sums of integers in the last N elements

Counting Itemsets

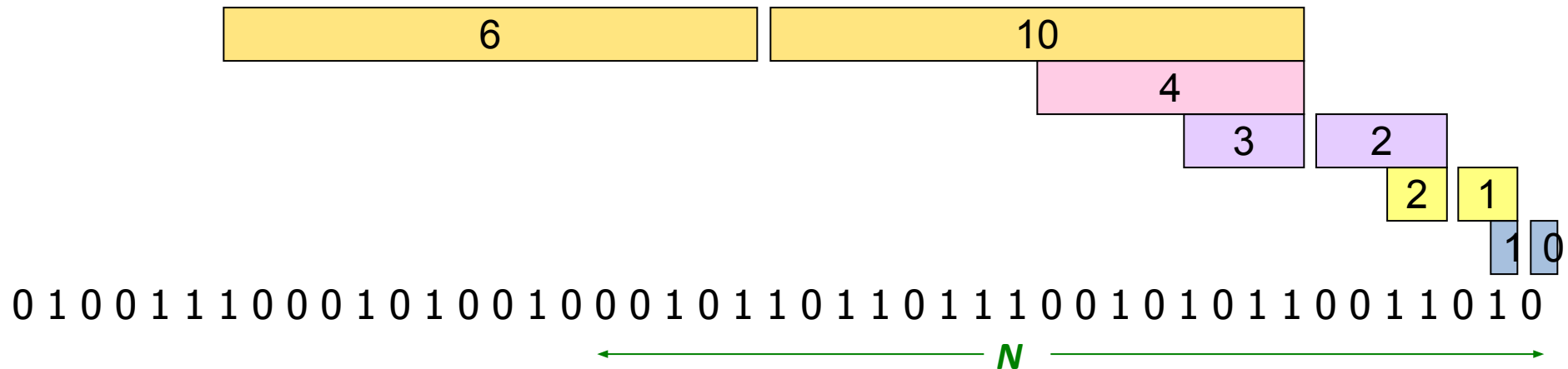
Counting Itemsets

New Problem: Given a stream, which items appear more than s times in the window?

Possible solution: Think of the stream of baskets as one binary stream per item

1 = item present; 0 = not present

Use **DGIM** to estimate counts of 1s for all items



Extensions

In principle, you could count frequent pairs or even larger sets the same way

One stream per itemset

Drawbacks:

Only approximate

Number of itemsets is way too big

Exponentially Decaying Windows

Exponentially decaying windows: A heuristic for selecting likely frequent item(sets)

What are “currently” most popular movies?

Instead of computing the raw count in last N elements

Compute a **smooth aggregation** over the whole stream

If stream is a_1, a_2, \dots and we are taking the sum of the stream, take the answer at time t to be: =

$$\sum_{i=1}^t a_i (1 - c)^{t-i}$$

c is a constant, presumably tiny, like 10^{-6} or 10^{-9}

When new a_{t+1} arrives:

Multiply current sum by $(1-c)$ and add a_{t+1}

Example: Counting Items

If each a_i is an “item” we can compute the **characteristic function** of each possible item x as an Exponentially Decaying Window

That is: $\sum_{i=1}^t \delta_i \cdot (1 - c)^{t-i}$

where $\delta_i=1$ if $a_i=x$, and 0 otherwise

Imagine that for each item x we have a binary stream (1 if x appears, 0 if x does not appear)

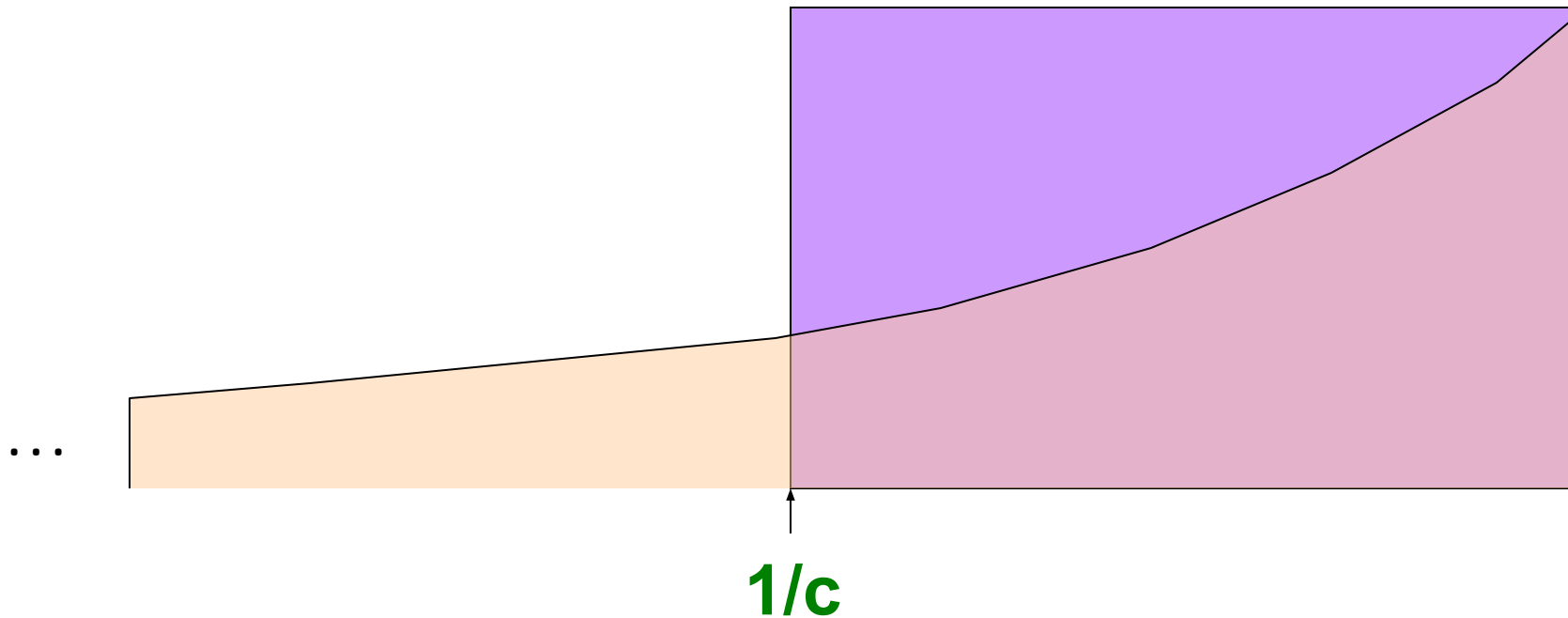
New item x arrives:

Multiply all counts by $(1-c)$

Add $+1$ to count for element x

Call this sum the “weight” of item x

Sliding Versus Decaying Windows



Important property: Sum over all weights
 $\sum_t (1 - c)^t$ is $1/[1 - (1 - c)] = 1/c$

Example: Counting Items

What are “currently” most popular movies?

Suppose we want to find movies of weight $> \frac{1}{2}$

Important property: Sum over all weights $\sum_t (1 - c)^t$ is $1/[1 - (1 - c)] = 1/c$

Thus:

There cannot be more than $2/c$ movies with weight of $\frac{1}{2}$ or more

So, $2/c$ is a limit on the number of movies being counted at any time

Extension to Itemsets

Count (some) itemsets in an E.D.W.

What are currently “hot” itemsets?

Problem: Too many itemsets to keep counts of
all of them in memory

When a basket **B** comes in:

Multiply all counts by **(1-c)**

For uncounted items in **B**, create new count

Add **1** to count of any item in **B** and to any **itemset** contained in **B** that is already being counted

Drop counts $< \frac{1}{2}$

Initiate new counts (next slide)

Initiation of New Counts

Start a count for an itemset $S \subseteq B$ if every proper subset of S had a count prior to arrival of basket B

Intuitively: If all subsets of S are being counted this means they are “frequent/hot” and thus S has a potential to be “hot”

Example:

Start counting $S=\{i, j\}$ iff both i and j were counted prior to seeing B

Start counting $S=\{i, j, k\}$ iff $\{i, j\}$, $\{i, k\}$, and $\{j, k\}$ were all counted prior to seeing B

How many counts do we need?

Counts for single items $< (2/c) \cdot (\text{avg. number of items in a basket})$

Counts for larger itemsets = ??

But we are conservative about starting counts of large sets

If we counted every set we saw, one basket of **20** items would initiate **1M** counts