KTH ROYAL INSTITUTE
OF TECHNOLOGY

Mining Data Streams (Part 2)

Ahmad Al-Shishtawy and Vladimir Vlassov

Acknowledgement: Some of the slides are adopted from the slide provided at http://www.mmds.org for the book “Mining of Massive Datasets” by Jure Leskovec, Anand Rajaraman, Jeff Ullman, Stanford University

Outline

- Part 1 (previous lecture)
  - The stream data model
  - Sampling data in a stream
  - Filtering streams
- Part 2 (this lecture: estimating and counting algorithms on streams)
  - Counting distinct elements
  - Estimating moments
  - Queries over a (long) sliding window: Counting ones
  - Decaying Windows: Counting itemsets

Counting Distinct Elements

Counting Distinct Elements

Problem:

- Data stream consists of elements chosen from a universal 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*)**
  - E.g., if ***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 Flajolet-Martin Works: Heuristic Intuition

Idea: The more distinct elements we see in the stream, the more distinct hash-values we shall see; it becomes more likely that one of these values ends in many 0's.

- $\boldsymbol{h(a)}$ is a sequence of $\mathbf{\log_2} \boldsymbol{N}$ bits
- The probability that $\boldsymbol{h(a)}$ ending in at least $\boldsymbol{r}$ 0's is $\mathbf{2^{-r}}$

Why Flajolet-Martin Works (cont)

- The probability that ***h*(*a*)** ending in at least ***r*** 0's is $\mathbf{2^{-r}}$
  - 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 Flajolet-Martin Works: More formally

- The probability that ***h*(*a*)** ends in at least ***r*** 0's is $\mathbf{2^{-r}}$
- The probability that none of ***m*** distinct elements has 0's tail length at least ***r*** is $(1-2^{-r})^m = (1-2^{-r})^{2^r(m2^{-r})} \approx e^{-m2^{-r}}$
- The probability that we find a tail of length at least ***r*** 0's is

$$(1-e^{-m2^{-r}})$$

  - If ***m* >> 2*^r^***, the probability approaches 1
  - If ***m* << 2*^r^***, the probability approaches 0

Thus $\mathbf{2^{R}}$ (where ***R*** is the largest 0's tail length for any stream element) will almost always be around ***m***

Why It Doesn't Work

- E[2^R] is actually infinite
  - Probability halves when $R \to R+1$, 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 ?
  - **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

Estimating Moments

Definition of 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 (see Section “Counting Distinct Elements”)

1st moment = count of the numbers of elements = length of the stream

- Easy to compute

2nd moment, a.k.a. *surprise number S*, is a measure of how uneven the distribution is $\sum_{i \in A} (m_i)^2$

Example: Surprise Number (2nd Moment)

Stream of length 100

11 distinct values

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

$$S = 10^2 + 10 \cdot 9^2 = 910$$

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

$$S = 90^2 + 10 \cdot 1^2 = 8110$$

AMS: Alon-Matias-Szegedy Method

- Name refers to the inventors: **A**lon, **M**atias, and **S**zegedy

- **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.element*** and ***X.value***
    - ***X.element*** corresponds to the item ***i***
    - ***X.value*** corresponds to the **count** of item ***i*** (i.e. the number of occurrences of the item ***i***)
  - Note this requires a count in main memory, so number of ***X***s is limited
- **Our goal is to compute** $S = \sum_i (m_i)^2$

where m_i is the number of occurrences(the count) of the item i

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

One Random Variable (X)

How to set *X.value* and *X.element*?

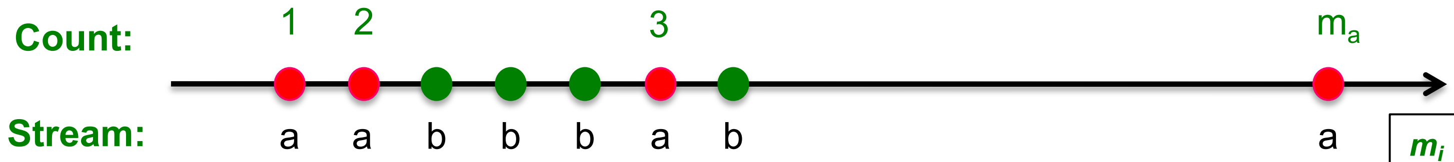
- Assume stream has length ***n*** (we relax this later)
- Pick some random time ***t*** (***t<n***) to start, so that any time is equally likely
- Let at time ***t*** the stream have item ***i***. **We set *X.elelment* = *i***
- Then we maintain count ***c*** (***X.value* = *c***) of the number of ***i*s** in the stream starting from the chosen time ***t***
- **Then the estimate of the 2nd moment** $S = \sum_i (m_i)^2$ **is:**

$$S = f(X) = n \cdot (2 \cdot c - 1)$$

  – Note, we will keep track of multiple ***X*s**, ($X_1, X_2, \ldots X_k$) and our final estimate will be the average of counts of ***X*s**

$$S = 1/k \sum_j^k f(X_j)$$

Expectation Analysis



m_i … total count of item i in the stream (we assume stream of length n)

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

c_t … number of times item at time t appears from time t onwards:

- $c_1 = m_a$ – from time $t = 1$, item a appears m_a times
- $c_2 = m_a - 1$ – from time $t = 2$, item a appears m_a-1 times
- $c_3 = m_b$ – from time $t = 3$, item b appears m_b times

Note that $1 + 3 + 5 + \cdots + (2m_i - 1) = m_i^2$

$$E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \frac{1}{n}\sum_i n(1 + 3 + 5 + \ldots + 2m_i - 1) = \frac{1}{n}\sum_i n(m_i)^2 = \sum_i (m_i)^2 = S$$

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)

Higher-Order Moments

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

- For $\boldsymbol{k}$**=2** we used $\boldsymbol{n\,(2\cdot c - 1)}$
- For $\boldsymbol{k}$**=3** we use: $\boldsymbol{n\,(3\cdot c^2 - 3c + 1)}$ (where $\boldsymbol{c}$**=*X*.value**)

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 cannot have one random variable ***X*** for each start-time, and must throw out some start-times (some ***X***s) as we read the stream (as time goes on).
   – **Objective**: Each starting time ***t*** is selected with probability ***k*/*n***.

Streams Never End: Fixups (cont)

- **Objective**: Each starting time ***t*** is selected with probability ***k*/*n***.
- **Solution:**
  - 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

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Queries over a (long) Sliding Window: Counting Ones in a 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 to answer queries, such as how many times **X** has been sold in the last ***k*** sales

Sliding Window on a Single Stream

Window of *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 →

Example: Averages

- Stream of integers, window of size ***N***.
- **Standing query**: ***What is the average of the integers in the window?***
- For the first ***N*** inputs, sum and count to get the average.
- Afterward, when a new input ***i*** arrives, change the average by adding **(*i* - *j*)/*N***, where ***j*** is the oldest integer in the window before ***i*** arrived.
- **Good**: **O(1)** time per input.
- **Bad**: Requires the entire window in main memory.

Approximate Counting

- To compute an exact sum or count of the elements in a window requires space at least as the window itself.
- But if you are willing to accept an approximation, you can use much less space.
- We'll consider ***the simple case of counting bits***, which includes counting elements of a certain type as a special case (1: appears; 0: not).
- Sums are a fairly straightforward extension.

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 $k \le N$

Obvious solution:

Store the most recent ***N*** bits

- When new bit comes in, discard the ***N*+1st** bit

0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 $\boxed{1\ 1\ 0\ 1\ 1\ 0}$ Suppose N=6

← Past Future →

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**

0 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 ~~1 1 0 1 1 0~~

← Past   Future →

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**

N

0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0

Past — Future

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

- Name refers to the inventors: **D**atar, **G**ionis, **I**ndyk, **M**otwani

DGIM solution that does not assume uniformity

Store $\boldsymbol{O}(\log_2 \boldsymbol{N})$ bits per stream

- where ***N*** is the window size

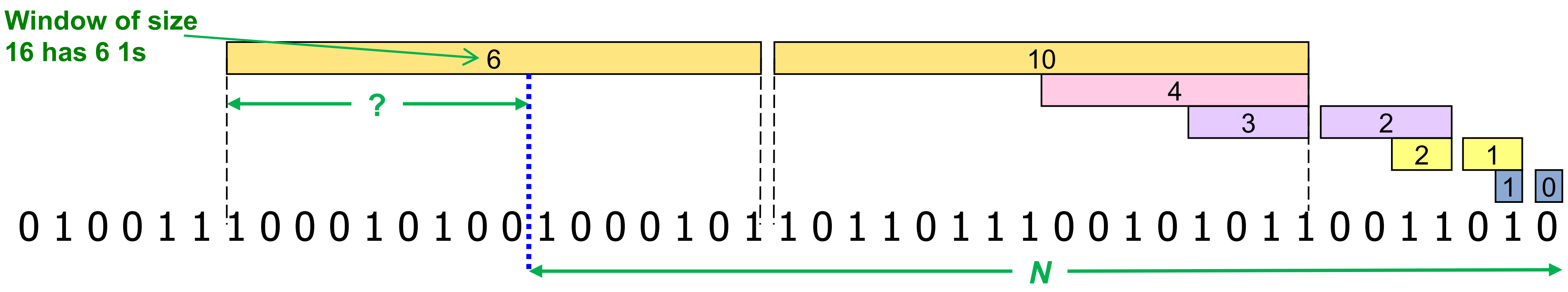
Solution gives approximate answer, never off by more than 50%

- 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***, i.e., count = (? + 10 + 2 + 1)

Pros: 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

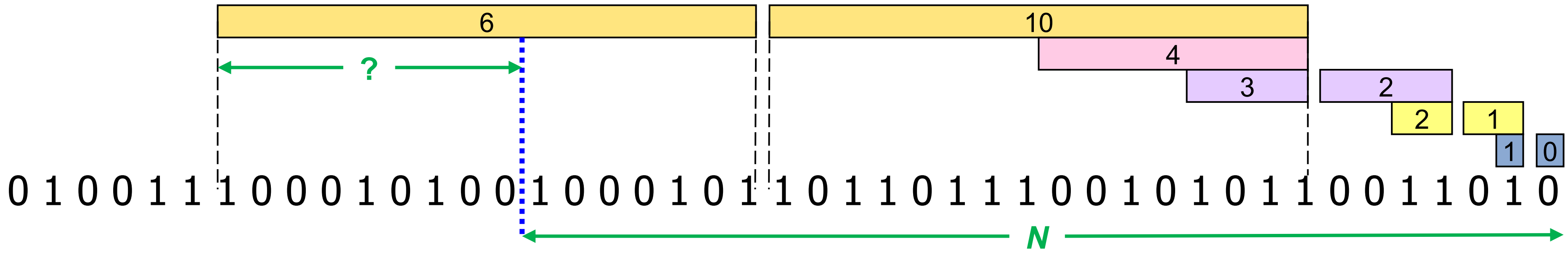
Error in count no greater than the number of **1s** in the "**unknown**" area

Cons: 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

1001010110001011 0 1010101010101011 0 10101010101111 0 1010101 110101 000 101 1 00 1 0

N

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 $\mathbf{O(\log_2 N)}$ bits

DGIM: Buckets

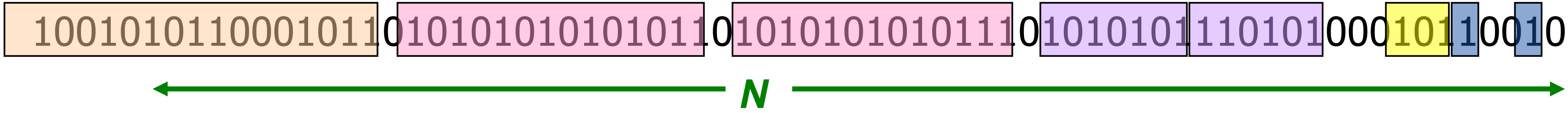
A ***bucket*** in the DGIM method is a record consisting of:

- **(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:

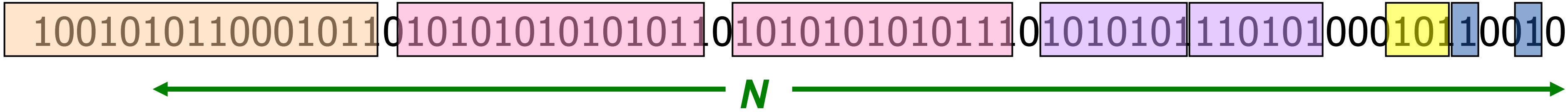
Number of **1s** must be a power of **2**

- That explains the **O(log log N)** in **(B) above**

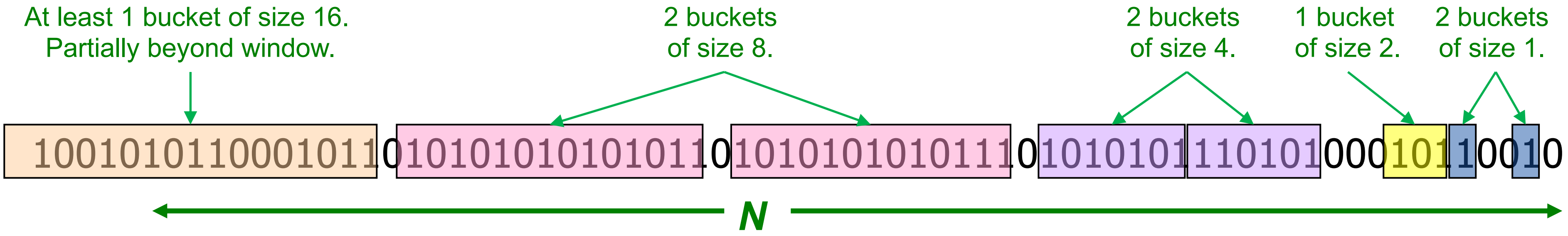


Representing a Stream by Buckets

- 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

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 (cont)

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:

10010101100010110101010101010110101010101010111010101011101010001011001 0

Bit of value 1 arrives and pit in a new bucket of size 1

001010110001011010101010101011010101010101011101010101110101000101100101

Two older buckets of size 1 (blue buckets) get merged into one bucket of size 2 (yellow bucket)

001010110001011010101010101011010101010101011101010101110101000101100101

Next bit 1 arrives, new blue bucket is created for it, then 0 comes, then 1:

010110001011010101010101011010101010101011101010101110101000101100101101

Buckets get merged…

010110001011010101010101011010101010101011101010101110101000101100101101

State of the buckets after merging

010110001011010101010101011010101010101011101010101110101000101100101101

How to Query?

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

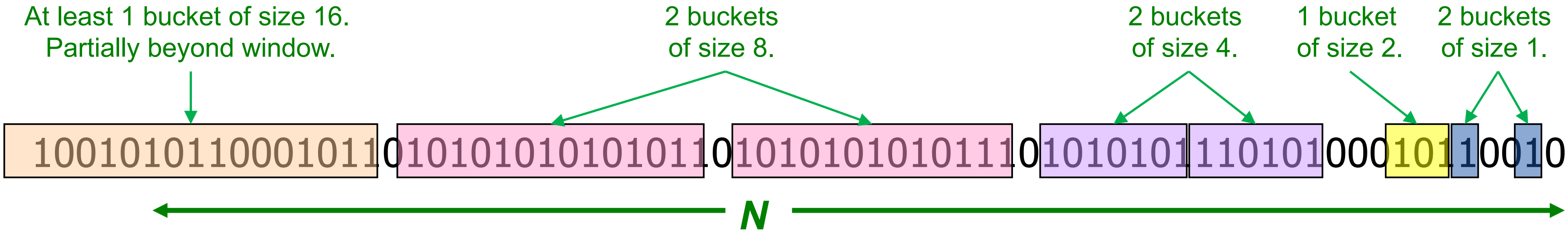
1. **Sum the sizes of all buckets but the last**

   (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

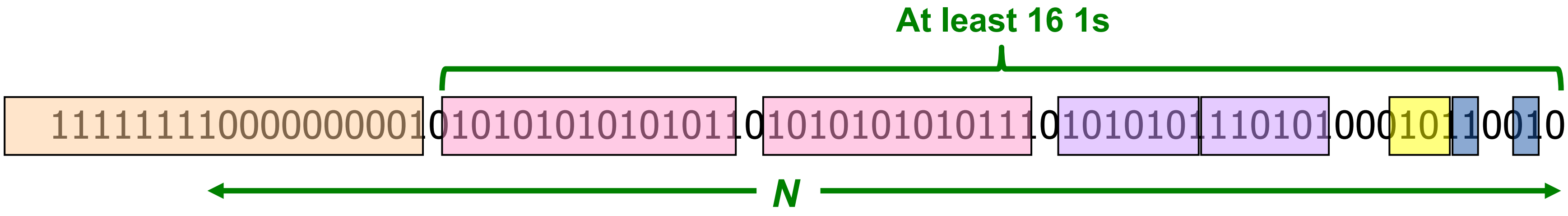


Error Bound: Proof

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

- Suppose the last bucket has size $\mathbf{2^r}$
- Then by assuming $\mathbf{2^{r-1}}$ (i.e., half) of its **1s** are still within the window, we make an error of at most $\mathbf{2^{r-1}}$
- Since there is at least one bucket of each of the sizes less than $\mathbf{2^r}$, the true sum is at least $\mathbf{1 + 2 + 4 + .. + 2^{r-1} = 2^r - 1}$

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

Decaying Windows: Counting itemsets

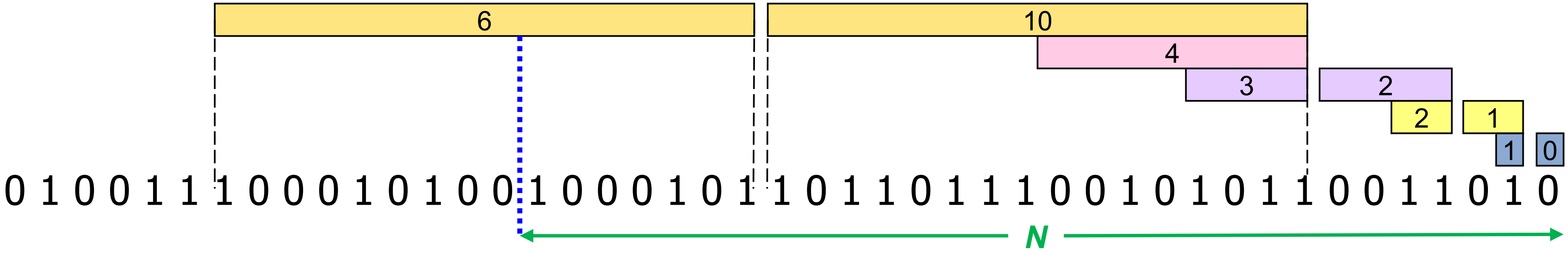
Counting Itemsets

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 **1**s 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 $a_1, a_2, \ldots$ 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

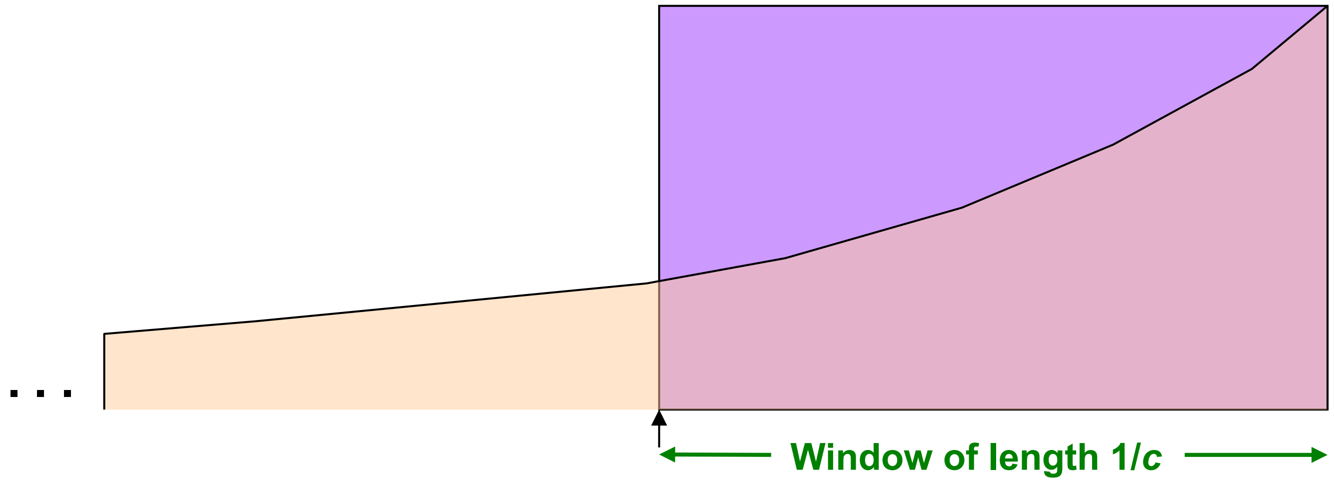
If each a_i is an "item" we can compute the ***characteristic function*** of each possible item x as an Exponentially Decaying Window

- 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 $\mathbf{1/[1-(1-c)] = 1/c}$

Example: Counting Items

What are "currently" most popular movies?

Suppose we want to find movies of weight > ½

- **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 ½ 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 Enterprise Data Warehouse

- **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 < ½**
- 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)·(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